FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA

Análisis de la efectividad del diseño detallado y de las revisiones personales del Team Software Process

Proyecto de Grado

Ana Menéndez David Dacoli Matias Nieva

Tutor
Diego Vallespir

Montevideo, Uruguay
Marzo, 2016

Índice general

Índice de figuras

Índice de cuadros

Resumen

El software se ha convertido en una herramienta esencial de toda empresa u organización, e incluso de nuestros propios hogares. Por este motivo, la calidad debe ser un requisito para su desarrollo. Para ello, es preciso la aplicación de un modelo de proceso que tome en cuenta dicho requisito, ya que la calidad del producto de software depende directamente de la calidad del proceso que lo generó.

Este trabajo describe un proceso para el desarrollo de software denominado *Team Software Process* (TSP). Siendo uno de los componentes fundamentales de la calidad la ausencia de defectos, TSP incluye dos fases de revisiones personales, una luego de la fase de diseño y otra luego de la fase de código, en donde los ingenieros revisan sus productos de software con la finalidad de encontrar y remover defectos eficientemente.

El objetivo general de este trabajo consiste en estudiar el modelo de proceso TSP y la efectividad del diseño detallado y de las revisiones personales para obtener un producto de software de calidad. Teniendo en cuenta esto, el objetivo particular se basa en evaluar si el tiempo dedicado al diseño detallado y a las revisiones personales, son un buen indicador de la calidad del producto de software.

Para el análisis final se utilizaron dos medidas de calidad, el *Process Quality Index* reducido (PQI*) basado en los tiempos dedicados en el diseño detallado y en las revisiones personales, y el *Process Yield* que es el porcentaje de defectos que fueron inyectados y removidos antes de la fase de compilación.

La base de datos con la que se trabajó fue proporcionada por el *Software Engineering Institute* (SEI). La misma contiene datos recolectados por varios equipos de desarrollo de software que aplicaron el proceso TSP.

Los resultados obtenidos sugieren que el tiempo dedicado al diseño detallado y a las revisiones personales influyen en la calidad del producto final. Observamos que, para un PQI* mayor a 0,45 se obtienen productos de software que tienden a cero defecto (medido a través de *Process Yield*).

Capítulo 1

Introducción

El software juega un rol fundamental en nuestras vidas por lo que la calidad debe ser un requisito para su desarrollo. La calidad es el conjunto de características de un producto que satisfacen las necesidades de los clientes y, en consecuencia, hacen satisfactorio el producto. También consiste en no tener deficiencias en el producto o en el proceso [9]. Para ello, es preciso la aplicación de un modelo de proceso que tome en cuenta la calidad, ya que la calidad del producto de software depende directamente de la calidad del proceso que lo generó [10].

Se estudió un modelo de proceso para el desarrollo de productos de software denominado *Team Software Process* (TSP). Incluye dos fases de revisiones personales, una luego de la fase de diseño y otra luego de la fase de código, en donde los ingenieros revisan sus productos de software con la finalidad de encontrar y remover defectos eficientemente. TSP cuenta con un conjunto de elementos para la gestión de la calidad que consiste en realizar un plan de calidad, identificar los problemas de calidad, y la búsqueda y prevención de los mismos.

También se llevó adelante un análisis de datos, los que fueron recolectados por varios equipos de TSP. Dado que uno de los componentes fundamentales de la calidad es la ausencia de defectos en el producto final, el análisis consistió en estudiar si el tiempo dedicado al diseño detallado y a las revisiones personales podrían ser un indicador razonable de la calidad del producto de software.

En la sección 1.1 se describe la motivación del proyecto. En la sección 1.2 se explica los objetivos del mismo. En la sección 1.3 se detalla el trabajo realizado. Por último, en la sección 1.4 se presenta la estructura del documento.

1.1. Motivación

El software se ha convertido en una herramienta esencial de toda empresa u organización, e incluso de nuestros propios hogares. Se utiliza en la rama de la medicina, la investigación científica, el comercio electrónico, la industria automovilística, por mencionar algunas de ellas, y se ha vuelto cada vez más complejo su desarrollo y gestión. Un producto de software es un conjunto completo de programas, procedimientos, así como documentación y datos asociados. Debido a que los errores humanos pueden introducir defectos en el producto de software, y los defectos causar fallas, a lo largo del tiempo se han estudiado métodos y procedimientos que permitan desarrollar productos de software de calidad. Uno de los componentes fundamentales de la calidad es la ausencia de defectos. Es importante destacar que no existe un método general para la prevención de la inyección de defectos.

En el pasado, la estrategia de calidad de la mayoría de las organizaciones industriales estaba

basada casi totalmente en el *testing*. Existían departamentos de calidad que se enfocaban en la búsqueda y remoción de defectos una vez que el producto había sido desarrollado. Esta estrategia es cara, consume mucho tiempo y es inefectiva. No fue sino hasta los años setentas y ochentas que W. Edwards Deming and J.M. Juran convencieron a la industria de los Estados Unidos a enfocarse en la mejora del trabajo que realiza cada persona. Siguiendo este camino trazado por Deming y Juran, en 1976 Michael Fagan introduce las inspecciones de software, y en 1987 surge el *Capability Maturity Model* (CMM) desarrollado por el *Software Engineering Institute* (SEI) de la Universidad de Carnegie Mellon, el cual es un modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software [11].

A pesar de los nuevos principios de calidad que han sido adoptados por las organizaciones en general, muchas aún persisten en confiar en el *testing* como su estrategia de gestión de la calidad. Solo muy pocos equipos de trabajo saben cómo gestionar la calidad de sus productos [12].

Así es como surge también un nuevo modelo de proceso para el desarrollo de productos de software denominado TSP. Fue desarrollado por el SEI y su primera versión data del año 1996. El mismo incluye dos fases de revisiones personales, una luego de la fase de diseño y otra luego de la fase de código, en donde los ingenieros revisan sus productos de software con la finalidad de encontrar y remover defectos eficientemente.

Es interesante conocer si estas revisiones personales que propone TSP influyen en la obtención de productos de software de calidad.

1.2. Objetivos

El objetivo general de este trabajo consiste en estudiar el modelo de proceso TSP y la efectividad del diseño detallado y de las revisiones personales para obtener un producto de software de calidad. Teniendo en cuenta esto, el objetivo particular se basa en evaluar si el tiempo dedicado al diseño detallado y a las revisiones personales, son un buen indicador de la calidad del producto de software.

A continuación se detallan los objetivos:

1. Objetivo uno: Estudiar el proceso *Team Software Process* (TSP), en qué consiste, cómo se usa y la recolección de los datos generados a través del uso de la herramienta *Process Dashboard*.

2. Objetivo dos: Estudiar una base de datos que contiene información recolectada por varios equipos de desarrollo de software que aplicaron el proceso TSP.

3. Objetivo tres: Analizar si el tiempo dedicado al diseño detallado y a las revisiones personales podría ser un indicador de la calidad del producto, a partir de consultas a los datos.

1.3. Trabajo Realizado

Se realizó un estudio del modelo de proceso TSP, haciendo énfasis en su estructura y *framework*, en las fases de diseño detallado, revisión de diseño, código, revisión de código, compilación y *testing*, y en la gestión de la calidad que brinda.

Se estudió la herramienta *Process Dashboard*, su funcionamiento y cómo los datos recolectados son registrados en una base de datos.

A partir de una base de datos que contiene información generada por la aplicación *Process Dashboard*, se estudió su estructura y contenido. Se realizaron consultas básicas para obtener

información y así poder caracterizar los datos. Se utilizó el lenguaje R con código SQL embebido para la codificación de las consultas básicas y de análisis.

Para el estudio de los datos se utilizaron dos medidas de calidad, el *Process Quality Index* reducido (PQI*) basado en los tiempos dedicados en las revisiones personales y en el diseño detallado, y el *Process Yield* que es el porcentaje de defectos que fueron inyectados y removidos antes de la fase de compilación.

En los resultados observamos que, para un PQI* mayor a 0,45 se obtienen productos de software que tienden a cero defecto (medido a través de *Process Yield*).

1.4. Estructura del documento

El documento dispone de cinco capítulos y cuatro apéndices.

El capítulo *Team Software Process*: describe los conceptos fundamentales del proceso y *framework* para las mediciones, una breve descripción del *Personal Software Process* (PSP) y la gestión integral de calidad.

En el capítulo *Process Dashboard*: se detalla la herramienta que apoya el proceso, las ventajas de su uso y cómo registra los datos. Además se examina la base de datos que genera la herramienta.

En el capítulo *Descripción de Datos*: se presenta la caracterización de los datos que fueron utilizados para la realización del trabajo.

En el capítulo *Análisis de Datos*: se desarrolla el análisis del tiempo dedicado al diseño detallado y a las revisiones personales a través de dos enfoques.

Por último, en el capítulo *Conclusiones*: se presentan las conclusiones del proyecto y el trabajo a futuro.

También contiene cuatro apéndices. En el Apéndice A *Manual de Uso de Process Dasboard*: se presenta el uso básico de la herramienta y el registro de datos.

El Apéndice B *Manual de Configuración del Ambiente de Trabajo*: se detallan las herramientas necesarias para trabajar con la base de datos y ejecutar las consultas realizadas.

El Apéndice C *Consultas Básicas*: contiene las consultas en el lenguaje R para generar las gráficas del capítulo de Descripción de Datos.

El Apéndice D *Consultas de Análisis*: contiene las consultas en el lenguaje R para generar las gráficas de los enfoques aplicados del capítulo de Análisis de Datos.

Capítulo 2

Team Software Process

El *Team Software Process* (TSP) es un modelo de proceso para el desarrollo de software focalizado en equipos de trabajo, y su primera versión data del año 1996. Según escribió James W. Over, director del Programa TSP del *Software Engineering Institue* (SEI) [14], TSP es un proceso específicamente diseñado para equipos de software con el propósito de ayudarlos a:

1. Planificar su trabajo
2. Negociar sus compromisos con la gerencia
3. Llegar a conclusiones exitosas mediante la gestión y el seguimiento de los proyectos
4. Producir productos de calidad en menor tiempo
5. Lograr su mejor rendimiento sin llegar a un final tipo "*death march*"

Existen cuatro conceptos fundamentales para entender TSP:

1. *Self-directed Teams y Coaching*
2. *Personal Software Process* (PSP)
3. Descripción del Proceso y *Framework* para las mediciones
4. Gestión integral de calidad

En las secciones 2.1 a 2.4 se especifican los puntos mencionados, haciendo énfasis en PSP y en la descripción del Proceso y *Framework* para las mediciones.

2.1. Self-directed Teams y Coaching

Mientras que en un equipo tradicional un líder es quien dirige y realiza un seguimiento sobre el trabajo del equipo, el rol de líder en TSP se concentra en la formación del equipo, motivación del mismo y algunas tareas de administración, dejando a cada integrante participar de la planificación, la gestión y el seguimiento de su propio trabajo. Además, un *coach* ayuda al equipo y a sus integrantes a ser más performantes, siendo un referente en el uso del proceso. A esta clase de funcionamiento de un equipo se denomina "*Self-directed Team*". Los equipos son integrados por al menos tres ingenieros y no más de quince. En el libro "*The Team Software Process Body of Knowledge* (TSP BOK)"[13] se describe que los roles de gestión estándar de TSP son:

- *Planning manager*

- *Process manager*
- *Quality manager*
- *Support manager*
- *Customer interface manager*
- *Design manager*
- *Implementation manager*
- *Test manager*

Los cuatro primeros forman parte del conjunto de roles de Gestión de Proyectos, mientras que los cuatro últimos forman parte de los roles Técnicos.

2.2. Personal Software Process

TSP hace uso del *Personal Software Process* (PSP), proceso que aplica cada miembro del equipo que participa en el desarrollo del producto. Es un proceso personal para el desarrollo de software que incluye pasos bien definidos, *scripts*, medidas, *forms*, *checklists* y estándares [11], que ofrecen a los ingenieros pautas para planificar, medir y gestionar su trabajo.

Por lo general, para el desarrollo de un producto de software, el proceso escogido está conformado por fases y establece una metodología de trabajo a llevar adelante. Existen fases asociadas al análisis de requerimientos, al diseño y arquitectura, a la implementación y al *testing* para verificar y corregir defectos. Al dejar el *testing* para el final, este tipo de procesos aplicados provocan horas de retrabajo en corregir defectos que se cometieron en fases tempranas del proceso, lo que generalmente resulta costoso. Por este motivo, PSP hace énfasis en identificar los defectos durante el proceso y no únicamente en la fase de *testing*. Existen dos grandes fases que PSP agrega al proceso tradicional mencionado anteriormente, que son la revisión de diseño y la revisión de código. En dichas fases, el ingeniero tiene que realizar una revisión personal de su diseño y de su codificación utilizando *checklists* de apoyo. También registra los defectos que fueron encontrados y removidos.

Según Watts S. Humphrey, el PSP se basa en los siguientes principios de planificación y de calidad:

- Cada ingeniero es diferente, por lo que para ser más eficaces deben planificar su trabajo basándose en sus propios datos personales.
- Para mejorar constantemente su rendimiento, los ingenieros deben utilizar procesos de medida bien definidos y personalizados.
- Para producir productos de calidad, los ingenieros deben sentirse personalmente responsables de la calidad de sus productos. Los productos de calidad superiores no son producidos por error, por lo que los ingenieros deben esforzarse en hacer trabajo de calidad.
- Cuesta menos encontrar y corregir defectos en etapas tempranas que tardías del proceso.
- Es más eficaz prevenir los defectos que encontrarlos y solucionarlos.
- La forma correcta es siempre la forma más rápida y barata para el trabajo.

Además, cada integrante lleva adelante los siguientes pasos para hacer un seguimiento personal de su trabajo. Ver figura 2.1:

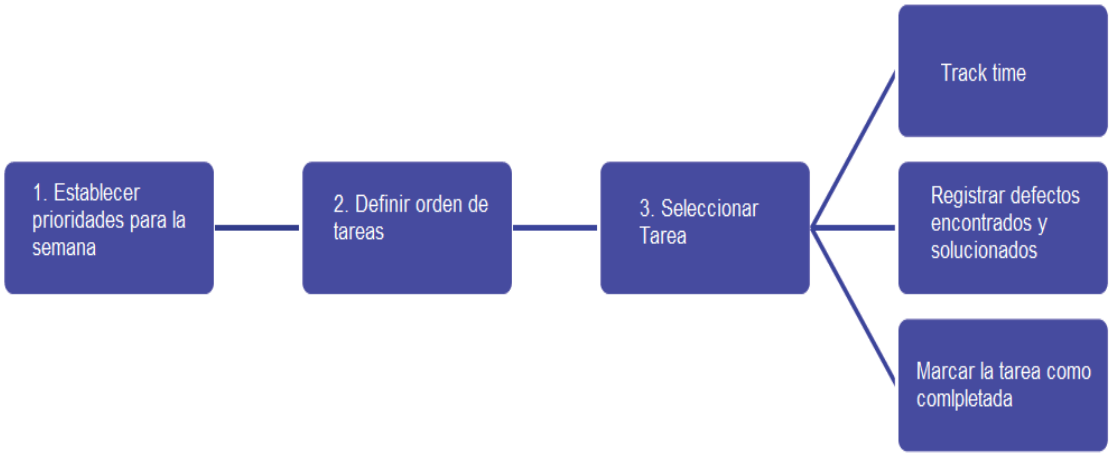

Figura 2.1: Seguimiento Personal

Cada usuario debe priorizar tareas, ordenarlas, seleccionar una y completarla haciendo el registro de defectos inyectados y removidos durante la realización de la tarea, el tiempo exacto utilizado para completarla y los tamaños correspondientes asociados a la tarea.

2.3. Descripción del Proceso y Framework para las mediciones

TSP posee siete elementos básicos, los cinco elementos básicos de PSP que son *scripts*, *forms*, medidas, *checklists* y estándares, y dos restantes exclusivos de TSP que son las especificaciones y guías o pautas. Las especificaciones proporcionan una descripción clara e inequívoca del producto o tarea, y los criterios de cómo deberían ser evaluados los mismos. Las guías o pautas se utilizan para describir las reglas o estrategias recomendadas, en forma de secuencia, que debe seguir la persona.

La estructura de TSP está dividida en ciclos, ver la figura 2.2. Cada ciclo comienza con un lanzamiento (*launch*) o re-lanzamiento (*re-launch*) y finaliza con un Postmortem. El contenido de un ciclo es determinado por el equipo.

Un *launch* cumple con la función de poder colocar a los profesionales a cargo de su propio trabajo personal, proveer un ambiente que apoye a la excelencia individual, habilitar a los equipos a producir procesos y planes que mejor se ajusten a sus necesidades y realizar planes y compromisos para el desarrollo del proyecto donde puedan participar todos los miembros del equipo. Básicamente produce los artefactos de planificación necesarios como lo son las metas, los roles, las estimaciones, los hitos, los planes para las tareas, para la calidad y para la mitigación de riesgos, entre otras cosas.

