Herramienta de soporte para análisis de datos del Personal Software Process (PSP)

Proyecto de grado

Angie Stephanie Lecot Emaldi

Tutor: Diego Vallespir

Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
Marzo de 2016

Resumen del trabajo

Este documento presenta el proyecto de grado “Herramienta de soporte para análisis de datos del *Personal Software Process* (PSP)” realizado en el Instituto de Computación de la Facultad de Ingeniería de la Universidad de la República.

El *Personal Software Process (PSP)* es un proceso que guía el desarrollo de software de forma individual. Su principal objetivo es ayudar a gestionar y mejorar la forma en que el individuo realiza su trabajo. Está compuesto por un conjunto de métodos, instrucciones y formularios que indican cómo planificar, medir y gestionar el trabajo. Se estructura en fases y estas son: Planificación, Diseño, Revisión de diseño, Codificación, Revisión de código, Compilación, Pruebas unitarias y Postmortem. Es enseñado en un curso, en el que se introducen de forma paulatina los elementos y fases mencionadas. Durante el curso los estudiantes realizan una serie de ejercicios (programas) en los cuales registran datos de sus mediciones. Existen algunas herramientas que dan soporte a la recolección de datos; una de ellas es el *PSP Student Workbook*. Cada estudiante utiliza una instancia de la herramienta, por lo que se tiene una base de datos por cada uno de ellos.

Los programas realizados tienen el mismo dominio de aplicación y, al generarse de forma consecutiva, se está ante la presencia de programación repetitiva.

Watts S. Humphrey, en el *PSP*, propone que para gestionar y así mejorar la calidad del trabajo realizado y, por lo tanto la performance, no solo se debe medir la calidad sobre el producto, sino que se debe medir la calidad sobre el propio trabajo.

En *The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers* se concluye que la aplicación del *PSP* mejora la calidad del producto. Dicho de otra forma, mejora la performance del sujeto. Este resultado involucra principalmente dos factores que a priori pueden generar esta mejora: el seguimiento del proceso y la programación repetitiva.

Este proyecto tiene, como uno de sus cometidos, crear una solución que dé soporte a la integración de datos que se recolectaron de forma independiente durante la aplicación del *PSP* con la herramienta *PSP Student Workbook*. La solución creada se compone de una estructura de base de datos con motor MySQL y una herramienta desarrollada en Java que migra e integra los datos desde las bases independientes, en las que se ingresaron, a la base de datos mencionada. Para validar su funcionamiento, se demuestra su aplicabilidad realizando una integración de un conjunto de datos reales.

Otro de los cometidos es realizar un análisis estadístico sobre los datos integrados. El análisis se realiza con dos conjuntos de datos trabajados de forma independiente. Por un lado se tienen datos recolectados en cursos del *PSP* en el periodo comprendido entre octubre de 2005 y mayo de 2010. Los cursos fueron dictados por el *Software Engineering Institute* (SEI) de la universidad *Carnegie Mellon* (CMU) o por *partners* del SEI. El otro conjunto fue obtenido en tres experimentos controlados efectuados en la Facultad de Ingeniería de la Universidad de la República. Se realizaron en los años 2012, 2013 y 2014 y participaron 12, 10 y 14 estudiantes, respectivamente. A diferencia de los cursos, en los experimentos los estudiantes no aplicarón las técnicas del proceso, pero sí registraron datos e hicieron los mismos ejercicios.

Para el análisis estadístico se plantean dos subobjetivos. Uno de ellos es evaluar qué tan efectivas son las fases de Revisión para remover defectos y cuál es el costo asociado. El otro es determinar si la mejora de la performance del estudiante es producto de aplicar un proceso que guía su trabajo (*PSP*) o de la programación repetitiva. La performance se evalúa utilizando dos medidas: una que mide la calidad sobre producto y otra que mide la calidad sobre el proceso. La calidad sobre el producto se mide utilizando la densidad de defectos en las Pruebas unitarias (UT); mientras que la calidad del proceso se mide a través de la relación (ratio) entre los tiempos dedicados a las fases de Diseño y Codificación, definido como ratio TCod/TDLD. Los datos a utilizar en el primero se recolectaron en los cursos del PSP y en el segundo en los experimentos.

Índice de contenido

Capítulo 1 - Introducción

El continuo avance tecnológico le ha dado más importancia al software. En la actualidad los negocios y muchas actividades que se realizan habitualmente dependen de él. Esto ha conducido a que cada vez los usuarios tengan más exigencias sobre su calidad y sus funcionalidades. A su vez la competitividad marca que se debe producir software de calidad en el menor tiempo y al más bajo costo posible.

En general la industria del software, para asegurar esa calidad, se basa en buscar y corregir errores en el software una vez construido. Contrario a ello, algunas líneas de trabajo han demostrado que para obtener software de calidad hay que enfocarse en la mejora de los procesos utilizados para su desarrollo [1].

Para ayudar a los profesionales a desarrollar software de calidad es que surge el *Personal Software Process (PSP).* El *PSP* es un proceso que guía de forma individual el desarrollo de software. Está compuesto por un conjunto de métodos, instrucciones y formularios que indican cómo planificar, medir y gestionar el trabajo. Está estructurado en fases y estas son: Planificación, Diseño, Revisión de diseño, Codificación, Revisión de código, Compilación, Pruebas unitarias y Postmortem. Su objetivo es mejorar la forma en que el profesional realiza su trabajo y por consiguiente la calidad del software desarrollado. Watts S. Humphrey, plantea que: “Para producir software de calidad, cada individuo que participa en su construcción debe hacer su trabajo con calidad” [1]. Para medir la calidad del trabajo realizado, el individuo no solo debe medir la calidad sobre el producto creado, sino que debe medirla sobre el propio trabajo. Esto se plantea en *PSP: A Self-Improvement Process for Software Engineers* [2].

El *PSP* es enseñado en un curso, donde mediante la realización de ejercicios (programas) se va induciendo al estudiante en las diferentes fases y métodos. Durante los cursos los estudiantes registran datos de sus mediciones. Para dar soporte a la recolección de datos, se dispone de una herramienta desarrollada en Access, *PSP Student Workbook.* Cada estudiante utiliza una instancia de la herramienta, por eso los datos quedan almacenados en una base de datos Access local. Una vez finalizada la carga de la misma, se realiza una exportación a un fichero .mdb.

Uno de los cometidos de este trabajo consiste en generar una solución que dé soporte a la integración de los datos que se ingresaron de forma independiente por cada individuo, durante la aplicación del *PSP,* utilizando la herramienta *PSP Student Workbook*. También pretende demostrar su aplicabilidad en una migración de datos reales y un análisis estadístico a partir de ellos.

El análisis de los datos complementa otros trabajos realizados en la misma línea. Por un lado, el trabajo realizado en *Quality is Free, Personal Reviews Improve Software Quality at No Cost,* en el que se estudia la eficiencia de las Revisiones en la remoción de defectos [3]. Por otro, *Demostrating the Impact of the PSP on Software Quality and Effort: Eliminating the Programming Learning Effect* donde se concluye que la mejora en la calidad del software se debe a seguir un proceso disciplinado como lo es el *PSP* y no por la programación repetitiva que se da durante la enseñanza del proceso [4]. En el primer caso se trabaja con datos recabados durante diferentes cursos del *PSP*. En los cursos participaron unos 130 estudiantes y fueron dictados por el *Software Engineering Institute (SEI)* de la universidad *Carnegie Mellon (CMU)*. Dichos cursos se dictaron entre octubre del 2005 y mayo del 2010. En el segundo caso, se trabaja con datos obtenidos en tres experimentos controlados efectuados en la Facultad de Ingeniería de la Universidad de la República. Se realizaron en los años 2012, 2013 y 2014 y participaron 12, 10 y 14 estudiantes, respectivamente. A diferencia de los cursos, los estudiantes no aplicaron los métodos del *PSP* pero sí hicieron los mismos ejercicios y registraron datos.

1.1 - Motivación

Dentro de las líneas de investigación del Grupo de Ingeniería de Software (Gris) se encuentra el estudio de procesos de desarrollo de software y su mejora.

Desde hace unos años, se vienen realizando trabajos que tienen como objeto el estudio del *PSP.* En ellos se estudian datos recabados por los individuos en diferentes ámbitos de aplicación. Para dichos estu-

dios se han utilizado datos provenientes de cursos dictados del *PSP* y en otros casos los datos se obtuvieron realizando experimentos controlados donde participaron estudiantes. Para analizar los datos recolectados durante la aplicación del *PSP,* se cuenta con archivos de base de datos independientes, uno por cada estudiante. Por lo cual, para facilitar el acceso y la manipulación de los mismos, surge la necesidad de contar con una herramienta que los integre en una única base de datos.

Está comprobado que el *PSP* mejora la calidad del software producido [5]. En el proceso intervienen principalmente dos factores que a priori pueden generar esta mejora: el seguimiento del proceso y la programación repetitiva. Una de las medidas utilizadas para evaluar la calidad sobre un producto de software, es la cantidad de defectos encontrados en las Pruebas unitarias (UT). Existe una correspondencia directa entre los defectos que llegan a la fase de UT y el remanente que queda en el producto una vez finalizado. O sea: cuantos menos defectos lleguen a la fase de UT, menos defectos tendrá el producto final. Una medida para medir la calidad sobre el proceso realizado es la relación entre los tiempos dedicados a Diseño y a Codificación.

En uno de los trabajos anteriores no se encuentra evidencia de que la mejora en la calidad sea consecuencia de la repetición de programas en el mismo dominio de aplicación, por lo cual se da por la aplicación del proceso [4]. Alineado con el estudio de las mejoras que genera la aplicación del *PSP,* surge la necesidad de complementar otros estudios realizados con anterioridad.

1.2 - Objetivos

El proyecto consta de dos objetivos principales. El primero consiste en proporcionar una solución para la integración de las bases de datos distribuidas, que fueron cargadas por cada estudiante durante la aplicación del *PSP,* y validar su funcionamiento con la migración de un conjunto de datos. El segundo, realizar un análisis estadístico sobre datos registrados por estudiantes durante los cursos del *PSP* y los experimentos.

En el primer objetivo, la solución debe incluir la estructura y el motor de la base de datos donde se integren las bases de datos individuales, el desarrollo de una herramienta que los migre y la especificación del proceso necesario para realizarlo. El funcionamiento debe validarse integrando un conjunto de archivos “reales” generados por la herramienta *PSP Student Workbook* al aplicar el *PSP.*

Para el segundo objetivo, el análisis estadístico de los datos, se plantean dos subobjetivos. El primer subobjetivo consiste en evaluar qué tan efectivas son las fases de Revisión para remover defectos y cuál es el costo asociado. El segundo subobjetivo consiste en analizar si la mejora en la performance del estudiante, en los programas desarrollados aplicando el *PSP*, es producto de la aplicación de un proceso disciplinado (como lo es el *PSP*) o por la creación de varios programas en el mismo dominio de aplicación de forma consecutiva (programación repetitiva). Los datos a utilizar en el primer subobjetivo se recolectaron en los cursos del *PSP* y en el segundo los experimentos.