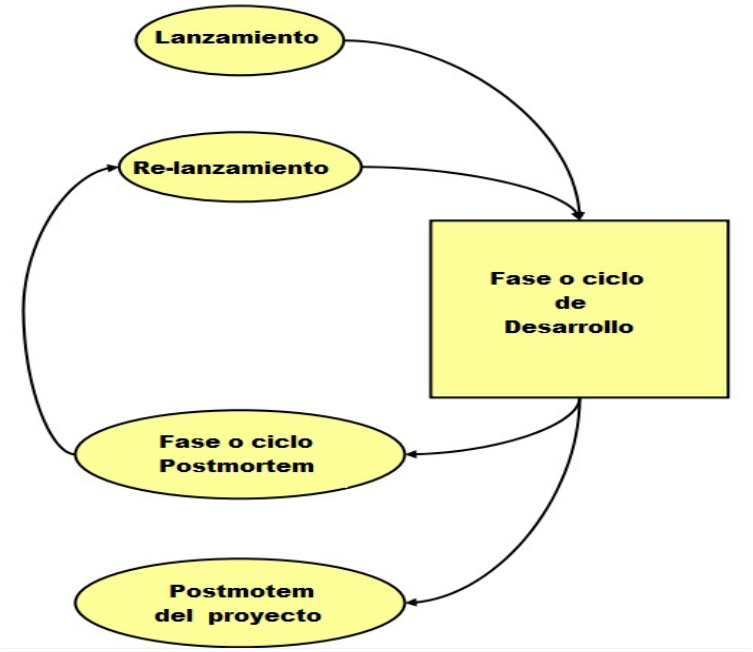

Figura 2.2: Estructura del Proceso TSP

El proceso de *Launch* se divide en nueve reuniones (*meetings*) que se lleva a cabo en un período de cuatro días, ver 2.3:

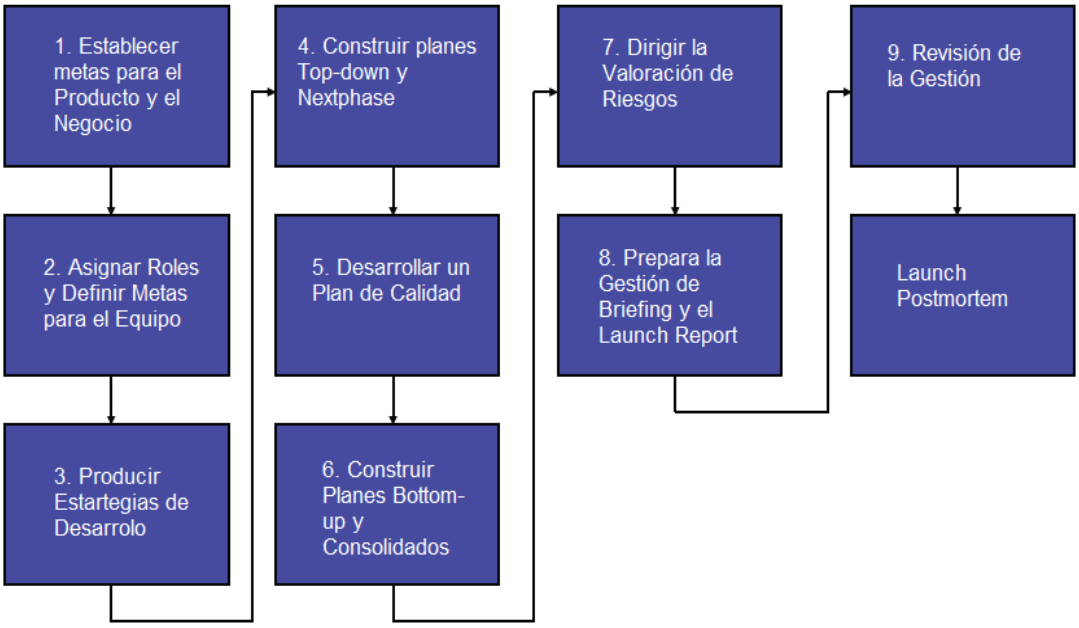

Figura 2.3: Proceso del Launch

Cada una de las nueve reuniones mencionadas anteriormente tiene un *script* específico que describe detalladamente las actividades a seguir.

Un *re-launch* es prácticamente lo mismo que el *launch* con la excepción de que es realizado por un equipo que ya ha completado un *launch* inicial del mismo proyecto.

Una fase representa una parte del ciclo de vida del desarrollo, como lo puede ser la fase de implementación. Tiene como entrada la salida del *launch*, esto es: estimaciones, planes, procesos, compromisos, etc. Como salida del mismo se obtienen productos de trabajo, estados, métricas y resultados.

La fase o ciclo postmortem incluye solamente la información del trabajo completado durante fases de proyectos o ciclos realizados con anterioridad. Además analiza los datos recolectados a través de la agenda, las horas dedicadas en tareas, estimación, calidad y proceso.

Una vez que el equipo tiene definido el producto a construir, se debe definir cómo hacerlo. Para ello se establece un proceso de desarrollo de TSP como se muestra en la figura 2.4

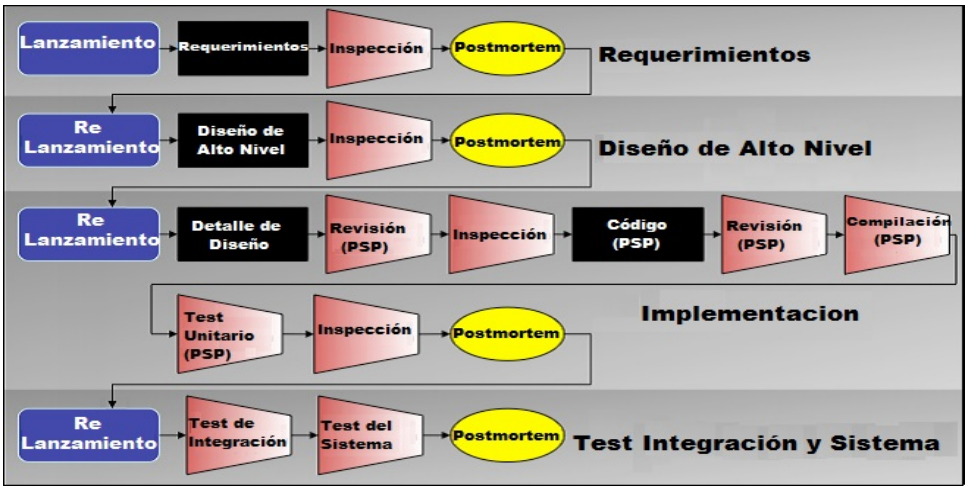

Figura 2.4: Proceso de desarrollo de TSP

Se puede observar que existen cuatro elementos del proceso diferenciados por colores:

- En azul de Planificación
- En negro de Desarrollo
- En rojo Filtros donde ubicar posibles defectos
- En amarillo de Postmortem

En la figura 2.5 se puede observar la fase de implementación de TSP completa:

Figura 2.5: Fase de Implementación de TSP

El *framework* de mediciones utilizado por TSP y PSP se divide en cuatro medidas fundamentales [11]. Respecto a sus unidades, la agenda puede definirse en horas, semanas o meses, pero no en días; el tamaño en puntos de función o líneas de código (LOCS); el tiempo en minutos; y los defectos en la fase en que fueron inyectados y removidos.

1. Agenda
2. Tamaño
3. Tiempo
4. Defectos

2.3.1. Medidas de Tiempo

Los ingenieros registran el tiempo de inicio de una tarea y el tiempo de finalización de la misma, tomando en cuenta las interrupciones que ocurrieron durante su realización. De esta manera se evita el registro de tiempos imprecisos, pudiendo calcular de manera más exacta el esfuerzo requerido para realizar las tareas del proceso. El registro del tiempo se realiza en minutos. En la tabla 2.1 se muestra un ejemplo del registro.

Estos tiempos podrán ser utilizados para comparar con los tiempos planificados, que fueron estimados durante la planificación de cada tarea.

2.3.2. Medidas de Tamaño

Para desarrollar un producto se debe estimar el tamaño del mismo, que incide directamente en el tiempo que se utiliza para llevarlo a cabo. Luego se registran distintas métricas de tamaño de los productos desarrollados que más tarde serán utilizadas para futuras estimaciones de

tamaño. En PSP la principal medida de tamaño es Líneas de Código (LOC), pero puede utilizarse cualquier otra medida que tenga una correlación entre el tiempo de desarrollo y el tamaño del producto.

Para realizar un seguimiento de cómo cambia el tamaño de un programa durante su desarrollo, es importante tener en cuenta las siguientes categorías de LOC:

- Base: es la cantidad de LOC del producto originalmente, es decir antes de que sea modificado.
- Added: es la cantidad de LOC agregadas a un código existente.
- Modified: es la cantidad de LOC Base que han sido modificadas en el código original.
- Deleted: es la cantidad de LOC Base que han sido eliminadas del código.
- Added and Modified: es la cantidad de LOC que fue agregado o modificado y que se utiliza para hacer estimaciones de tamaño.
- Total: es la cantidad de LOC de un programa.

A continuación se muestra un ejemplo del cálculo de la categoría Total.

```
Tenemos el caso de un programa que utilizó 100.000 LOC (100 KLOC) donde :
12.000 LOC pertenecientes a la categoría Deleted
23.000 LOC pertenecientes a la categoría Added
5.000  LOC pertenecientes a la categoría Modified
3.000  LOC pertenecientes a la categoría Reused

Entonces la cantidad de LOC pertenecientes a New and Changed será el total de Added mas
el total de Modified 28.000 LOC (23.000 + 5.000).

El tamaño total de un producto será el siguiente:
Total = Base - Deleted + Added + Reused = 114.000 LOC (100.000 - 12.000 + 23.000 + 3.000.)
```

Fecha	**Inicio**	**Fin**	**Tiempo de Interrupción**	**Tiempo Delta**	**Actividad**	**Comentarios**
9/9	9:00	9:50		50	Planeación	
	12:40	1:18		38	Diseño	
	2:45	3:53	10	58	Diseño	Teléfono
	6:25	7:45	6+5	80	Codificación	Baño, tomo café
10/9	11:06	12:19		62	Codificación	
11/9	9:00	9:50		50	Codificación	
	1:15	2:35	3+8	69	Codificación	Consulta libro
	4:18	5:11	25	28	Prueba	Reunión con mi jefe
12/9	6:42	9:04	10+6+12	114	Prueba	Teléfono, baño, teléfono
13/9	9:00	9:50		50	Prueba	
	12:33	1:16		38	Postmortem	

Cuadro 2.1: Ejemplo de registros de tiempo

2.3.3. Medidas de Calidad

El foco principal sobre la calidad son los defectos. Para gestionar los defectos se necesitan datos de los defectos que fueron inyectados y removidos:

- Fase de cuándo fue inyectado
- Fase de cuándo fue removido
- El tiempo que llevó su corrección.

Un defecto puede ser un error ortográfico, un error de puntuación, o una declaración de programa incorrecto. Los defectos pueden estar en los programas, en los diseños, o incluso en los requisitos, especificaciones, u otra documentación, por lo que pueden encontrarse en cualquier fase del proceso.

A continuación se describen algunas medidas:

- Densidad de defectos: es una medida que se utiliza para todo el proceso de desarrollo o para una fase específica. Indica la cantidad de defectos para la categoría *Added and Modified* en KLOC de un programa.
- *Yield*: es una medida que se puede calcular de forma parcial o de todo el proceso.

  Phase Yield: es una medida que se utiliza para conocer el porcentaje de defectos que se encuentran y remueven en una fase determinada. Se calcula dividiendo el total de defectos que se encontraron en la fase sobre el total de defectos con el que se llegó a la misma.

  Process Yield: a diferencia de la anterior, esta medida calcula el porcentaje de defectos removidos antes de la fase de compilación. El valor recomendado es de al menos un 70 %.

FASE	**DEFECTOS INYECTADOS**	**DEFECTOS REMOVIDOS**	**DEFECTOS INICIO FASE**	**YIELD DE LA FASE**
Diseño Detallado	26	0	0	
Revisión de Diseño	0	11	26	42.3 %
Código	39	0	15	
Revisión de Código	0	28	54	51.9 %
Compilación	0	12	26	46.2 %
Testing Unitario	0	7	14	50.0 %
Después del Testing Unitario	0	7	7	
Total	65	65		

Cuadro 2.2: Ejemplo de cálculo de Phase Yield.

Como se observa en la figura 2.2, el total de defectos inyectados es de sesenta y cinco. En la fase de revisión de diseño, la cantidad de defectos presentes son veintiséis y provienen de la fase anterior. El total de defectos removidos en dicha fase es de once, por lo que el *Phase Yield* de la fase será 42,3 % (11/26 * 100).

Si se quisiera además calcular el *Process Yield*, se debe tomar la cantidad de defectos inyectados antes de la fase de Compilación que son un total de sesenta y cinco, y la cantidad de defectos removidos antes de la fase de Compilación que son un total de treinta y nueve. Entonces, el *Process Yield* será de 60 % (39/65 * 100).

Existen otras medidas que hacen al proceso, como Tasa de Tiempo de Desarrollo, Tasa de Defectos o Defectos por hora, que no son objeto del presente trabajo.

2.4. Gestión integral de calidad

2.4.1. Gestión de Calidad en TSP

Para realizar una gestión de calidad adecuada los equipos TSP deben establecer ciertas medidas, metas y un plan para cumplirlas. Además, se deberán definir medidas de avance respecto al plan y medidas de contención para cuando este no se cumpla. Entonces, los elementos de la gestión de la calidad TSP consisten en hacer un plan de calidad, identificar los problemas de calidad, y la búsqueda y prevención de los mismos [12].

2.4.2. Plan de Calidad

Durante el lanzamiento, el equipo TSP realiza el Plan de Calidad. Basándose en estimaciones de tamaño del producto y tasas de inyección de defectos, se estima en qué fases podría inyectarse un defecto. Cuando los equipos no tienen datos históricos para respaldarse, pueden hacer uso de guías provistas por TSP como referencia. Una vez que los ingenieros estimaron los defectos que podrán ser inyectados, estiman los que podrán ser removidos, nuevamente a partir de datos históricos o directrices de calidad TSP. Estas estimaciones de remoción se basan en el *Yield* para cada fase.

Una vez que se han hecho las estimaciones de inyección y remoción, el equipo puede generar el plan de calidad. Finalmente, se examina el plan de calidad para verificar si los parámetros de calidad son razonables y si cumplen con los objetivos. De lo contrario los ingenieros ajustan las estimaciones y generan un nuevo plan de calidad.

2.4.3. Identificación de Problemas de Calidad

En TSP existen varias formas de identificar problemas de calidad. Por ejemplo, comparando la información de cada componente con el plan de calidad. Aquí es donde se puede ver fácilmente dónde algunas medidas como la densidad de defectos, las tasas de revisión, entre otras, presentan un desvío significativo respecto a las metas del equipo.

TSP introduce una serie de medidas de calidad:

- *Percent defect free—PDF* (PDF) : es el porcentaje de los componentes del sistema donde no se encontraron defectos en una fase de remoción.
- *Defect-removal profile*: se puede extraer para el sistema, para cada uno de sus subsistemas, cualquier componente, o incluso hasta el nivel de módulo.
- *Quality Profile*: mide los datos del proceso para cada componente. Son cinco dimensiones del perfil de calidad basadas en los datos de diseño, revisiones de diseño, revisiones de código, defectos de compilación y defectos en las pruebas unitarias.
- *Process Quality Index* (PQI): se calcula como el producto de las cinco dimensiones del perfil de calidad obteniendo como resultado un único valor de calidad.

2.4.4. Búsqueda y prevención de problemas

Las medidas de calidad TSP pueden indicar problemas de calidad en fases tempranas del proceso, incluso antes de la primera compilación, y proporcionan una medida fiable del componente antes de su integración o pruebas de sistema. Una vez que el equipo identifica un componente que probablemente tenga problemas de calidad, las medidas correctivas sugeridas por el TSP son las siguientes:

- Supervisar el componente durante las pruebas para verificar si se encuentran problemas y así determinar las acciones para corregirlo.
- Inspeccionar nuevamente el componente antes de las pruebas de integración o de las pruebas del sistema.
- El ingeniero debe trabajar en el componente para corregir posibles problemas.
- Desarrollar nuevamente el componente.

Capítulo 3

Process Dashboard

En este capítulo se presenta la herramienta utilizada para registrar los datos obtenidos durante la aplicación del proceso TSP. En la sección 3.1 se describe la herramienta de software denominada *Process Dashboard*, y en la sección 3.2 se detalla el *data warehouse* que genera la herramienta.

3.1. Descripción del Process Dashboard

Según el sitio oficial [7], *Process Dashboard* es una herramienta de propósito general diseñada para apoyar cualquier proceso definido, inicialmente creada para brindar apoyo a PSP y TSP. Es especialmente útil para apoyar métricas intensivas y procesos de alta madurez. Fue originalmente desarrollada en 1998 por la fuerza aérea de los Estados Unidos y continuada por un modelo de código abierto. Es gratuita para descargar bajo las condiciones de la licencia pública de GNU. Process Dashboard soporta:

1. Recolección de datos: Tiempo, defectos, tamaños, plan versus dato real.
2. Planificación: *Scripts*, planillas, *forms* y resúmenes integrados, PROBE, valor ganado.
3. Seguimiento: Soporte al valor ganado.
4. Análisis de Datos: Cuadros e informes de ayuda en el análisis de tendencias de los datos históricos.
5. Exportación de datos: Exportar datos a *excel* o exportar los datos a formato de texto para su uso con herramientas externas.