En el primero subobjetivo del análisis estadístico, la efectividad se estudiará observando el comportamiento de la inyección y remoción de defectos en las distintas fases del proceso. Se comparará la cantidad de defectos encontrados y removidos en las fases de Revisión respecto al resto de las fases. La remoción de defectos tiene un costo asociado. En este caso se calculará, para cada una de las fases, el costo promedio de remover un defecto.

En el segundo subobjetivo del análisis estadístico, para evaluar si la mejora en la performance se debe a la aplicación del proceso *PSP* o a la programación repetitiva, se estudiará qué ocurre en los experimentos donde solo esté presente este último factor. Para determinar si en los experimentos hay mejora en la performance*,* se deberá observar la evolución de la calidad del trabajo, o sea si mejora o no. La calidad se evaluará utilizando dos medidas (indicadores): una es aplicada sobre el producto y la otra sobre el proceso. La medida que aplica sobre la calidad del producto, es la densidad de defectos en las Pruebas unitarias (UT), calculada como la cantidad de defectos encontrados cada mil líneas de código. Mientras que la que aplica sobre el proceso, es la relación (ratio) entre el tiempo dedicado al Diseño (TDLD) y el tiempo de dedicado a Codificación (TCod), definida como: ratio TCod/TDLD. Esta medida es propuesta por Humphrey en el *PSP* [2].

En uno de los trabajos que se complementa en este proyecto, se analizó la densidad de defectos en UT utilizando los datos del experimento realizado en 2012 [4]. Por lo cual este trabajo buscará extender dicho estudio a los otros dos experimentos considerando también otra medida: el ratio TCod/TDLD.

1.3 - Trabajo realizado

Al momento de comenzar el proyecto, se contaba con dos herramientas cuya principal funcionalidad era la de integrar en una única base de datos, las bases (Access) generadas individualmente por cada estudiante. Una de estas herramienta está desarrolla en Java, AllPSPStudentData_v1, y la otra desarrollada en Genexus. La herramienta Java utiliza el motor de base de datos HSQLDB y la desarrollada en Genexus uno MySQL.

En primera instancia se investiga cual es más apropiada para reutilizar en este trabajo, dando por resultado un mix de ambas. Se reutiliza la herramienta AllPSPStudentData_v1 y la estructura de la base MySQL. Se toma esta decisión a raíz de que la herramienta Genexus presenta fallas y para modificarla se debe contar con la licencia del producto, mientras que la Java no la necesita. En cuanto a la estructura, en la base HSQLDB no se contemplan todas las tablas de las bases individuales Access, mientras que en la MySQL sí. Por lo cual se opta por esta última. Luego se selecciona como motor de base de datos MySQL y se realiza un análisis de las estructura de la base de datos Access y la MySQL. De dicho análisis surgen algunos cambios que se deben realizar a la estructura MySQL.

Posteriormente, se planifica el “proceso de migración” el cual incluye: generar la estructura de la base de datos (en la cual se unificarán las bases Access) mediante la ejecución de algunos *scipts* y la migración de los datos utilizando la herramienta a generar. Se modifica la herramienta AllPSPStudentData_v1 para que migre datos contenidos en las tablas de las bases individuales (Access) a la integrada (MySQL). Se agregan opciones para poder visualizar y modificar la parametrización utilizada para acceder a ambas bases y la posibilidad de ejecutar algunos *scripts* desde la herramienta en vez de utilizar una herramienta de gestión de base de datos.

Para validar el proceso de migración y cargar los datos para el análisis estadístico, se realizan dos migraciones independientes. En primer lugar, se migran las bases individuales generadas durante los cursos y se replican cálculos realizados en el estudio *Quality is Free, Personal Reviews Improve Software Quality at No Cost* para verificar si se obtienen los mismos resultados [3]. En segundo lugar se migran las bases generadas durante experimentos controlados. A partir de los dos conjuntos se realiza el análisis especificado en los objetivos.

Para hacer el análisis estadístico sobre los datos provenientes de los experimentos, fue necesario estudiar algunos test estadísticos con el fin de conocer sus condiciones de aplicabilidad y cómo realizar cada uno de ellos. Dentro de los test estadísticos a utilizar se encuentra el Test de rangos de Wilcoxon de dos colas para muestras apareadas y Test de rangos de Wilcoxon de una cola para muestras apareadas. La ejecución de estos test es complementada con el cálculo de la magnitud de las variaciones en cada test, la d de Cohen.

A su vez fue necesario aprender el uso de algunas funcionalidades del software estadístico R y el lenguaje R utilizado para implementaciones en el mismo. Para que los resultados obtenidos puedan ser replicados en los mismos conjuntos de datos u otros, se generaron *scripts* en los cuales están implementados y documentados cada uno de los cálculos realizados.