Las principales ventajas son:

1. Facilidad de uso: Facilidad en la recolección de datos para métricas (tiempo y defectos)
2. Flexibilidad y extensibilidad: Los nuevos procesos y tipos de datos se pueden agregar sin programación.
3. Independencia de la plataforma: ya que está implementada en *Java*, puede ser ejecutada en entornos varios como lo son Linux, Unix, Windows, Macintosh , etc.
4. Precio: Debido a que la herramienta es de código abierto se distribuye sin costo.

Al ejecutar la aplicación, se tiene acceso a una ventana principal como se muestra en la figura 3.1, que permite gestionar todo lo necesario para TSP.

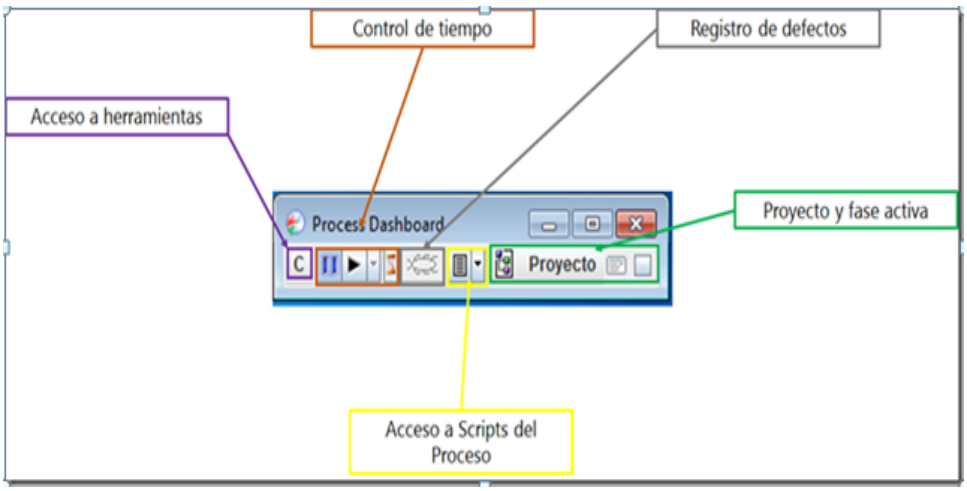

Figura 3.1: Ventana inical del Process Dashboard

Las opciones que presenta la herramienta son:

- Acceso a herramientas de configuración: acceso a la bitácora de tiempos, de defectos, plan de tareas, herramientas, etc.
- Control de Tiempos: con un solo clic se inicia/detiene el conteo del tiempo.
- Registro de Defectos: despliega la ventana de registro de defectos contando automáticamente el tiempo invertido en corregir el defecto.
- Proyectos y fase activa: permite visualizar el proyecto y fase en la cual se trabaja.
- Acceso a Scripts del Proceso: acceso a lo *scripts* y *forms* del proceso.

El registro de tiempo es una parte fundamental del proceso, pues establece una línea base para futuras estimaciones y por consiguiente para la planificación. Cuando se inicia una tarea o se retorna de una interrupción, basta con dar clic en el botón *Play* de la sección Control de Tiempos y automáticamente la herramienta empieza a registrar la fecha y hora de inicio, tiempos de interrupción (al seleccionar *Pause*) y fecha y hora de finalización, todos en minutos y segundos. A partir de esto se obtiene un delta en la fase que esté seleccionada, que en definitiva representa el tiempo real trabajado. Es decir, esto permite obtener el tiempo invertido en cada fase descontando las interrupciones realizadas.

Para registrar un defecto se hace clic en el botón de la sección de Registro de Defectos, donde se despliega una ventana como se muestra en la figura 3.2, que permite ingresar fecha, fase de inyección y remoción, tipo de defecto, descripción y defectos relacionados. A partir de esto la herramienta inmediatamente inicia el conteo del tiempo del defecto.

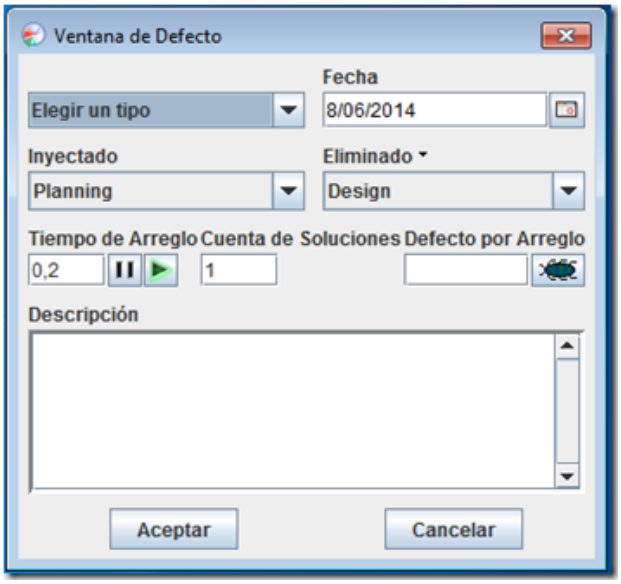

Figura 3.2: Ventana de registro de defectos

En el apéndice A se puede encontrar un manual básico del uso de la herramienta.

3.2. Team Process Data Warehouse

La herramienta genera una base de datos relacional que contiene los datos de los proyectos que aplican TSP. El diseño de este esquema sigue las mejores prácticas de los modelos dimensionales.

El *data warehouse* utiliza dos tipos de tablas, de dimensiones y de hechos (conocidas como *fact tables*).

Las tablas de dimensiones incluyen información contextual, como la fase del proceso y las claves de identificación de proyectos, equipos y personas. Las dimensiones son las siguientes: *ETL Audit Log, Organization, Team, Person, Project, EV Schedule, WBS Element,Task, Process, Phase, Plan Item, Process Enactment, Data Block, Size Metric, EV Metric, Defect Type, Measurement Type, Attribute, Text y Date.*

Los datos del proceso de desarrollo de software se registran en las tablas de hechos. Estas son: *Time Log, Defect Log, Task Status, Task Dates, EV Schedule Periods, EV Metric Values, Size, Plan Item Attribute, Plan Item Note y Process Enactment.*

En la figura 3.1 se puede observar la descripción en alto nivel de cómo se relacionan las tablas de dimensiones y de hechos del *data warehouse.*

DIMENSIONES HECHOS	Organization	Person	Team	Project	WBS Element	Task	Process	Metric
Time Log	X	X	X	X	X	X	X	
Defect Log	X	X	X	X	X	X	X	
Size Data	X	X	X	X	X	X		X
Process Metrics	X	X	X	X	X	X	X	X
Earned Value	X	X	X	X	X	X		

Cuadro 3.1: Relación entre dimensiones y hechos

Los equipos están conformados por un grupo de personas o en su defecto por una única persona, y pueden tener asignados varios proyectos. En la figura 3.3 se muestran las relaciones mencionadas. La jerarquía de un proyecto se basa en componentes y tareas. Las componentes están definidas en un WBS (*Work Breakdown Structure*). Las tareas es donde se efectúa el trabajo de una persona y se corresponden con una única fase del proyecto. Para la misma se registran datos de tiempo y de defectos.

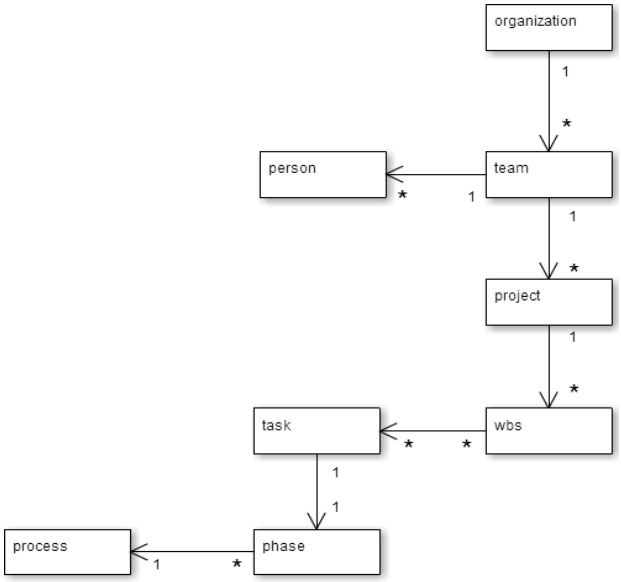

Figura 3.3: Relaciones importantes

Las principales tablas que se relacionan para dar la información necesaria sobre registros de tiempo, defectos, tamaños, proyectos, equipos y personas son las de las figura 3.4.

En la tabla *Time_log* se almacena el tiempo de inicio, de finalización, del delta y de la interrupción de una tarea.

En la tabla *Defect_log* se almacenan los datos de los defectos encontrados, la fase de inyectado y de removido, fecha de encontrado, tiempo de corrección, tipo de defectos, etc.

En la tabla *Size_fact* se almacena la información de tamaño total, agregada y modificada, base, eliminada y reutilizada.

En la tabla *measurement_type* permite indicar cuándo los tamaños son planificados o reales.

En la tabla *size_metric* se almacenan diferentes métricas de tamaños como por ejemplo LOC, document page, entre otras.

En la tabla *Plan_item* se almacena información a nivel de tareas. Permite relacionar las tablas de dimensiones *phase*, *project* y *WBS_element* con las tablas de hechos *time_log*, *defect_log* y *size_log* a través de la clave *plan_item_key*.

En la tabla *WBS_element* se almacenan las jerarquías de las componentes.

En la tabla *Data_block* se almacenan personas, equipos y organizaciones. Esta tabla se conecta con las tablas de hechos mediante el atributo *data_block_key*.

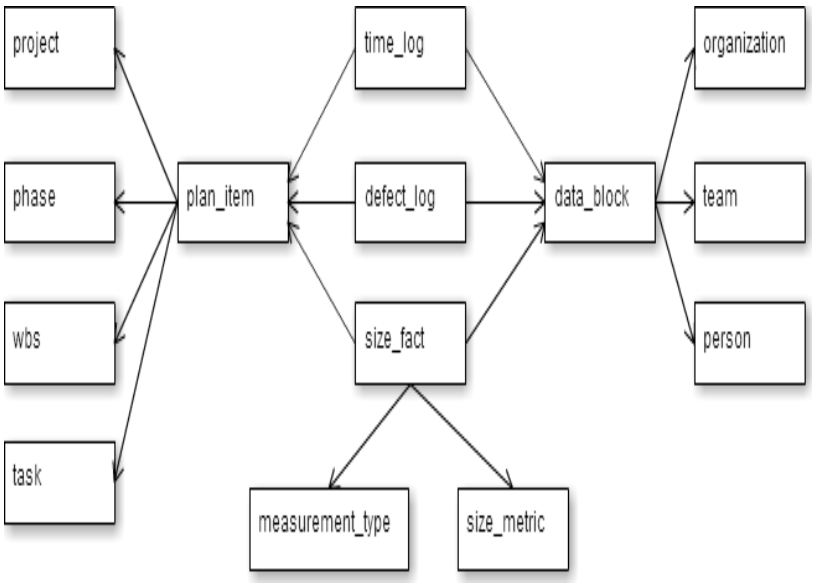

Figura 3.4: Tablas principales del data warehouse

Se deben tener consideraciones especiales al momento de trabajar con la información que contienen las tablas. En la tabla *measurement_type*, el atributo *measurement_type_name* debe ser igual al valor *Actual* para obtener una medida real y no la planificada, no alcanza con controlar que el *measurement_type_key* tenga un valor numérico específico. Para obtener medidas de tamaño de tipo líneas de código en la tabla *size_metric*, el atributo *size_measure_type* debe ser igual a 'LOC'. Y para calcular el tamaño se debe utilizar el atributo *size_added_and_modified* con valores mayores a cero.

Otro atributo a tener en cuenta es *current_flag_key*, donde su valor debe ser igual a 1. Este sirve para asegurarse que se está tomando la última actualización de la fila. Es importante saber que dicho valor se utiliza en todas las tablas donde se realizan las consultas. Estos atributos mencionados son utilizados para realizar las consultas SQL de los siguientes capítulos.

Capítulo 4

Descripción de los Datos

Se empleó una base de datos proporcionada por el *Software Engineering Institute* (SEI) de la Universidad de Carnegie Mellon. La misma cuenta con cuarenta y una tablas que almacenan la información que los equipos TSP registraron a través de la herramienta *Process Dashboard*. Dichas tablas corresponden a organizaciones, equipos, proyectos, personas, procesos, fases, componentes, tareas, registros de tiempo, tamaños y defectos.

En este capítulo se caracterizan los datos. En la sección 4.1 se presentan los datos y gráficas descriptivas, y en la sección 4.2 se mencionan los problemas de calidad de datos encontrados. La configuración del ambiente de trabajo necesario se encuentra en el apéndice B.

4.1. Gráficas Descriptivas

Como se mencionó anteriormente, la base de datos contiene datos de equipos, proyectos, personas, etc. La cantidad total de equipos es treinta y cinco, de proyectos es ciento nueve y de personas es trescientos treinta y cuatro.

La figura 4.1 muestra la cantidad de proyectos en los que participó cada equipo. El equipo 31 fue el que participó en más proyectos, con un total de catorce. Mientras que un 60 % del total de los equipos participó en tan solo un proyecto. Los equipos que no tienen proyectos asignados (estos son el 8 y 29) no son tenidos en cuenta para el análisis y tampoco se muestran en la figura.

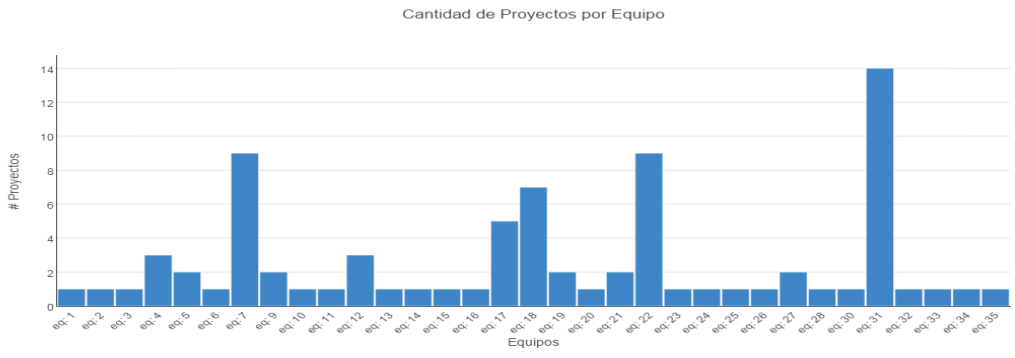

Figura 4.1: Cantidad de Proyectos por Equipos

Los datos estadísticos calculados se muestran en el cuadro 4.1:

Mediana	Rango	Media	Varianza	Desviación Estándar
1	13	2,45	8,854	2,975

Cuadro 4.1: Estadística descriptiva de la cantidad de proyectos por equipo

La figura 4.2 muestra la cantidad de personas que integran cada equipo. Estas son personas que a lo largo del tiempo pudieron haber participado en algún proyecto. Los equipos 22 y 31 registraron la mayor cantidad de personas con un total de ciento veintinueve y ciento cuarenta y dos respectivamente. Estos equipos no se muestran en la figura para una mejor visualización de los datos. El equipo 8 que no tiene personas asignadas, no es tenido en cuenta para el análisis y no se muestra en la figura.

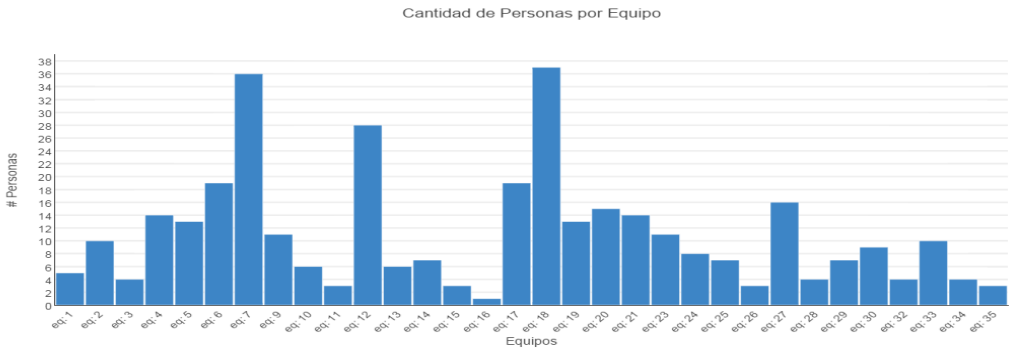

Figura 4.2: Cantidad de Personas por Equipos

En el cuadro 4.2 se muestran datos estadísticos descriptivos:

Mediana	Rango	Media	Varianza	Desviación Estándar
9,5	141	18,265	934,606	30,57133

Cuadro 4.2: Estadística descriptiva de la cantidad de personas por equipo

Un equipo puede tener distintos grupos de personas asignados a varios proyectos. A dicho grupos de personas se le denomina equipo de trabajo TSP. La figura 4.3 muestra los grupos de personas que integraron cada proyecto. Los proyectos 42 y 50 registraron la mayor cantidad de personas con un total de quince cada uno. Se puede observar que de los noventa y tres proyectos que registraron datos, doce registran menos de tres personas. Los proyectos que no registraron personas son un total de dieciséis, lo cuales no son tenidos en cuenta para el análisis y no se muestran en la figura.

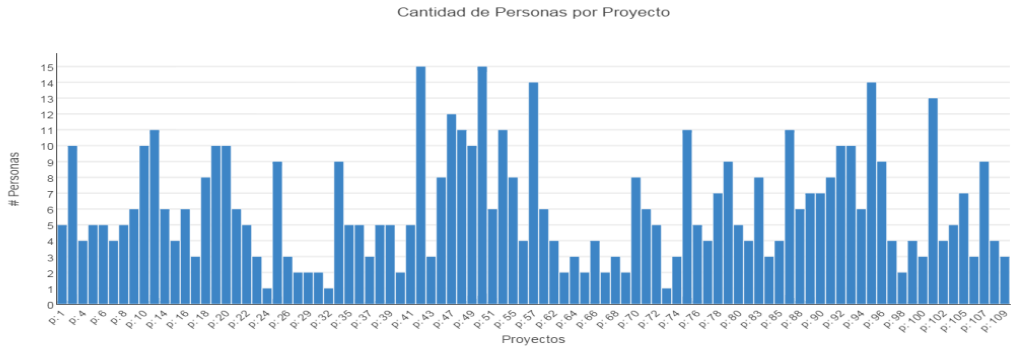

Figura 4.3: Cantidad de Personas por Proyectos

En el cuadro 4.3 se muestran datos estadísticos descriptivos:

Mediana	**Rango**	**Media**	**Varianza**	**Desviación Estándar**
5	14	6,043	13,146	3,625

Cuadro 4.3: Estadística descriptiva de la cantidad de personas por proyectos

En la figura 4.4 se puede ver un histograma con la distribución de proyectos agrupados por cantidad de personas. De un total de noventa y tres proyectos, el 87 % son proyectos conformados por tres a quince personas. El resto de los proyectos no estaría cumpliendo con la cantidad mínima de integrantes como se indica en la sección 2.1.

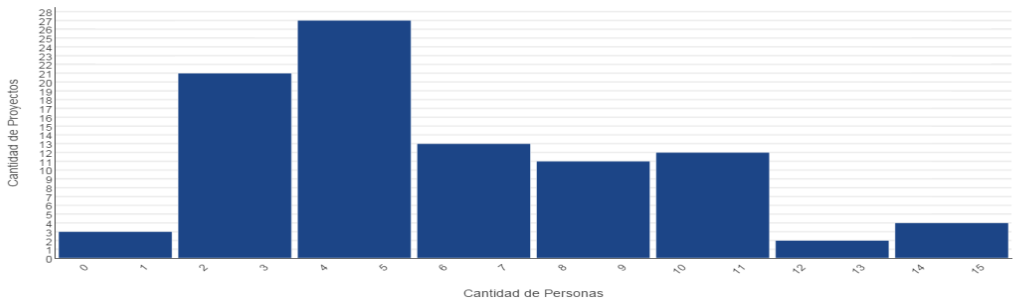

Figura 4.4: Histograma de Proyectos y Personas

La figura 4.5 muestra la cantidad de componentes que se realizan por proyecto. Los proyectos 50 y 102 registraron la mayor cantidad de componentes con un total de quinientos ochenta y nueve y seiscientos veintiséis respectivamente. Los proyectos que no registran componentes son un total de nueve, los cuales no son tenidos en cuenta para el análisis y no se muestran en la figura.

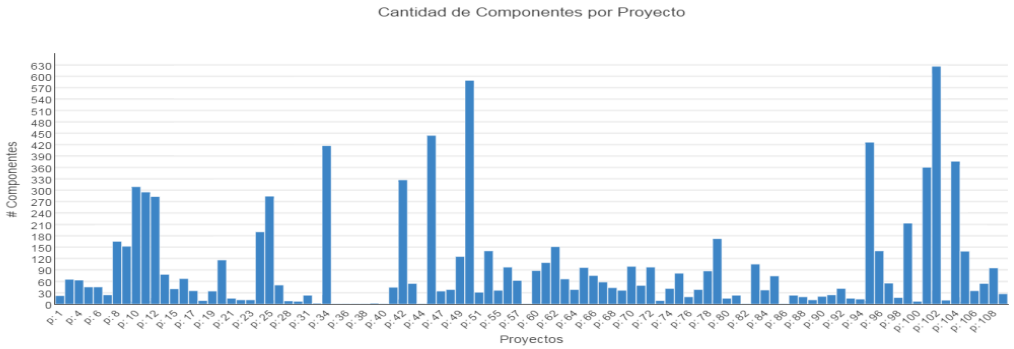

Figura 4.5: Cantidad de Componentes por Proyectos

En el cuadro 4.4 se muestran datos estadísticos descriptivos:

Mediana	**Rango**	**Media**	**Varianza**	**Desviación Estándar**
43,5	625	93,49	15908,55	126,12

Cuadro 4.4: Estadística descriptiva de la cantidad de componentes por proyectos

En la figura 4.6 se puede ver un histograma con la distribución de proyectos agrupados por cantidad de componentes. De un total de cien proyectos, el 54 % son proyectos que contiene una cantidad de componentes que oscila entre uno y cincuenta.

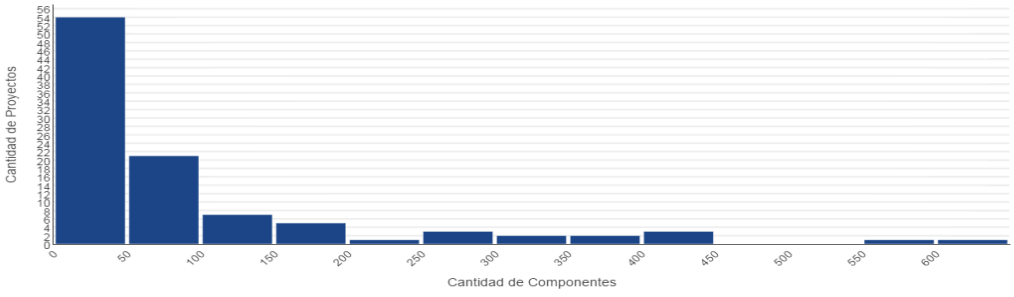

Figura 4.6: Histograma de Proyectos y Componentes

La figura 4.7 muestra los tamaños en KLOC (mil líneas de código) de cada proyecto. El proyecto de mayor tamaño es el proyecto 1 con un total de veintinueve KLOC. Un total de sesenta y ocho proyectos no registraron tamaño o no utilizaron la medida LOC, lo que equivale a más del 62 % del total de proyectos registrados en la base de datos.

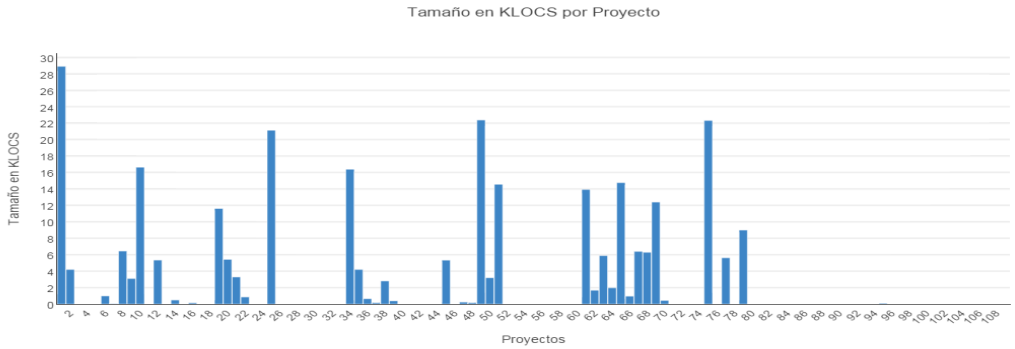

Figura 4.7: Tamaños por Proyectos

En el cuadro 4.5 se muestran datos estadísticos descriptivos:

Mediana	**Rango**	**Media**	**Varianza**	**Desviación Estándar**
4.77	28,86	6,851	55,005	7,416

Cuadro 4.5: Estadística descriptiva de la cantidad de tamaños de los proyectos

En la figura 4.8 se puede ver un histograma con la distribución de proyectos agrupados por tamaños en KLOC. De un total de cuarenta y un proyectos, el 37 % son proyectos que tienen un tamaño menor o igual a 2000 líneas de código.

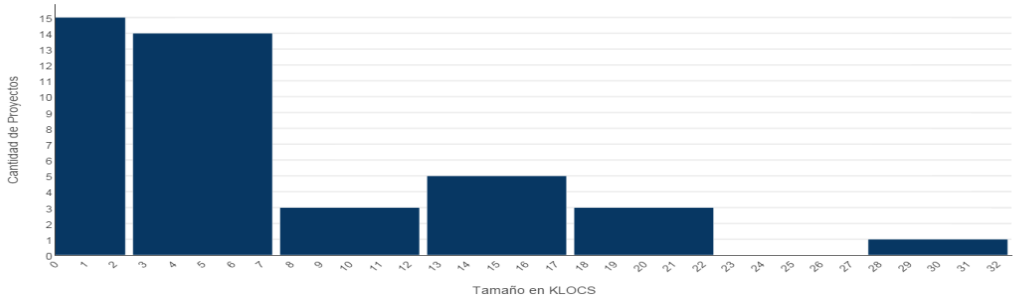

Figura 4.8: Histograma de Proyectos y Tamaños

La figura 4.9 muestra la cantidad de defectos registrados por proyectos. El proyecto 95 registra la mayor cantidad de defectos con un total de tres mil trescientos ochenta y nueve. Este proyecto no se muestra en la figura para una mejor visualización de los datos. Un total de veintiocho proyectos no registraron ningún defecto, los cuales no son tenidos en cuenta para el análisis y no se muestran en la figura.

Figura 4.9: Cantidad de Defectos por Proyectos

En el cuadro 4.6 se muestran datos estadísticos descriptivos:

Mediana	**Rango**	**Media**	**Varianza**	**Desviación Estándar**
56	3388	220,84	226545,58	475,969

Cuadro 4.6: Estadística descriptiva de la cantidad de defectos de los proyectos

En la figura 4.10 se puede ver un histograma con la distribución de proyectos agrupados por cantidad de defectos. De un total de ochenta y un proyectos, el 78 % son proyectos que presentan menos de doscientos defectos.

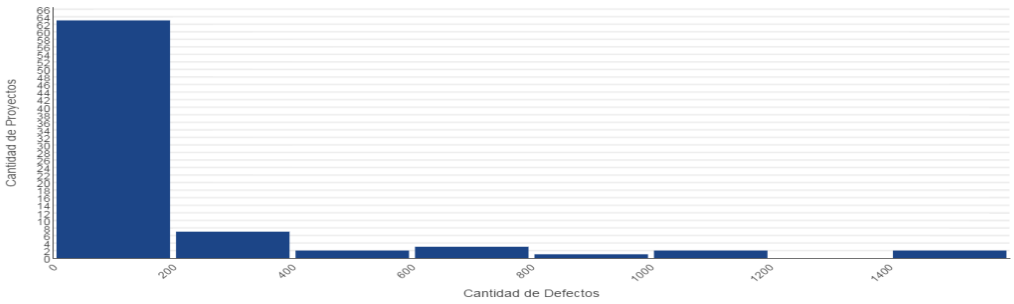

Figura 4.10: Histograma de Proyectos y Defectos

La figura 4.11 muestra el total de horas dedicadas a la realización de cada proyecto. El proyecto 95 tiene la mayor cantidad de horas registradas con un total de cinco mil quinientos dieciséis horas aproximadamente. De acuerdo al volumen de horas registradas por los distintos proyectos, se observa que el 20 % de los proyectos no alcanzan una hora. Los proyectos que no registran tiempos son un total de dieciséis, los cuales no son tenidos en cuenta para el análisis y no se muestran en la figura.

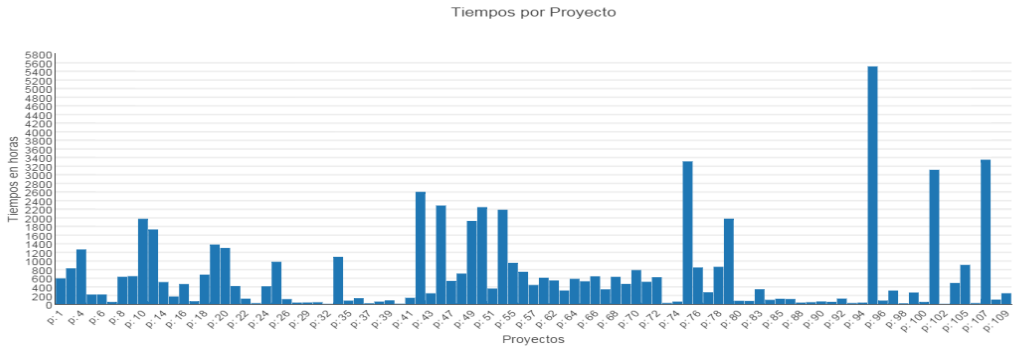

Figura 4.11: Proyectos y Tiempos

En el cuadro 4.7 se muestran datos estadísticos descriptivos:

Mediana	Rango	Media	Varianza	Desviación Estándar
344,55	5510,916	661,536	916546,99	957,36

Cuadro 4.7: Estadística descriptiva de los tiempos dedicados por los proyectos

En la figura 4.12 se puede ver un histograma con la distribución de proyectos agrupados por tiempos en horas. De un total de noventa y tres proyectos, el 58 % son proyectos que utilizaron menos de seiscientas horas de trabajo.

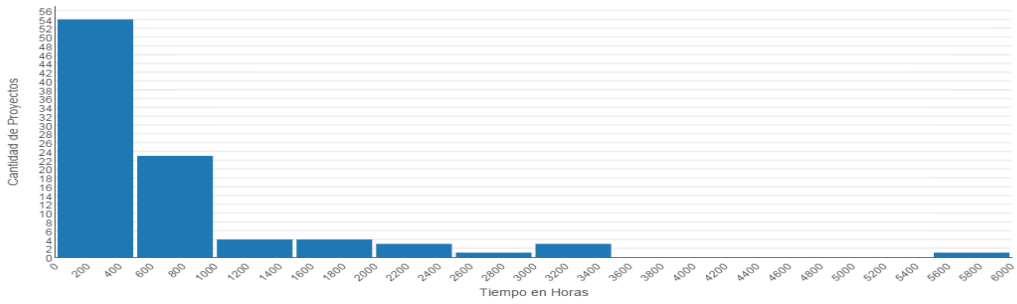

Figura 4.12: Histograma de Proyectos y Tiempos

Para graficar los resultados obtenidos se utilizan *scripts* escritos en el lenguaje R que se encuentran en el apéndice C.

4.2. Problemas de Calidad de Datos

En esta sección se mencionan algunos problemas de calidad que surgieron al trabajar con los datos proporcionados. Como se describió anteriormente, la información contenida en el *data*

warehouse utilizado es registrada por los usuarios a través de la herramienta *Process Dashboard* y la misma puede ser editada una vez ingresada, con lo cual, en algunos casos, pueden generar inconsistencias.

Los usuarios registran los tiempos de las tareas que les fueron asignadas (fecha y hora de inicio, interrupciones, fecha y hora de finalización). Y además registran los defectos encontrados (fecha y hora, fases de inyectado y removido, tipo de defecto) cuando realizan dichas tareas. Existen casos en que para cierta tarea, su fecha y hora de inicio y finalización no coincide con la fecha y hora registrada de un defecto.