1.4 - Organización del documento

El presente documento se encuentra estructurado en seis capítulos, incluida la introducción. En el segundo capítulo ***Personal Software Process (PSP)*** se detalla en qué consiste el proceso y cómo se encuentra estructurado, así como también los principales conceptos que se mencionan a lo largo del informe.

En los siguientes dos capítulos se desarrolla la investigación realizada y la solución implementada para la integración, presentándose en el tercer capítulo la ***Elección de la herramienta y sistema gestor de***

base de datos. En este, se describe cómo es el proceso de búsqueda de la herramienta y el motor de base de datos adecuado para el almacenamiento de los datos. Mientras que en ***Diseño de la solución para la integración de los datos*** se presenta el proceso de integración de forma completa, describiendo paso a paso cómo debe ser ejecutado. Se incluye también la especificación de la estructura de la base de datos y la herramienta generada para la migración de estos.

En el capítulo de ***Análisis*** se presentan los resultados obtenidos en el análisis y sus conclusiones, incluyendo una especificación del conjunto de datos utilizados y los análisis estadísticos realizados sobre cada uno de ellos.

Finalmente, en el último capítulo, ***Conclusiones y trabajo futuro,*** se describen los resultados obtenidos y las conclusiones, dejando abiertos algunos planteos para trabajos a futuro.

El documento también consta de siete anexos. En el ***Anexo 1*** se presenta una tabla de equivalencia entre las tablas de la base de datos que contienen originalmente los datos y las tablas a las que se migraron. Por otro lado, el ***Anexo 2*** detalla las claves primarias y foráneas de cada una de las tablas, marcando los cambios en cada una de ellas.

Para facilitar la reutilización de la base y de la herramienta, en el ***Anexo 3,*** se indican los pasos a seguir para realizar una migración de datos desde “cero”. En el ***Anexo 4,*** se describe el formato del archivo de log que genera la herramienta al momento de migrar los datos. En el ***Anexo 5,*** se presentan algunos inconvenientes de calidad de datos y “problemas conocidos” en la ejecución de la herramienta. Por último, en ***Anexo 6*** *y* ***Anexo7*** se presentan cálculos de la d de Cohen obtenidos en el análisis estadístico.

Capítulo 2 - *Personal Software Process* (PSP)

En este capítulo se presenta el *Personal Software Process (PSP),* se describe el proceso, sus principios, objetivos y los principales conceptos.

En primera instancia se aborda el contexto histórico en el cual surge, su principio fundamental y sus objetivos. Luego se describen los elementos y actividades involucrados en el proceso relacionándolo con su enseñanza. Este capítulo se basa principalmente en el reporte técnico realizado por Watts S. Humphrey [1], así como también en cursos impartidos del *PSP*.

2.1 - Introducción

El *Personal Software Process (PSP)* es un proceso que guía el desarrollo de software de forma individual. Su objetivo principal es ayudar a gestionar y mejorar la forma en que el individuo realiza su trabajo. El mismo está conformado por un conjunto de métodos, formularios y *scripts*, los cuales indican cómo planificar, medir y gestionar el trabajo [1].

2.1.1 - Contexto

A mediados del siglo xx la industria se basaba únicamente en el *testing* como herramienta para asegurar la construcción de productos de calidad, la estrategia aplicada era la de *Test and Fix.* Dicha estrategia consiste en buscar y corregir defectos sobre el producto terminado, sin considerar el proceso realizado para producirlo.

Posteriormente, en los 70-80 W. Edwards Deming y J.M. Juran plantearon un cambio de enfoque. El cambio apunta a mejorar la forma en que las personas realizan su trabajo, o sea enfocarse en el proceso que utilizan. Demostraron y convencieron a la industria estadounidense de que la estrategia *Test and Fix* es costosa y poco efectiva. Ante ello, la industria se volcó al nuevo enfoque.

La comunidad del software no se alineó a dicho enfoque y continuó basandose en *Test and Fix.* Ello impacta directamente en la calidad del software, el calendario y los costos.

En los 80 estaba instalada la “crisis del software”, donde la calidad y los costos del software no eran lo esperado y no se cumplía con los plazos previstos. A partir de ese momento, hay una creciente necesidad de contar con software más complejo y de mayor tamaño. En consecuencia, surgen enfoques, procesos y modelos que van en línea con lo planteado por Deming y Juran.

En 1987 surge el Modelo de Capacidad y Madurez (CMM), un modelo de evaluación de los procesos de una organización. Su objetivo es mejorar los procesos relacionados con el desarrollo y mantenimiento del software con el fin de obtener productos de calidad. Asume que los individuos realizan trabajo de forma eficiente, por lo cual no brinda indicatrices a seguir de forma individual.

Watts S. Humphrey observó que lo asumido no era correcto y, sumado a que el trabajo del individuo afecta a todo el proceso, comenzó a estudiar cómo aplicar las ideas del CMM en un proceso individual de desarrollo. Esto llevó a que en 1995 publicara el *PSP*.

2.1.2 - Objetivos y Principios del PSP

El principio detrás del *PSP* es que "Para producir sistemas de software de calidad, cada ingeniero que trabaja en el sistema debe hacer su trabajo con calidad" [1]. Tiene como objetivo ayudar a mejorar el desempeño personal del individuo.

Para lograrlo, debe seguir un proceso disciplinado que lo ayude a controlar y medir su trabajo. El *PSP* busca que el individuo, conociendo sus estadísticas, pueda gestionar y planificar mejor su trabajo. Y así, mejora su rendimiento, efectividad y, como resultado final, genera productos con mayor calidad.