Como se muestra en la figura 4.13, un usuario identificado con el *data_block_key* 33 (clave foránea de la tabla *data_block*), registra defectos con identificadores *defect_log_fact_key* 1647 y 1674 (identificador de la tabla *defect_log*) que se corresponden con los *plan_item_key* 3392 y 3407 (clave foránea de la tabla *plan_item*) respectivamente.

defect_log_fact_key	row_created_by_key	data_block_key	plan_item_key	defect_found_date
1647	51	33	3392	2011-09-01 10:10:37
1674	51	33	3407	2011-09-13 11:41:58
NULL	NULL	NULL	NULL	NULL

Figura 4.13: Instancia parcial de tabla defect_log_fact

Como se muestra en la figura 4.14, los datos registrados en la tabla de tiempos (tabla *time_log*) para el *data_block_key* y para los *plan_item_key* antes mencionados se aprecian distintos casos. En el primero, el registro del defecto (en la tabla *defect_log*) con *plan_item_key* 3392, cuenta con una fecha y hora comprendida en la franja horaria del registro de la tarea (en la tabla *time_log*). En cambio para el otro caso, el defecto fue registrado (en la tabla *defect_log*) con *plan_item_key* 3407 y con una fecha posterior a los tiempos registrados (en la tabla *time_log*) para esa tarea.

time_log_fact_key	row_created_by_key	data_block_key	plan_item_key	time_log_start_date	time_log_end_date
6693	47	29	3392	2011-08-29 10:24:29	2011-08-29 10:28:29
6694	47	29	3392	2011-08-29 10:50:39	2011-08-29 10:53:39
7140	51	33	3392	2011-08-29 10:50:14	2011-08-29 10:53:14
7161	51	33	3392	2011-09-01 10:10:36	2011-09-01 10:17:36
7145	51	33	3407	2011-08-29 14:57:23	2011-08-29 16:01:23
7146	51	33	3407	2011-08-29 16:14:44	2011-08-29 18:02:44
7147	51	33	3407	2011-08-29 17:46:54	2011-08-29 17:47:54
NULL	NULL	NULL	NULL	NULL	NULL

Figura 4.14: Instancia parcial de tabla time_log_fact

Otra situación similar se muestra en la figura 4.15, donde el usuario identificado con el *data_block_key* 32 (clave foránea de la tabla *data_block*), registra un defecto con identificador

defect_log_fact_key 1633 (de la tabla *defect_log*) que se corresponde con el *plan_item_key* 3619 (clave foránea de la tabla *plan_item*).

	defect_log_fact_key	row_created_by_key	data_block_key	plan_item_key	defect_found_date
	1633	50	32	3619	2011-11-23 15:26:32
*	NULL	NULL	NULL	NULL	NULL

Figura 4.15: Instancia parcial de tabla defect_log_fact_hist

Como se muestra en la figura 4.16, no existen registros de tiempo (en la tabla *time_log*) para el *data_block_key* y el *plan_item_key* mencionado.

	time_log_fact_key	row_created_by_key	data_block_key	plan_item_key	time_log_start_date	time_log_end_date
*	NULL	NULL	NULL	NULL	NULL	NULL

Figura 4.16: Instancia parcial de tabla time_log_fact

Otro problema identificado se da en los datos registrados en la tabla *size_fact*. Como se muestra en la figura 4.17, existen distintos valores de los atributos *size_total* y *size_added_and_modified*, y en particular para este último se pueden apreciar valores en cero y negativos. Para calcular el tamaño de una componente se utiliza el atributo *size_added_and_modified*. Para dicho cálculo no son tenidos en cuenta los registros con valores inconsistentes.

size_fact_key	size_added_and_modified	size_added	size_reused	size_deleted	size_modified	size_base	size_total
5446	6473	0	0	0	0	0	0
5448	0	0	0	0	0	16630	0
5476	33.8	0	0	0	0	0	0
6852	-1887	0	0	29	9	5662	3737
6981	1	0	0	0	1	1793	1793

Figura 4.17: Otra instancia parcial de tabla size_fact_hist

Como se muestra en la figura 4.18, el identificador *phase_key* de la fase no determina un orden de prelación entre las fases del proceso. Por lo cual es necesario utilizar la tabla *phase_order* que es donde se mapea el orden de las fases del proceso.

phase_key	process_key	phase_ordinal
94	6	9
95	6	10
96	6	32
97	6	11
98	6	12
99	6	13
100	6	17
101	6	7
450	6	1
451	6	2

Figura 4.18: Fases de un proceso

Todos los problemas de inconsistencias de la base de datos antes mencionados, afectan al conjunto de datos con los que se trabaja. Por este motivo fue necesario no tener en cuenta registros con las inconsistencias mencionadas en las consultas del análisis.

Capítulo 5

Análisis de los Datos

El proceso TSP incluye dos fases de revisiones personales, una luego de la fase de diseño y otra luego de la fase de código, en donde los ingenieros revisan sus productos de software con el objetivo de encontrar y remover defectos eficientemente.

El análisis consiste en estudiar por componente, la efectividad del diseño detallado y de las revisiones personales con el objetivo de obtener un producto de software de calidad. Se hará uso de la medida de calidad Process Quality Index reducido (PQI*), que recomienda ciertos tiempos a dedicar en las fases de diseño detallado y revisiones personales. Para evaluar la calidad del producto de software se plantearon dos enfoques, el primero que calcula la cantidad de defectos encontrados en las fases de *test* de integración (TI) y *test* de sistema (TS), y un segundo enfoque que hace uso de la medida de calidad *Process Yield*.

En la sección 5.1 se presenta el PQI* utilizado. En la sección 5.2 y 5.3 se estudian los enfoques aplicados.

5.1. Process Quality Index reducido (PQI*)

El PQI es una medida utilizada para identificar problemas de calidad [8]. Para este caso, se define un PQI* con el fin de analizar la calidad del producto desarrollado basándose únicamente en el diseño detallado y en las revisiones personales.

PQI* toma tres de las cinco dimensiones del perfil de calidad de PQI. Estas dimensiones toman valores entre 0 y 1, siendo 1 el mejor valor esperado. Cabe destacar que si el resultado en cualquiera de las dimensiones retorna un valor mayor a 1, entonces se deberá devolver el valor 1. El resultado de PQI* será el producto de todas las dimensiones calculadas. Las dimensiones que se tuvieron en cuenta son las siguientes:

- Tiempo Diseño Detallado: Establece el tiempo necesario a dedicar en la fase de diseño detallado para evitar errores de diseño. Se calcula como:

$$Tiempo\ Diseño\ Detallado\ = \frac{Tiempo\ Diseño}{Tiempo\ Código}$$

- Tiempo Revisión de Diseño: Establece el tiempo necesario a dedicar en la fase de revisión de diseño. De esta manera el ingeniero podrá remover un gran porcentaje de defectos. Se calcula como:

$$Tiempo\ Revisión\ Diseño\ = \frac{2\ *\ Tiempo\ Revisión\ Diseño}{Tiempo\ Diseño}$$

- Tiempo Revisión de Código: De forma similar al anterior, establece el tiempo necesario a dedicar en la fase de revisión de código. De esta manera el ingeniero podrá remover un gran porcentaje de defectos. Se calcula como:

$$Tiempo\ Revisión\ Código\ = \frac{2\ *\ Tiempo\ Revisión\ Código}{Tiempo\ Código}$$

.

Las siguientes dimensiones no fueron consideradas debido a que se hizo énfasis en las fases de diseño detallado y de revisiones personales, y no en fases posteriores como los son compilación y *testing* unitario.

- Defectos de compilación/KLOC.
- Defectos de *testing* unitario/KLOC.

5.1.1. Ejemplo de cálculo

A continuación se muestra un ejemplo del cálculo de PQI*.

Datos recolectados durante la aplicación de TSP:

- Tiempo Diseño Detallado (Hora): 7,6.
- Tiempo Revisión de Diseño (Hora): 1,25.
- Tiempo de Codificación (Hora): 8,9.
- Tiempo Revisión de Codificación (Hora): 3,9.

A partir de los datos se calculan los perfiles para cada dimensión:

- Tiempo Diseño Detallado = 7,6 / 8,9 = 0,85.
- Tiempo Revisión de Diseño = 2 * 1,25 / 7,6 = 0,33.
- Tiempo Revisión de Código = 2 * 3,9 / 8,9 = 0,88.

El PQI* del componente es igual a 0,85 * 0,33 * 0,88 = 0,25, sobre un valor óptimo de 1,0. En este caso particular el valor final obtenido indica, entre otras cosas, que el tiempo de revisión de la fase de diseño detallado es bajo de acuerdo a lo esperado. En la siguiente figura 5.1 se puede muestra el resultado.

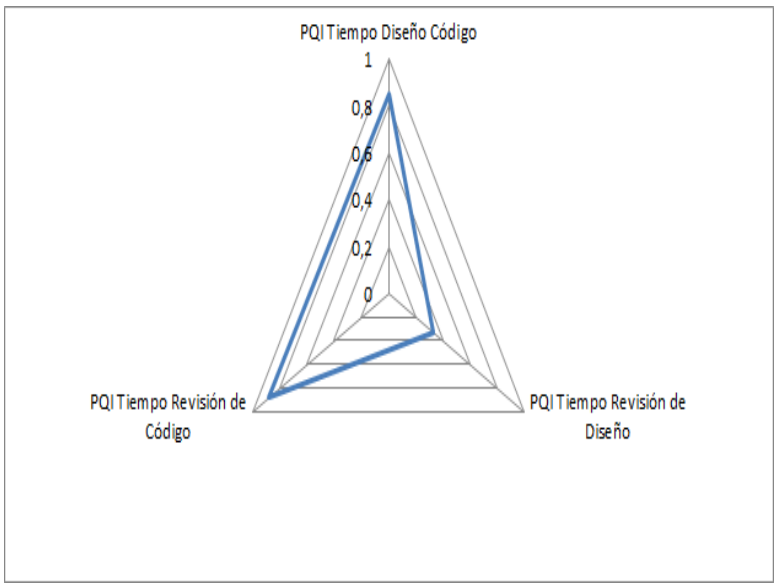

Figura 5.1: Resultado del Process Quality Index reducido

5.2. *Test* de integración (TI) y *test* de sistema (TS)

Para este enfoque, la calidad del producto de software se basó en la cantidad de defectos encontrados en las fases de *test* de integración (TI) y *test* de sistema (TS) para cada componente. Haciendo uso del PQI*, se intenta analizar si el tiempo dedicado en las fases del diseño detallado y de las revisiones personales podría llegar a reducir la cantidad de defectos en las fases de TI y TS.

5.2.1. Desarrollo

Para el enfoque se involucraron las componentes que cumplían con los siguientes filtros:

- Registraron tiempos para la realización de tareas en las fases de diseño detallado, revisión de diseño, código, revisión de código, TI y TS.
- Registraron defectos removidos en las fases de TI y/o TS.
- Registraron tamaños en líneas de código.

El *script* basado en R y las consultas SQL que generan las gráficas, se encuentran en el apéndice D.

5.2.2. Resultados

Aplicando los filtros de la sección anterior, en la figura 5.2 se muestra la variación de los defectos por KLOC (mil líneas de código) en las fases de TI y TS con respecto a PQI*.

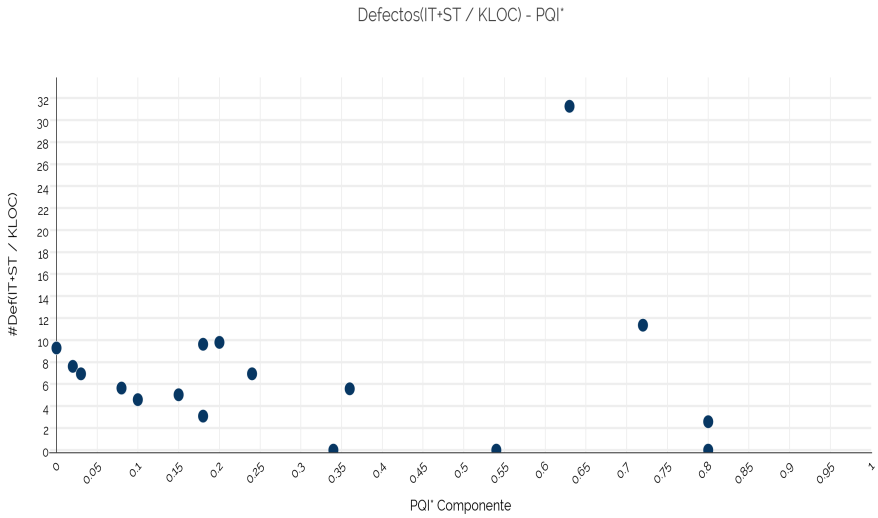

Figura 5.2: Defectos en TI y TS por KLOC respecto a PQI*

Como se puede observar, sobre un total de cuatrocientos setenta y ocho componentes que registran tiempos en las fases de diseño detallado, revisión de diseño, código y revisión de código, sólo son diecisiete los componentes obtenidos que además registran tiempo en TI y TS simultáneamente. Los mismos pertenecen a cuatro proyectos que a su vez son parte de un único equipo, siendo este el equipo 7.

5.2.3. Conclusiones

Debido a la insuficiente cantidad de componentes como consecuencia de los filtros especificados, no es posible estudiar si el tiempo dedicado en el diseño detallado y en las revisiones personales podría incidir en la calidad del producto de software.

La insuficiencia de componentes puede deberse a que:

- En la mayoría de los casos, no se registran tiempos en tareas de las fases de TI y TS por componente debido a que por lo general, se registran tiempos para un conjunto de componentes que se está integrando con el fin de obtener el producto de software final.
- No todas las componentes recolectan datos de defectos.
- Para algunas componentes no es posible calcular sus tamaños en KLOC debido a los problemas mencionados en la sección 4.2.

5.3. *Process Yield*

Process Yield es el porcentaje de defectos que fueron inyectados antes de la fase de compilación que fueron removidos antes de compilación. Su fórmula matemática es:

$$Process\ Yield(\%) = \frac{\#Defectos\ Removidos\ antes\ de\ Compilacion}{\#Defectos\ Inyectados\ antes\ de\ Compilacion} * 100$$

Para este enfoque, la calidad del producto de software se basó en el cálculo del *Process Yield* para cada componente. Haciendo uso del PQI*, se intenta analizar si el tiempo dedicado en las fases del diseño detallado y de las revisiones personales se podría llegar a obtener un valor del *Process Yield* cercano al 100 %.

5.3.1. Desarrollo

Para el enfoque se involucraron las componentes que cumplían con los siguientes filtros:

- Registraron tiempos para la realización de tareas en las fases de diseño detallado, revisión de diseño, código, revisión de código y compilación.
- Registraron al menos un defecto inyectado antes de compilación.

El *script* basado en R y las consultas SQL que generan las gráficas, se encuentran en el apéndice D.

5.3.2. Resultados

Aplicando los filtros de la sección anterior, en la figura 5.3 se muestra el gráfico obtenido:

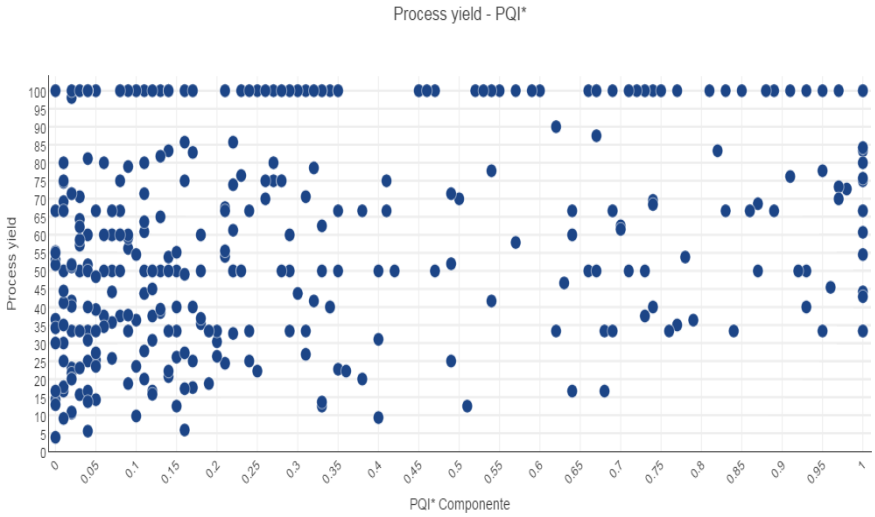

Figura 5.3: Process Yield PQI*

Se observa una alta variabilidad de los resultados para valores de PQI* menores a 0,45, por lo que no se puede deducir ninguna relación con el *Process Yield*. Sin embargo, cuando el PQI* es mayor o igual a 0,45, se puede ver que el *Process Yield* tiende a crecer cuando crece el PQI*, existe una franja de puntos que se van despegando del eje de las x's como se muestra en la figura 5.4.

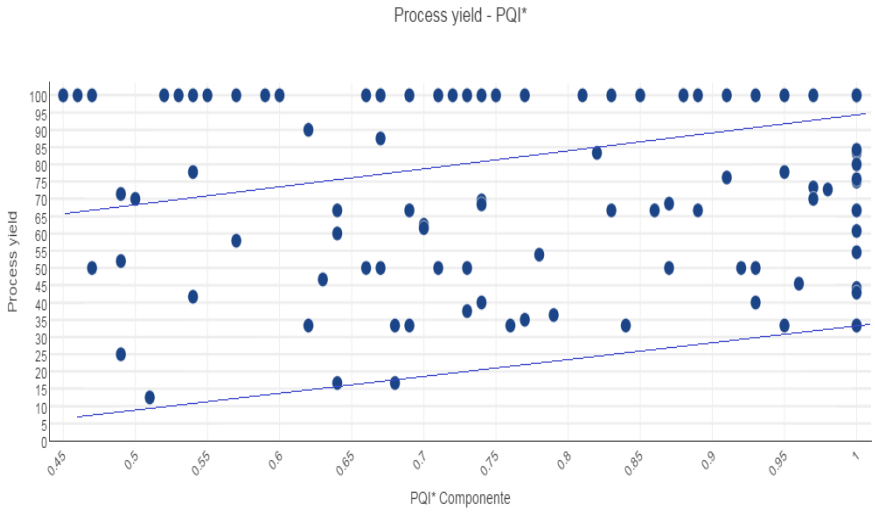

Figura 5.4: Process Yield PQI* mayor a 0,45

Para un PQI* mayor a 0,45 se caracterizan nuevamente los datos que fueron obtenidos, con el fin de verificar la variabilidad de los proyectos, componentes, personas, defectos y tiempos.

En la figura 5.5 se puede ver un histograma con la distribución de proyectos agrupados por componentes. La cantidad total de proyectos es veintinueve y de componentes es ciento nueve. El 73 % de los proyectos están conformados por cero a cuatro componentes.

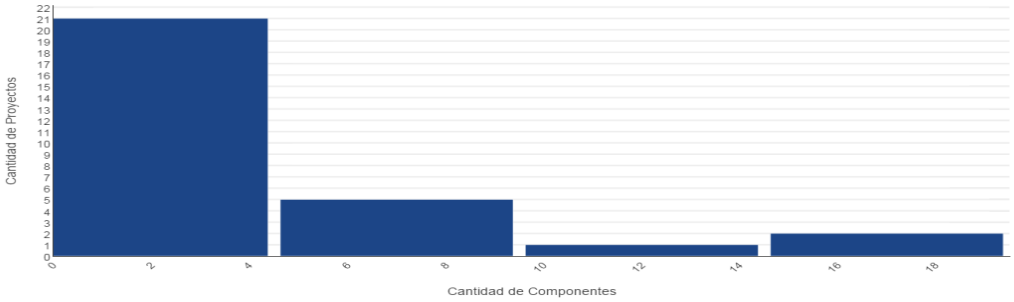

Figura 5.5: Histograma de Proyectos y Componentes para Process Yield

En la figura 5.6 se puede ver un histograma con la distribución de proyectos agrupados por cantidad de personas. De un total de veintinueve proyectos, el 72,4 % están conformados por tres a quince personas. El resto de los proyectos no estarían cumpliendo con la cantidad mínima de integrantes como se indica en la sección 2.1.

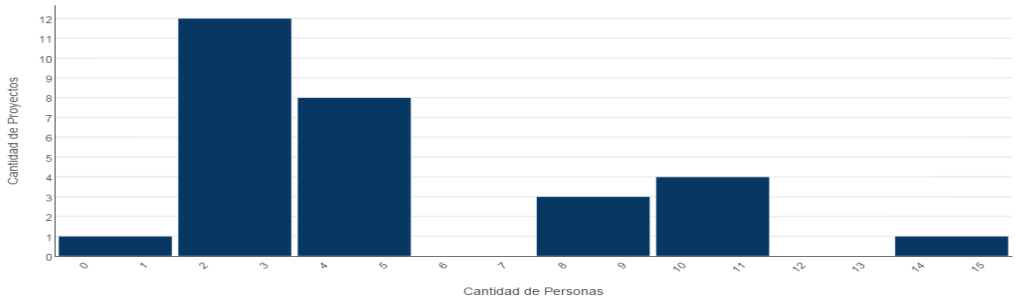

Figura 5.6: Histograma de Proyectos y Personas para Process Yield

En la figura 5.7 se puede ver un histograma con la distribución de proyectos agrupados por cantidad de defectos. De un total de veintinueve proyectos, el 83 % presentan menos de quinientos defectos registrados.

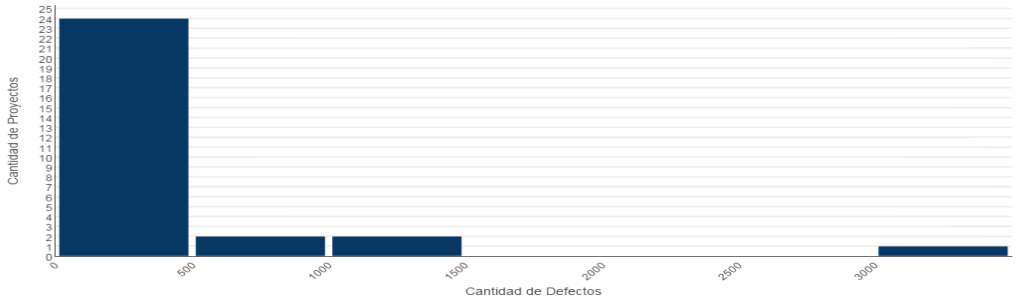

Figura 5.7: Histograma de Proyectos y Defectos para Process Yield

En la figura 5.8 se puede ver un histograma con la distribución de proyectos agrupados por tiempos en horas. De un total de veintinueve proyectos, el 83 % utilizaron menos de mil seiscientas horas de trabajo.

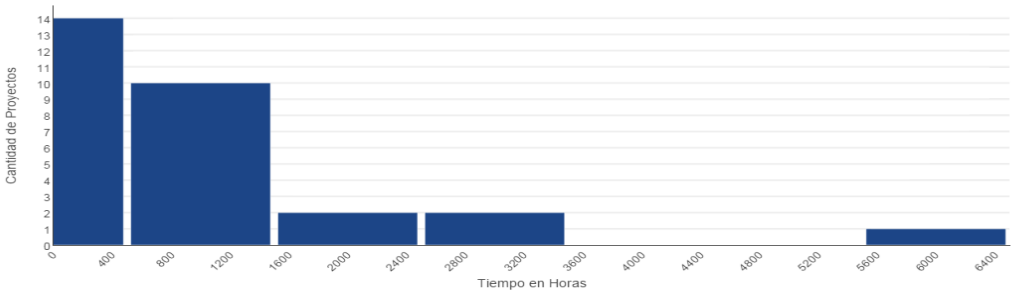

Figura 5.8: Histograma de Proyectos y Tiempos para Process Yield

5.3.3. Conclusiones

Se puede observar que existe una tendencia de crecimiento en el *Process Yield* para valores de PQI* mayores o iguales a 0,45 y no así para valores menores. Con esto, se podría llegar a concluir que el tiempo dedicado al diseño detallado y a las revisiones personales en diseño y código inciden en la obtención de un producto de calidad. Además, se caracterizaron nuevamente los datos, y se pudo observar variabilidad de los proyectos, componentes, personas, defectos y tiempos involucrados.

Capítulo 6

Conclusiones

En este capítulo se presentan las conclusiones en la sección 6.1 y el trabajo a futuro en la sección 6.2.

6.1. Conclusiones

En la actualidad, el software se ha convertido en una herramienta esencial de toda empresa u organización, e incluso de nuestros propios hogares. Juega un rol fundamental en nuestras vidas por lo que es de vital importancia el estudio de la calidad para su desarrollo. Para ello, es preciso la aplicación de un modelo de proceso que tome en cuenta la calidad, ya que la calidad del producto de software depende directamente de la calidad del proceso que lo generó.

Un modelo de proceso para el desarrollo de productos de software es el denominado *Team Software Process* (TSP). Este incluye dos fases de revisiones personales, una luego de la fase de diseño detallado y otra luego de la fase de código. En dichas fases, los ingenieros revisan sus productos de software con el objetivo de encontrar y remover defectos.

El objetivo general de este trabajo fue estudiar el modelo de proceso TSP y la efectividad del diseño detallado y de las revisiones personales para obtener un producto de software de calidad. Teniendo en cuenta esto se plantearon tres objetivos particulares.

En primer lugar se estudió el modelo de proceso TSP, haciendo énfasis en las fases de diseño detallado y de las revisiones personales. A partir de esto, tomando en cuenta la necesidad de recolectar datos durante la aplicación del proceso, se investigó el uso de la herramienta Process Dashboard utilizada para este fin. Se observó su funcionamiento y cómo se registran los datos en la base de datos que la herramienta genera.

En segundo lugar se realizó el estudio de los datos contenidos en la base de datos que fue proporcionada por el *Software Engineering Institute* (SEI). La misma incluía información recolectada a través de la aplicación del proceso TSP por equipos de desarrollo de software. El estudio de los datos consistió en la realización de consultas básicas con el fin de determinar el universo de los mismos, lo que permitió su correcta caracterización así como la elaboración de una sección con problemas de calidad de datos. Muchos de estos problemas fueron la faltante de registros o inconsistencias en los datos, lo que presentó un problema importante y determinante en la realización de esta tesis.

Finalmente se analizó la efectividad del diseño detallado y de las revisiones personales con el objetivo de obtener un producto de software de calidad. Se utilizó la medida de calidad Process Quality Index reducido (PQI*), que recomienda ciertos tiempos a dedicar en las fases de diseño detallado y revisiones personales. Para evaluar la calidad del producto de software se plantearon

dos enfoques.

El primero calcula la cantidad de defectos encontrados en las fases de *test* de integración (TI) y *test* de sistema (TS). Como resultado, luego de aplicados los filtros especificados en el desarrollo, debió ser descartado por entender que la cantidad de componentes involucradas eran insuficientes para su análisis.

El segundo hace uso de la medida de calidad *Process Yield*, que es una medida de calidad calculada como el porcentaje de defectos que fueron inyectados y removidos antes de la fase de compilación. De los resultados obtenidos, se observó una alta variabilidad en los datos, y se pudo inferir que solo en aquellas componentes donde el PQI* era superior a 0,45, el *Process Yield* creció proporcionalmente al PQI*. De esta manera, se pudo inferir que la calidad en el diseño detallado y las revisiones personales para un PQI* superior a 0,45, podría ser un indicador de la calidad del producto medido a través de un *Process Yield* que tiende a cero defectos. Se caracterizaron nuevamente los datos, y se pudo observar variabilidad de los proyectos, componentes, personas, defectos y tiempos involucrados.

Como indica PQI*, el tiempo dedicado en la fase diseño detallado debería ser igual o superior al tiempo dedicado en la fase código, así como los tiempos en las fases de revisión de diseño y revisión código deberían ser de al menos la mitad del tiempo que se le dedicó a la fase de diseño detallado y código respectivamente. Esto implicaría que dichas fases son fundamentales en la detección de defectos en fases tempranas del proceso para obtener un producto de software de calidad.

6.2. Trabajo a futuro

Se debería profundizar el análisis de los problemas de calidad de los datos propocionados por el SEI. Se tuvieron que realizar filtros en las consultas de los datos debido a los niveles de calidad inadecuados, por lo que los resultados quizás puedan no reflejar la realidad de todo el universo. Esto implicaría realizar una limpieza de los datos.

Se podría aplicar un estudio similar al realizado utilizando otras medidas de calidad. De esta manera se podrían comparar los resultados y establecer así puntos de relación entre los mismos.

Sería bueno realizar un estudio sobre el registro de datos en la herramienta Process Dashboard durante la aplicación del proceso, ya que podría tener una incidencia directa sobre los problemas de calidad.

Debido a los resultados obtenidos, se debería seguir trabajando sobre otros conjuntos de datos que puedan proporcionar más información sobre la aplicación del proceso TSP.

Apéndice A

Manual de Uso de la Herramienta Process Dashboard

La herramienta se descarga de la página oficial [1], una vez que se encuentra instalada se accede a través del ícono *Process Dashboard* de la figura A.1.

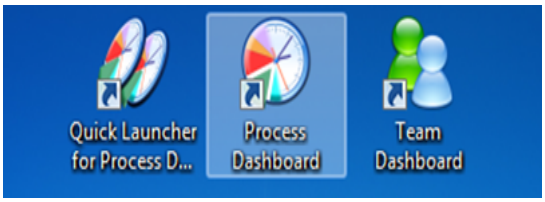

Figura A.1: Icono de Process Dashboard

La ventana principal de la herramienta se muestra en la figura A.2, esta permite gestionar todo lo necesario para TSP.

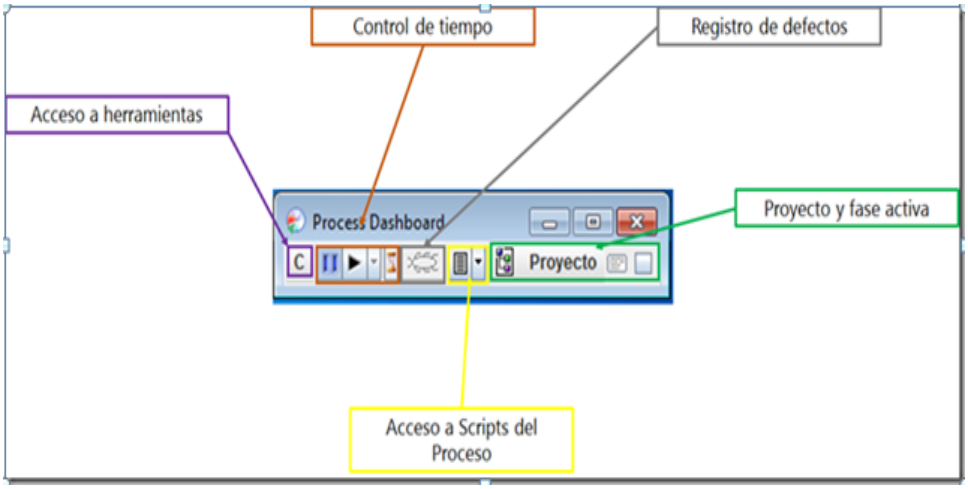

Figura A.2: Ventana Principal de Process Dashboard

- Acceso a herramientas de configuración: acceso a las siguientes opciones como se muestra en la figura A.3

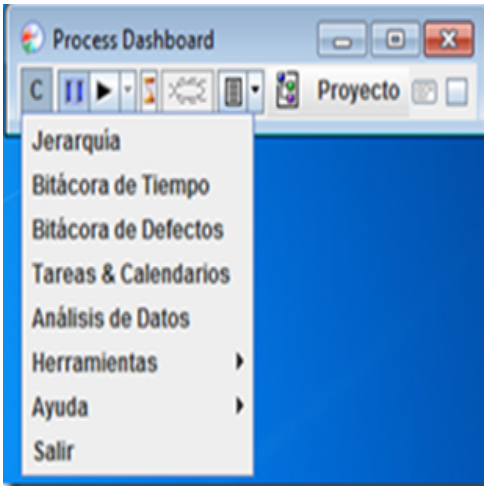

Figura A.3: Ventana Configuración

  - Jerarquía: permite la creación de nuevos proyectos a partir de Plantillas predefinidas o proyectos.
  - Bitácora de Tiempo: acceso a los registros de tiempos
  - Bitácora del Defectos: acceso al registro de los defectos.
  - Tareas y Calendarios: permite acceder al Plan de Tareas.
  - Análisis de Datos: gráficos y resúmenes de información.
  - Herramientas: acceso a diferentes opciones de respaldo de información, configuraciones, etc.

- Control de Tiempos: con un solo clic se inicia y se detiene el contador de tiempos registrándolo en el proyecto y fase actual. La aplicación automáticamente cuenta la Fecha y Hora de inicio, tiempos de interrupción y Fecha y Hora de Finalización, todos en minutos y segundos, obteniendo finalmente el Delta (tiempo real trabajado).

- Registro de Defectos: esta opción da inicio al registro de los defectos como se muestra en la figura A.4, permitiendo ingresar Fecha, Fase de Inyección y Remoción, Tipo de Defecto, Descripción y Defectos enlazados.

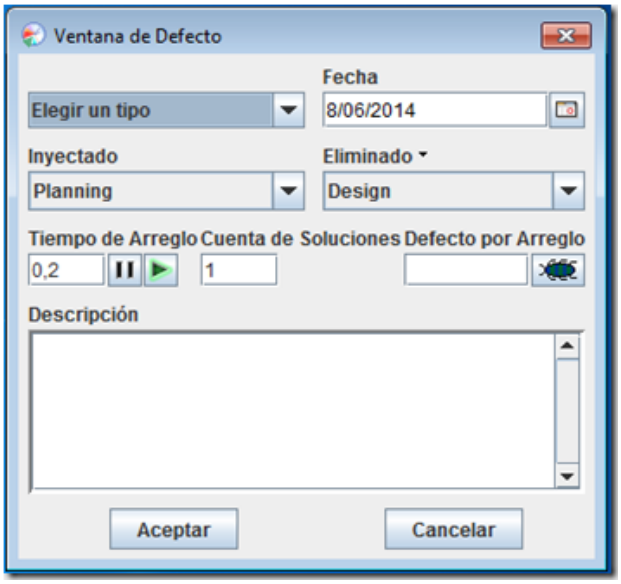

Figura A.4: Ventana registro defecto

- Proyectos y fase activa: con esta opción se puede navegar por el proyecto y sus diferentes fases como se muestra en la figura A.5.

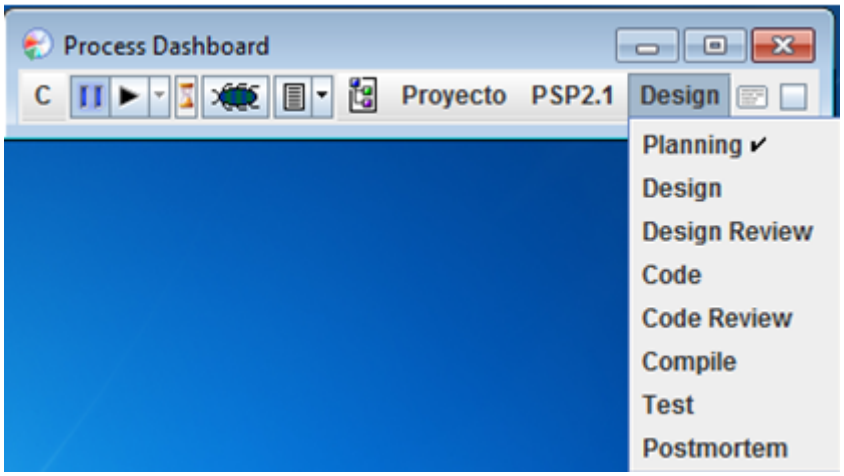

Figura A.5: Ventana proyecto y fase activa

- Acceso a *scripts* del proceso: brinda acceso a los *scripts* del proceso y a diferentes plantillas para el registro, visualización y análisis de información como se muestra en la figura A.6.