Su estructura está definida sobre la base de que "es más eficiente prevenir los defectos que encontrarlos y corregirlos" y "es menos costoso encontrar y corregir defectos en etapas más tempranas del desarrollo" [1]. Puede aplicarse de forma independiente al lenguaje de programación y metodología utilizados.

2.2 - Descripción del proceso

El proceso se encuentra estructurado en varias fases por las cuales se va a transitar a medida que se avanza. Las fases son: Planificación, Diseño, Revisión de diseño, Codificación, Revisión de código, Compilación, Pruebas unitarias y Postmortem[1]. Se debe notar que el nombre de la fase no es completamente análogo a la actividad que se realiza. A modo de ejemplo, estando en la fase de Codificación se puede detectar un error de diseño. En este caso, cronológicamente, se está en la fase de Codificación pero se realiza una actividad de diseño para corregir el error.

También proporciona un conjunto de elementos para guiar y facilitar el trabajo. Los elementos son: instrucciones, formularios, *logs* y estándares. Cada uno es un insumo de entrada o producto de salida para las fases.

Instrucciones(Scripts): Son la "guía" para que el individuo siga el proceso de forma correcta. Se definen actividades, medidas a tomar y registros a generar en cada fase.

Formularios y logs: Los formularios son plantillas para dar soporte a las actividades y al registro de datos que se almacenan en los *logs*. Se registran: tiempos, defectos, tamaños, etc. Un formulario clave es el resumen del plan. En él se mantiene la información planificada y real del proyecto. Se divide en: tamaños de programa, tiempos, cantidad de defectos inyectados y removidos por fase y total.

*Estándares: S*e utilizan algunos estándares, entre ellos: estándares de codificación (dependiendo del lenguaje utilizado), clasificación de defectos y para medir el tamaño del programa.

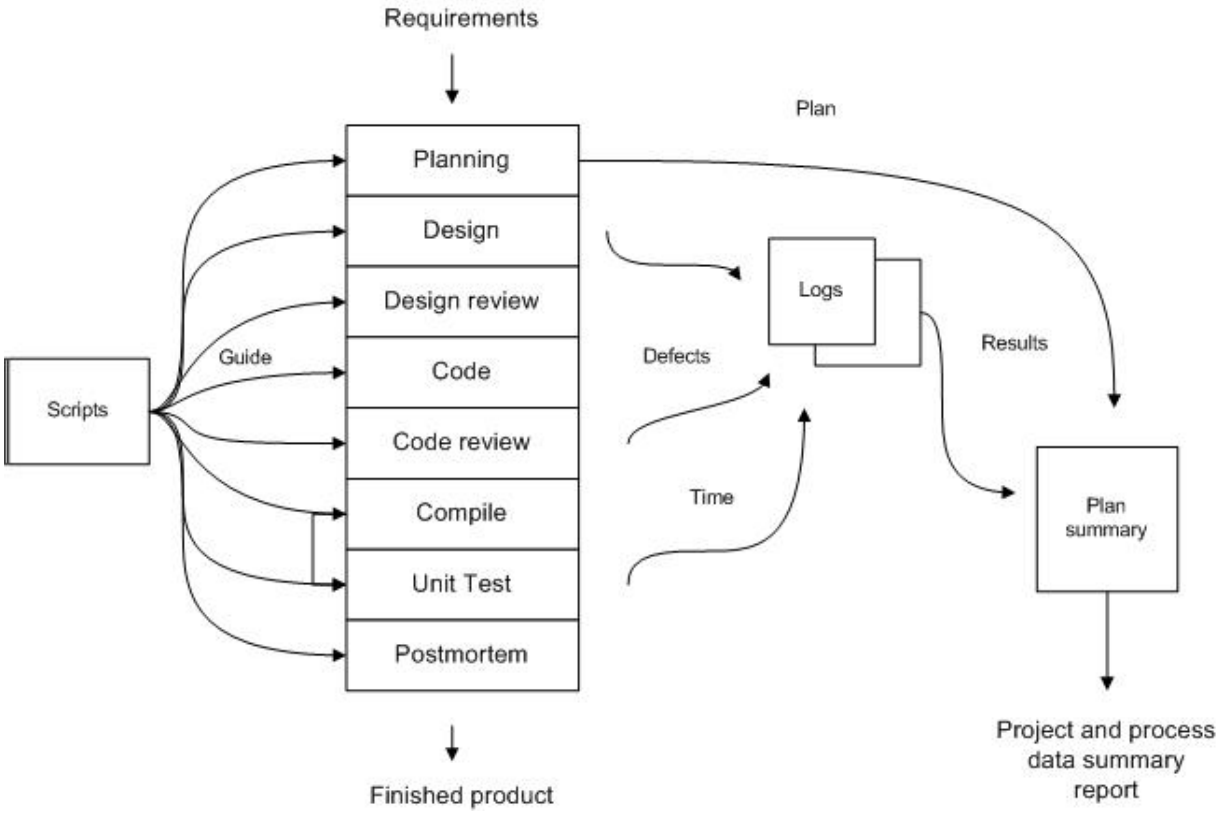

Figura 1 - *Estructura general del PSP* [2]

1 Las abreviaturas en inglés son: *Planning*(PLAN), *Detailed Design*(DLD), *Detailed Design review* (DR), *Code*(CODE), *Code review* (CR), *Compile* (Comp), *Unit Test*(UT), *Postmortem*(PM).

2 La figura corresponde a la versión completa del PSP.

El proceso inicia con la planificación del desarrollo. Sus insumos son: la descripción de los requisitos, el *script* de planificación y el resumen del plan. La planificación se registra en el resumen del plan. Posteriormente están las fases de construcción del software: Diseño, Revisión de diseño, Codificación, Revisión de código, Compilación y Pruebas unitarias. En cada una se dispone de un *script* que guía el trabajo y se generan elementos que son insumo en las fases sucesivas. Se toman medidas de tamaño de programa, tiempos dedicados a cada fase, y detalles de defectos inyectados y removidos por fase. Se almacenan en *logs* de tiempos y defectos. Finaliza con la fase Postmortem, donde se mide el tamaño del programa, y se unifican los datos de tiempos y defectos. Los elementos de salida son el producto construido y el resumen del plan completo. En la Figura 1 se ilustra la estructura y los elementos del proceso.

2.3 - Enseñanza del PSP

El PSP es enseñado en un curso donde, mediante la realización de ejercicios (programas o proyectos), se va induciendo al estudiante en las diferentes fases y métodos.

Es introducido en una serie de seis versiones incrementales del proceso, cada una tiene un procedimiento asociado. Las versiones son: PSP0, PSP0.1, PSP1, PSP1.1, PSP2 y PSP2.1.

Como se puede observar en la Figura 2, en la versión inicial del proceso PSP0 el estudiante hace su trabajo tal cual lo viene realizando. Lo único que se indica es que tome algunas medidas básicas de tiempos y defectos. La siguiente versión PSP0.1 únicamente agrega la utilización de un estándar de codificación (acorde al lenguaje utilizado) y la creación de una propuesta de mejora al proceso que está realizando. En ambas versiones se registran datos pero no se aplican las prácticas ni técnicas del PSP. Completan el nivel más básico, lo que es comparable con realizar ejercicios sin un proceso ya que las guías solo indican registro de datos. A partir de ellas, se adicionan nuevos métodos, fases, elementos, conceptos y actividades de gestión para las siguientes versiones (Ver Figura 2).

Los ejercicios son realizados dentro del marco de las guías y elementos que proporciona cada versión del PSP. La cantidad de programas a realizar varía según el curso y van desde ocho a diez ejercicios. El estudiante debe elegir y utilizar el mismo lenguaje de programación para todos los ejercicios que le sea conocido. De esta manera el aprendizaje del lenguaje no es un factor que incida en el proceso.

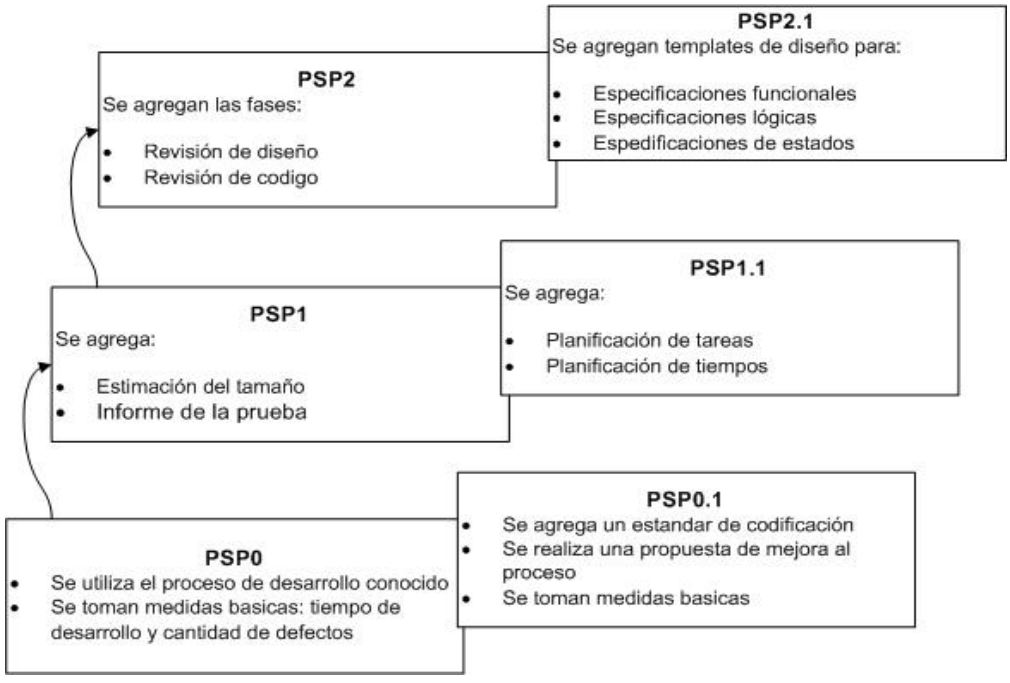

Figura 2 - Descripción de versiones del PSP

Algunas herramientas de software son utilizadas con el fin de facilitar el acceso a los elementos del proceso y que los registros sean más precisos. Los datos a utilizar en este trabajo se registraron utilizando una herramienta que da soporte a la recolección de datos, *PSP Student Workbook*. La misma da soporte para la toma de medidas y crea *logs* de los programas, acceso a *scripts* y formularios de las diferentes versiones del proceso. También incluye funciones para el análisis y material del curso. Utiliza una base de datos Access para almacenar los *logs*.