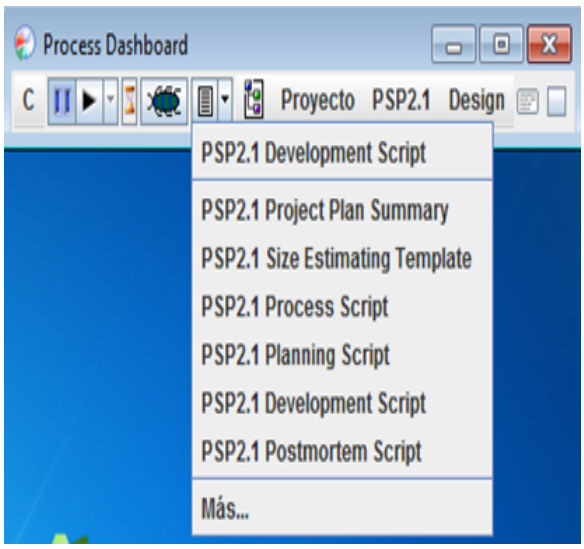

Figura A.6: Ventana Script del Proceso

A.0.1. Creación de un proyecto

1. Seleccionar el menú de configuración y luego el menú Jerarquía y se obtiene la ventana que se muestra en la figura A.7.

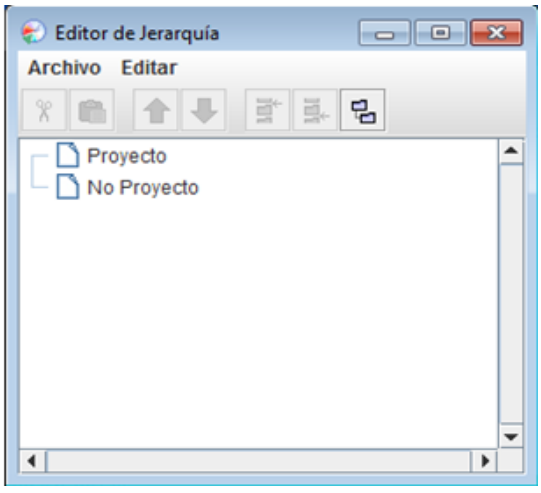

Figura A.7: Ventana Crear proyecto

2. Seleccionar Proyecto en el panel central y luego Editar >Agregar Plantilla y esto permite ingresar un nombre descriptivo para este Proyecto como se muestra en la figura A.8.

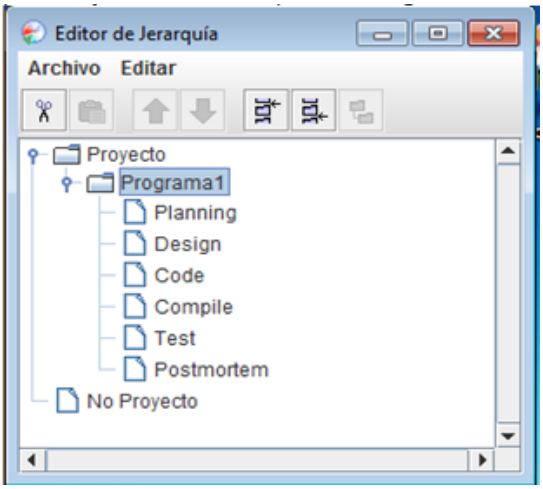

Figura A.8: Ventana jerarquia

A.0.2. Registro de Tiempos

El registro de tiempos es una parte fundamental del proceso, pues establece la línea base para las futuras estimaciones y por consiguiente para la planificación.

1. Al iniciar el proyecto o regresar de una interrupción solo basta dar clic en el botón *Play* de la figura A.9 de la sección Control de Tiempos y automáticamente empieza a registrar el tiempo en la fase que esté seleccionada.

Figura A.9: Ventana principal con boton play presionado

2. Cuando transcurran unos segundos en el menú C >Bitácora de Tiempos se tiene acceso a los registros como se muestra en la figura A.10.

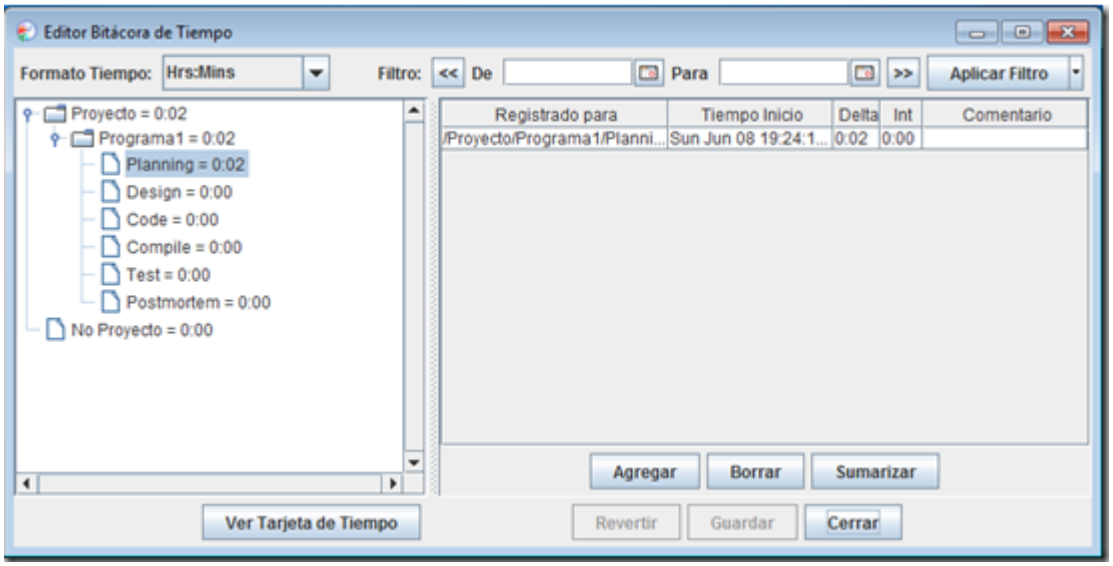

Figura A.10: Ventana de bitacora de tiempo

3. La aplicación empezará a registrar el tiempo invertido en cada fase, descontando las interrupciones (solo basta dar *Pause* y *Process Dashboard* resta el tiempo y calcula el delta)

4. Al completar la fase se marca el *checkbox* que se muestra en la figura A.11 y *Process Dashboard* salta inmediatamente pasa a la siguiente fase.

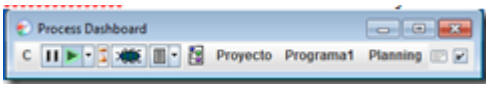

Figura A.11: Ventana Fase Terminada

A.0.3. Registro de Defectos

Cuando se trabaja en una fase determinada y se requiere cambiar algún elemento (Código, Documentos, Diseño, etc.) es necesario registrar un defecto.

1. Sin detener el tiempo, dar clic en el ícono de la figura A.12.

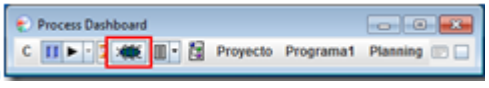

Figura A.12: Boton para registrar un defecto

2. Esto depliega la ventana de la figura A.13.

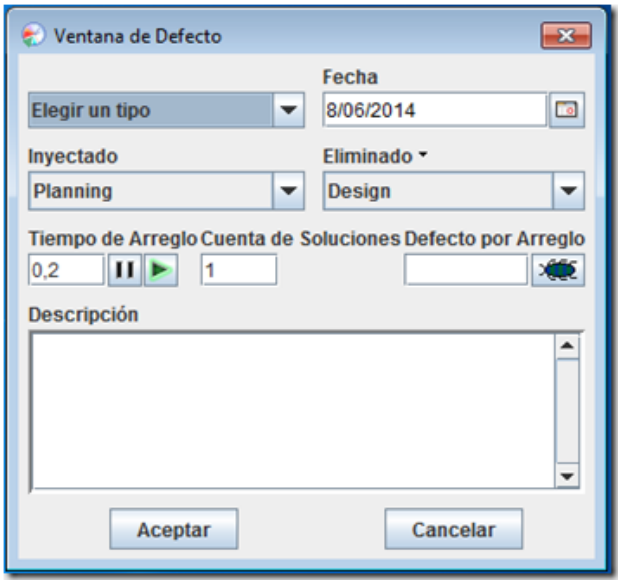

Figura A.13: Ventana registro defecto

3. La herramienta inmediatamente inicia el conteo del tiempo del defecto, y permite seleccionar:

   - Tipo del Defecto: PSP viene con algunos tipos preestablecidos los cuales nos sirven como base.
   - Fecha: momento en el cual se está registrando el defecto.
   - Fase de Inyección: fase en la cual se cometió el error, la omisión o se debió haber tenido en cuenta la incorporación del defecto.
   - Fase de Remoción: fase en la cual se eliminó/Corrigió/implementó lo necesario para solucionar el defecto.
   - Descripción: Resumen del defecto.
   - Defectos relacionados: Nuevos defectos generados a partir del Defecto que se está corrigiendo.

4. Al dar click en aceptar el defecto queda registrado.

Apéndice B

Manual de Configuración del Ambiente de Trabajo

Configuración del Ambiente para Windows:

1. Instalar MySQL Community y MySQL Workbench descargado de [2].

2. Importar los datos con MySQL Workbench: iniciar MySQL Workbench y abrir una instancia local en el cuadro gris que se muestra en la figura B.1.

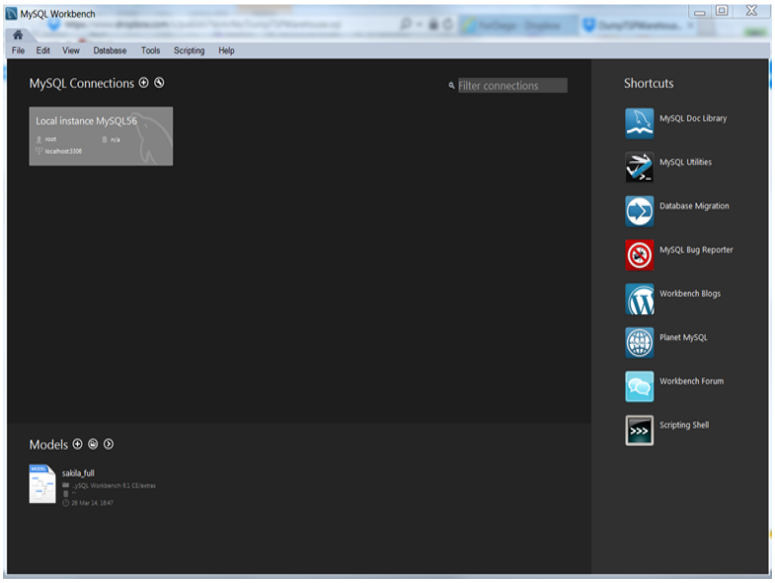

Figura B.1: Ventana inicial de MySQL Workbench

Seleccionar la opción Data Import /Restore. Luego importar de un archivo e indicar la dirección del archivo como se muestra en la figura B.2.

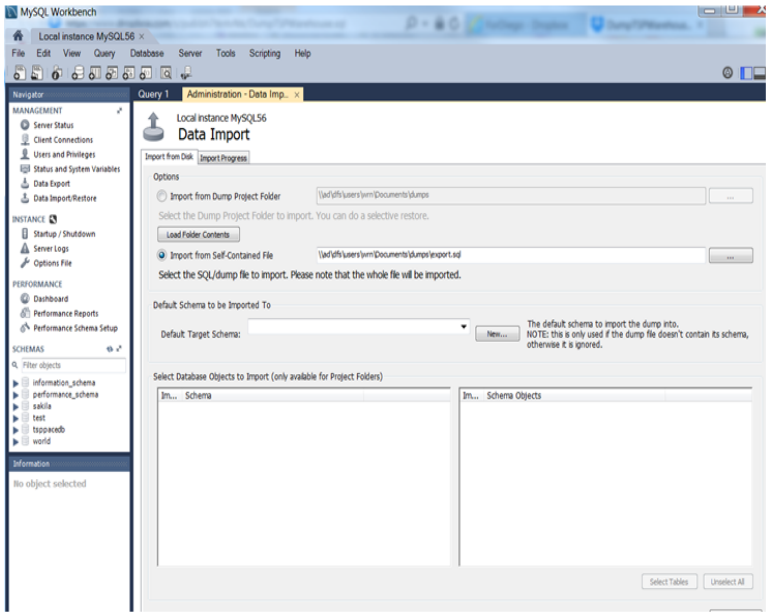

Figura B.2: Ventana importacion de MySQL Workbench

3. Instalar R descargado de [4]. En la instalación es recomendable instalar de acuerdo a su sistema operativo 32 o 64 bits ver figuara B.3.

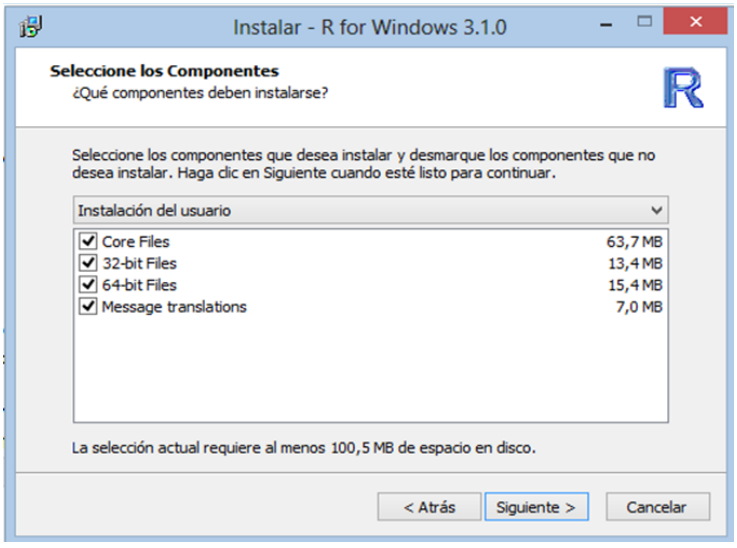

Figura B.3: Instalación de R para Windows 3.1.0

4. Instalar Rtools descargado de [6]. En el siguiente paso es importante marcar todos los checkboxes como se muestra en la figura B.4.

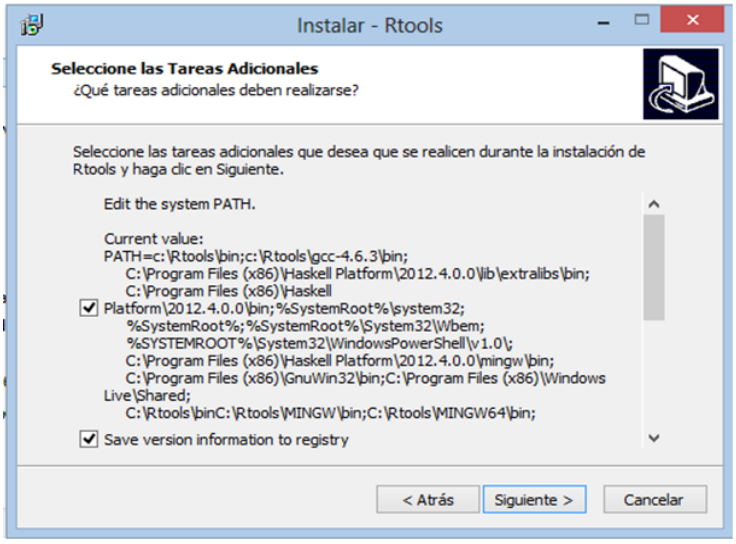

Figura B.4: Ventana de checkboxes de Rtools

Editar el path para que tenga las rutas necesarias para encontrar el compilador como se muestra en la figura B.5 en amarillo.

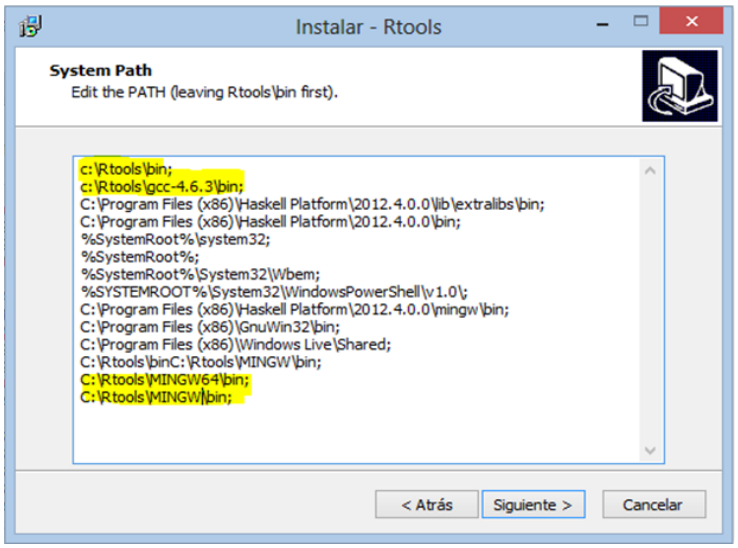

Figura B.5: Ventana de rutas de Rtools

5. Instalar RStudio descargando de la [5].

6. Verificar o setear variable de entorno como se muestra en la figura B.6:

   - R_Libs, e.g /../R/winlibrary/3.1
   - MYSAL_HOME, e.g C:/Program Files/MYSQL/MYSQL Server 5.6

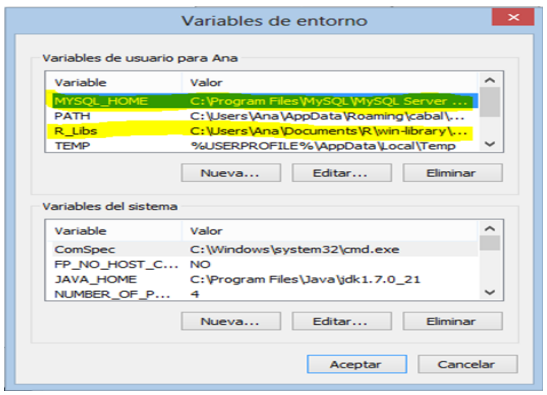

Figura B.6: Ventana de Variables Entorno

7. RMySQL necesita libmysql.dll para compilar exitosamente. Copiar libmysql.dll desde la carpeta lib a la carpeta bin dentro del directorio C://Program Files/MySQL/MySQL Server 5.5/

8. Abrir R y probar la conexión con la base de datos. Paquetes->Cargar Paquete y seleccionar DBI. De la misma forma cargar el paquete RMySQL y luego ejecutar el siguiente test de conexión:

```
# set up the driver
m<-dbDriver("MySQL");
#Connect and authenticate to a MySQL database
con<-dbConnect(m, user=root, password=root, host='localhost',dbname='tsppacedb'
    );
# count entries in a table
dbGetQuery(con, "select count(*) from etl_batch");
#read the table ev_metric into a local variable
tab_ev_metric<-dbGetQuery(con, "select * from ev_metric") ;
# display table column names
colnames(tab_ev_metric);
```

9. También se puede realizar el paso anterior desde RStudio. Se debe instalar el paquete DBI y luego ejecutar el siguiente comando: install.packages('RMySQL',type='source') Y luego ejecutar el test de conexión.