En la Figura 3 se muestra el formulario principal a partir del cual el estudiante accede a los proyectos y procesos[3] que tiene definido para realizar. Al seleccionar un proceso muestra sus fases. Por ejemplo, en el PSP2.1 figuran las Revisiones de diseño y código mientras que en el PSP0, no. Cada proceso tiene su conjunto de *scripts* que describe para cada fase del proceso los pasos a seguir.

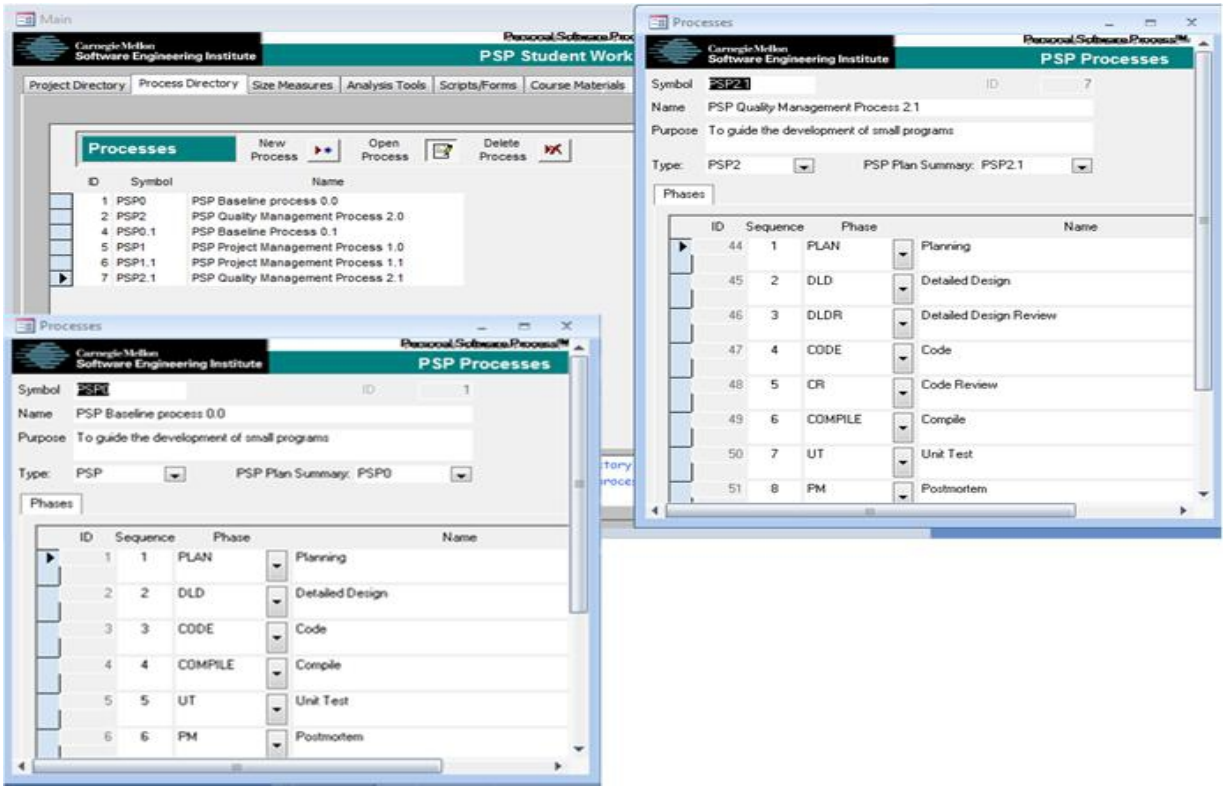

Figura 3 - Formulario principal PSP *Student Workbook*

El estudiante completa el resumen del plan durante la Planificación y al final del proyecto, ingresando los datos en los campos de plan y actual, respectivamente. Ambos casos se muestran en la Figura 4.

2.3.1 - Medidas y registros

Para mejorar la planificación el sujeto debe generar datos para analizar sus propias estadísticas. Para ello estima, mide y almacena los resultados. Los valores se asocian al proyecto, proceso y fase que está realizando el estudiante.

Tiempo en fase es el tiempo total de dedicación del estudiante a la fase. La fase puede involucrar intervalos de tiempo disjuntos, el estudiante debe registrarlos todos. En cada uno se registra la fecha y hora de inicio y fin; en caso de interrupción, se registra el tiempo y el motivo. El tiempo total es la suma de todos los "Delta" de la fase, donde "Delta" es el tiempo transcurrido entre la hora de inicio y fin, menos las interrupciones.