10. Es necesario instalar las siguiente librerias en RStudio.

    - install.packages("viridis")
    - install.packages("devtools")
    - install.packages(çurl")
    - devtools::install_github(ropensci/plotly")

11. Para hacer uso de gráficas plotly se debe crear un usuario [3] y con la información setear las credenciales.

    - Sys.setenv("plotly_username-zour_plotly_username")
    - Sys.setenv("plotly_api_key-zour_api_key")

Apéndice C

Consultas Básicas

C.1. Equipos y proyectos

La siguiente consulta obtiene la cantidad de proyectos registrados por equipo.

```
(SELECT COUNT(*)
FROM (SELECT DISTINCT pi.project_key
FROM defect_log_fact_hist d
INNER JOIN plan_item pi ON d.plan_item_key = pi.plan_item_key
INNER JOIN data_block db ON d.data_block_key = db.data_block_key
WHERE db.team_key = team ) as b)
UNION
(SELECT COUNT(*)
FROM (SELECT DISTINCT pi.project_key
FROM time_log_fact_hist d
INNER JOIN plan_item pi ON d.plan_item_key = pi.plan_item_key
INNER JOIN data_block db ON d.data_block_key = db.data_block_key
WHERE db.team_key = team ) as b)
UNION
(SELECT COUNT(*)
FROM (SELECT DISTINCT pi.project_key
FROM task_date_fact_hist d
INNER JOIN plan_item pi ON d.plan_item_key = pi.plan_item_key
INNER JOIN data_block db ON d.data_block_key = db.data_block_key
WHERE db.team_key = team ) as b)
UNION
(SELECT COUNT(*)
FROM (SELECT DISTINCT pi.project_key
FROM size_fact_hist d
INNER JOIN plan_item pi ON d.plan_item_key = pi.plan_item_key
INNER JOIN data_block db ON d.data_block_key = db.data_block_key
WHERE db.team_key = team ) as b)
```

Para esta consulta se itera por cada equipo obtenido - representado con “team” - de la tabla data_block y luego se realiza la unión de las tablas time_log, defect_log, size_fact y task_date a través de un JOIN con la tabla plan_item para obtener los proyectos asignados.

C.2. Equipos y personas

La consulta obtiene la cantidad de personas por equipo.

```
SELECT COUNT(*)
```

```
FROM data_block WHERE team_key = team
```

Para esta consulta se itera por cada equipo obtenido - representado con “team” - de la tabla data_block y se cuenta la cantidad de personas en cada uno de ellos.

C.3. Proyectos y personas

La consulta obtiene la cantidad de personas por proyectos.

```
(SELECT COUNT(*)
FROM (SELECT DISTINCT db.person_key
FROM defect_log_fact_hist d
INNER JOIN plan_item pi ON d.plan_item_key = pi.plan_item_key
INNER JOIN data_block db ON d.data_block_key = db.data_block_key
WHERE pi.project_key =  proj ) as b)
UNION
(SELECT COUNT(*)
FROM (SELECT DISTINCT db.person_key
FROM time_log_fact_hist d
INNER JOIN plan_item pi ON d.plan_item_key = pi.plan_item_key
INNER JOIN data_block db ON d.data_block_key = db.data_block_key
WHERE pi.project_key =  proj ) as b)
UNION
(SELECT COUNT(*)
FROM (SELECT DISTINCT db.person_key
FROM task_date_fact_hist d
INNER JOIN plan_item pi ON d.plan_item_key = pi.plan_item_key
INNER JOIN data_block db ON d.data_block_key = db.data_block_key
WHERE pi.project_key =  proj) as b)
UNION
(SELECT COUNT(*)
FROM (SELECT DISTINCT db.person_key
FROM size_fact_hist d
INNER JOIN plan_item pi ON d.plan_item_key = pi.plan_item_key
INNER JOIN data_block db ON d.data_block_key = db.data_block_key
WHERE pi.project_key = proj ) as b)
```

Para esta consulta se itera en cada uno de los proyectos obtenidos - representado con “proj” - de la tabla plan_item y luego se realiza la union de las tablas time_log, defect_log , task_date y size_fact y estas con un JOIN con data_block para obtener las personas.

C.4. Proyectos y componentes

La consulta obtiene la cantidad de componentes por proyecto.

```
SELECT COUNT(*)
FROM (SELECT DISTINCT(pi.wbs_element_key)
                      FROM plan_item pi
      WHERE pi.task_key is not null and pi.project_key  = proj) as b
```

Para esta consulta se itera sobre la cantidad de proyectos - representado con “proj” -, y para cada uno de ellos se obtiene las componentes.

C.5. Proyectos y tamaños

La consulta obtiene el tamaño por proyecto.

```
SELECT ifnull(sum(s.size_added_and_modified), 0)
FROM size_fact_hist s
INNER JOIN plan_item pi ON pi.plan_item_key = s.plan_item_key
INNER JOIN measurement_type m ON m.measurement_type_key = s.measurement_type_key
INNER JOIN size_metric e ON e.size_metric_key = s.size_metric_key
INNER JOIN phase ph ON ph.phase_key = pi.phase_key
WHERE s.row_current_flag = 1 and  m.measurement_type_name = 'Actual' and e.size_
    metric_name = 'Lines of Code' and (ph.phase_name like 'Code' or ph.phase_name
     like 'TSP - Code') and pi.project_key  = proj)
```

Para esta consulta se itera en cada uno de los proyectos - representado con "proj" - obtenidos de plan_item y se seleccionan los tamaños registrados en la tabla size_fact_hist que cumplen con: medida igual a líneas de código, fase código y métricas reales.

C.6. Proyectos y defectos

La consulta obtiene la cantidad de defectos por proyecto.

```
SELECT ifnull(COUNT(*), 0)
FROM defect_log_fact_hist d
INNER JOIN plan_item pi ON pi.plan_item_key = d.plan_item_key
WHERE d.row_current_flag = 1 and pi.task_key IS NOT NULL and pi.project_key  =
    proj;
```

Para esta consulta se itera en cada uno de los proyectos - representado con "proj" - obtenidos de plan_item y se seleccionan los defectos registrados en la tabla defect_log_fact_hist.

C.7. Proyectos y tiempos

La consulta obtiene los tiempos por proyecto.

```
SELECT ifnull(sum(tl.time_log_delta_minutes), 0)
FROM time_log_fact_hist tl
INNER JOIN plan_item pi ON pi.plan_item_key = tl.plan_item_key
WHERE tl.row_current_flag = 1 and pi.project_key  = proj;
```

Para esta consulta se itera en cada uno de los proyectos - representado con "proj" - obtenidos de plan_item y se seleccionan los tiempos registrados en la tabla time_log_fact_hist.

Apéndice D

Consultas de Análisis

D.1. Primer Enfoque

D.1.1. Obtener la lista de proyectos

```
SELECT distinct(pi.project_key) FROM plan_item pi
JOIN defect_log_fact_hist d where d.plan_item_key=pi.plan_item_key
```

D.1.2. Obtener el total de proyectos

```
SELECT count(*) FROM (SELECT distinct(pi.project_key) FROM plan_item pi
JOIN defect_log_fact_hist d where d.plan_item_key=pi.plan_item_key) as b
```

D.1.3. Obtener la lista de componentes

```
SELECT distinct(w.wbs_element_key) FROM time_log_fact_hist tl
INNER JOIN plan_item pi ON pi.plan_item_key = tl.plan_item_key
INNER JOIN wbs_element w ON w.wbs_element_key = pi.wbs_element_key
INNER JOIN phase ph ON ph.phase_key = pi.phase_key
WHERE tl.row_current_flag = 1 AND pi.task_key IS NOT NULL AND pi.project_key =
    proj
```

D.1.4. Obtener el total de componentes

```
SELECT count(*) FROM (SELECT distinct(w.wbs_element_key) FROM time_log_fact_hist
    tl
INNER JOIN plan_item pi ON pi.plan_item_key = tl.plan_item_key
INNER JOIN wbs_element w ON w.wbs_element_key = pi.wbs_element_key
INNER JOIN phase ph ON ph.phase_key = pi.phase_key
WHERE tl.row_current_flag = 1 AND pi.task_key IS NOT NULL AND pi.project_key =
    proj as b
```

D.1.5. Obtener el tamaño de cada componente

```
SELECT ifnull(sum(s.size_added_AND_modified), 0) FROM size_fact_hist s
INNER JOIN plan_item pi ON pi.plan_item_key = s.plan_item_key
INNER JOIN wbs_element w ON w.wbs_element_key = pi.wbs_element_key
INNER JOIN measurement_type m ON m.measurement_type_key = s.measurement_type_key
INNER JOIN size_metric e ON e.size_metric_key = s.size_metric_key
INNER JOIN phase ph ON ph.phase_key = pi.phase_key
WHERE s.row_current_flag = 1 AND m.measurement_type_name = 'Actual' AND e.size_
    metric_name = 'Lines of Code' AND (ph.phase_name like 'Code' or ph.phase_name
     like 'TSP - Code') AND pi.project_key = proj AND pi.wbs_element_key = comp
```

D.1.6. Obtener el log de tiempos de cada componente para una fase dada

```
SELECT ifnull(sum(tl.time_log_delta_minutes), 0) FROM time_log_fact_hist tl
INNER JOIN plan_item pi ON pi.plan_item_key = tl.plan_item_key
INNER JOIN wbs_element w ON w.wbs_element_key = pi.wbs_element_key
INNER JOIN phase ph ON ph.phase_key = pi.phase_key
WHERE tl.row_current_flag = 1 AND pi.project_key = proj AND pi.wbs_element_key
    = comp AND (ph.phase_name like 'Fase')
```

A pesar de que todos los proyectos aplican el proceso TSP, cada uno puede personalizar sus propias fases, ya sea modificando el nombre de una fase o agregando o quitando fases. Esto implica que cuando se quiere realizar consultas sobre un fase determinada, es necesario realizar un mapeo entre fases de cada proyecto.

A continuación se nombran el mapeo de las fases involucradas:

1. Fase de diseño: *Detailed Design, TSP - Detailed Design*
2. Fase de revisión de diseño: *Detailed Design Review, TSP - Detailed Design Review*
3. Fase de código: *Code, TSP - Code*
4. Fase de revisión de código: *Code Review, TSP - Code Review*
5. Fase de test de integración: *Build AND Integration Test, Integration Test*
6. Fase de test de sistema: *System Test*

D.1.7. Obtener el total de defectos en TI y ST

```
SELECT ifnull(count(*), 0) FROM defect_log_fact_hist d
INNER JOIN plan_item pi ON pi.plan_item_key = d.plan_item_key
INNER JOIN wbs_element w ON w.wbs_element_key = pi.wbs_element_key
INNER JOIN phase ph ON ph.phase_key = d.defect_removed_phase_key
WHERE d.row_current_flag = 1 AND pi.project_key = proj, AND pi.wbs_element_key
    = comp AND (ph.phase_name like 'System Test' or ph.phase_name like 'Build
   AND Integration Test' or ph.phase_name like 'Integration Test'
```

D.2. Segundo Enfoque

Dado que para la realización de este enfoque, varias consultas del enfoque anterior se usaron, solo se especifican las que no se repiten. Las repetidas son: Obtener la Lista de Proyectos, Obtener el Total de Proyectos, Obtener la Lista de Componentes, Obtener el Total de Componentes y

Obtener el Log de Tiempos de cada Componente para una Fase Dada, para la cual se nombran las fases involucradas en este caso:

1. Fase de diseño: *Detailed Design, TSP - Detailed Design*
2. Fase de revisión de diseño: *Detailed Design Review, TSP - Detailed Design Review*
3. Fase de código: *Code, TSP - Code*
4. Fase de revisión de código: *Code Review, TSP - Code Review*
5. Fase de compilación: *Compile*

Al igual que en el enfoque anterior, es necesario realizar un mapeo entre fases de cada proyecto.

D.2.1. Verificar existencia del ordinal de la fase de compilación

```
SELECT distinct ifnull(count(po.phase_ordinal),0) FROM plan_item pi
INNER JOIN phase ph ON ph.phase_key = pi.phase_key
INNER JOIN phase_order po ON po.phase_key = ph.phase_key
WHERE pi.project_key = proj  AND ph.phase_name like 'Compile'
```

D.2.2. Obtener el ordinal de la fase de compilación

```
SELECT distinct po.phase_ordinal FROM plan_item pi
INNER JOIN phase ph ON ph.phase_key = pi.phase_key
INNER JOIN phase_order po ON po.phase_key = ph.phase_key
WHERE pi.project_key = proj,  AND ph.phase_name like 'Compile'
```

D.2.3. Obtener el total de defectos inyectados antes de la fase de compilación

```
SELECT ifnull(count(d.defect_injected_phase_key), 0)
FROM defect_log_fact_hist d
INNER JOIN plan_item pi ON pi.plan_item_key = d.plan_item_key
INNER JOIN wbs_element w ON w.wbs_element_key = pi.wbs_element_key
INNER JOIN phase ph ON ph.phase_key = d.defect_injected_phase_key
INNER JOIN phase_order po ON po.phase_key = d.defect_injected_phase_key
WHERE d.row_current_flag = 1 AND pi.project_key  =  proj  AND pi.wbs_element_key
    = comp AND po.phase_ordinal < ordinal
```

D.2.4. Obtener el total de defectos removidos antes de la fase de compilación

```
SELECT ifnull(count(d.defect_removed_phase_key), 0)
FROM defect_log_fact_hist d
INNER JOIN plan_item pi ON pi.plan_item_key = d.plan_item_key
INNER JOIN wbs_element w ON w.wbs_element_key = pi.wbs_element_key
INNER JOIN phase ph ON ph.phase_key = d.defect_removed_phase_key
INNER JOIN phase_order po ON po.phase_key = d.defect_removed_phase_key
WHERE d.row_current_flag = 1 AND pi.project_key  =  proj  AND pi.wbs_element_key
    = comp AND po.phase_ordinal < ordinal
```

Bibliografía

[1] Download software process dashboard. http://www.processdash.com/download. Accessed: 2015-12-04. A

[2] Mysql-mysql community y mysql workbench. http://dev.mysql.com/downloads/. Accessed: 2015-12-04. 1

[3] Plotly-paquete para crear gráficas en r. https://plot.ly/r/getting-started/. Accessed: 2015-12-04. 11

[4] R-r version 3.1.0 para windows. http://cran.r-project.org/bin/windows/base/. Accessed: 2015-12-04. 3

[5] Rstudio-ambiente de desarrollo para r. http://www.rstudio.com/. Accessed: 2015-12-04. 5

[6] Rtools-paquete de herramientas para r. //cran.r-project.org/bin/windows/Rtools/. Accessed: 2015-12-04. 4

[7] Software process dashboard. http://www.processdash.com/. Accessed: 2015-12-04. 3.1

[8] Sue Carroll and Taz Daughtrey. *Fundamental Concepts for the Software Quality Engineer*. William A. Tony, 2007. 5.1

[9] A Blanton Godfrey. *Juran's quality handbook*. McGraw Hill, 1999. 1

[10] W.S. Humphrey. A personal commitment to software quality. Technical report, Carnegie Mellon University, 1994. 1

[11] W.S. Humphrey. The personal software process (psp). Technical report, Carnegie Mellon University, 2000. 1.1, 2.2, 2.3

[12] W.S. Humphrey. *Team Software Process (TSP)*. 2000. 1.1, 2.4.1

[13] W.S. Humphrey, T.A. Chick, W.R. Nichols, and M. Pomeroy-Huff. The team software process body of knowledge (tsp bok). Technical report, Carnegie Mellon University, 2010. 2.1

[14] James Over. Software process overview, 2009. Slides. 2