3 Corresponde a las versiones del PSP.

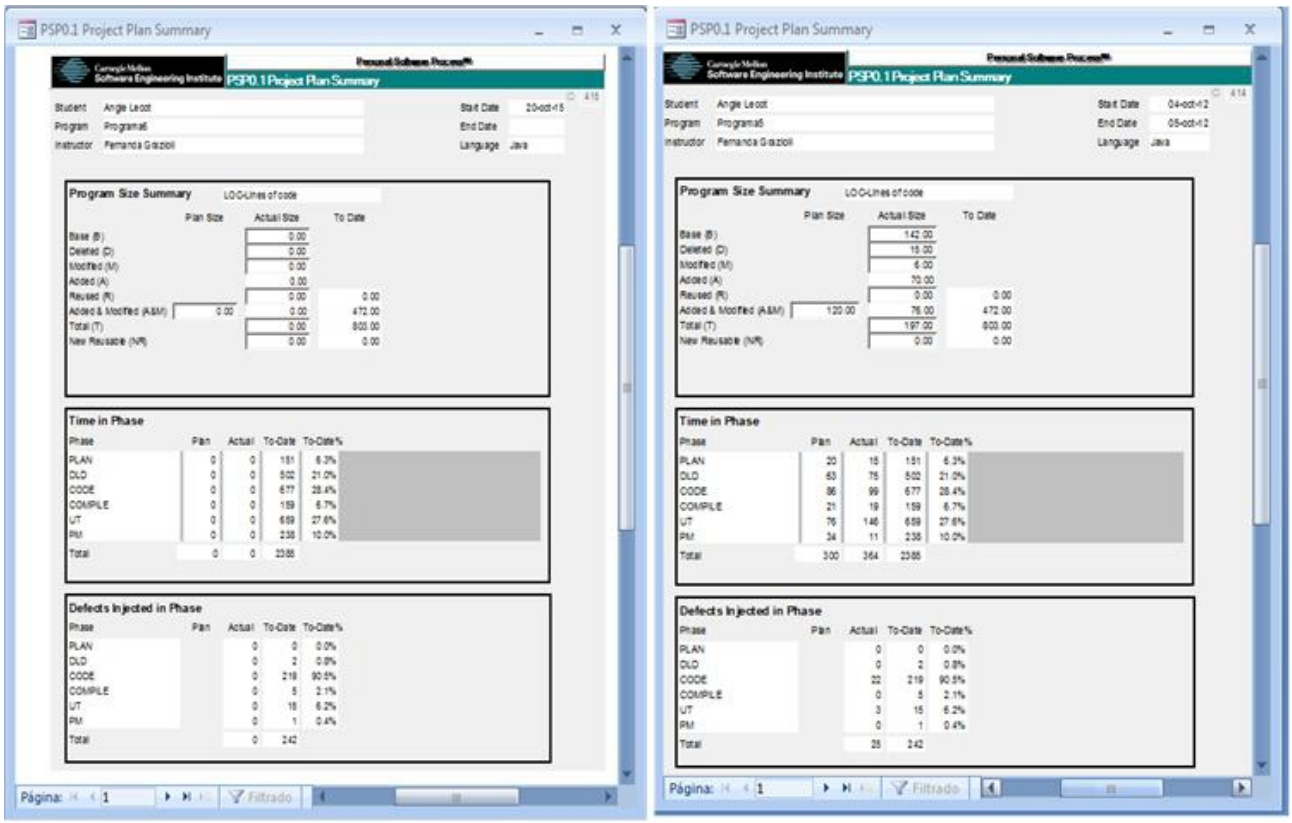

Figura 4 - Resumen del plan

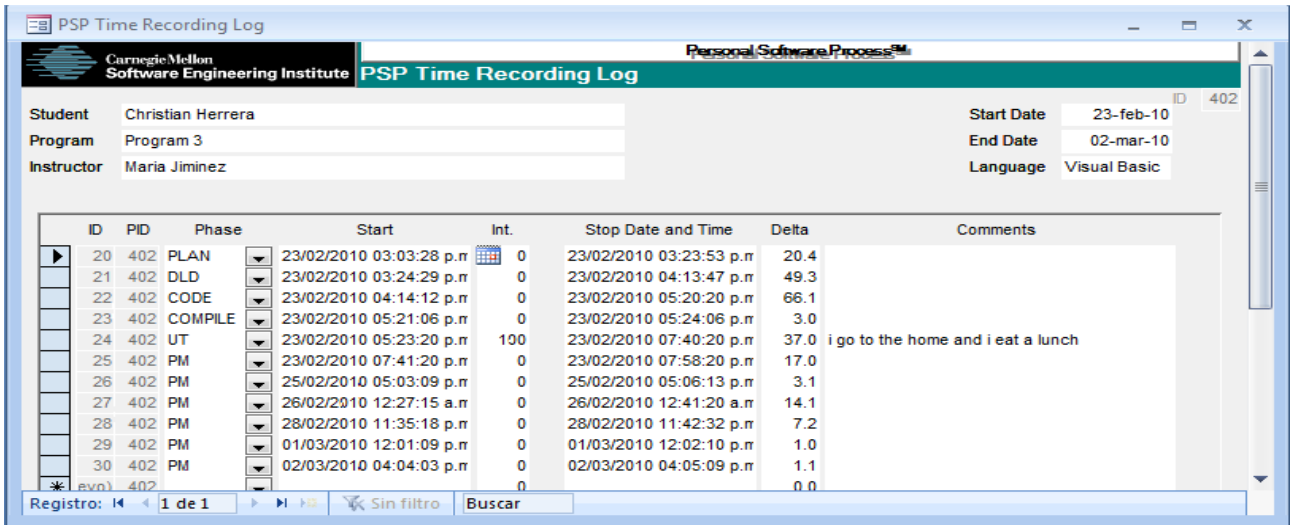

Figura 5 - Formulario para registro de tiempo en fase

Se registra información de los **Defectos** inyectados y removidos. Para cada uno, los datos ingresados son la fecha y hora en que se encuentra el defecto (*Date*), fases en que se inyecta y remueve, clasificación del defecto y descripción del error, y tiempo dedicado a encontrar y solucionar el defecto (*Fix time*).

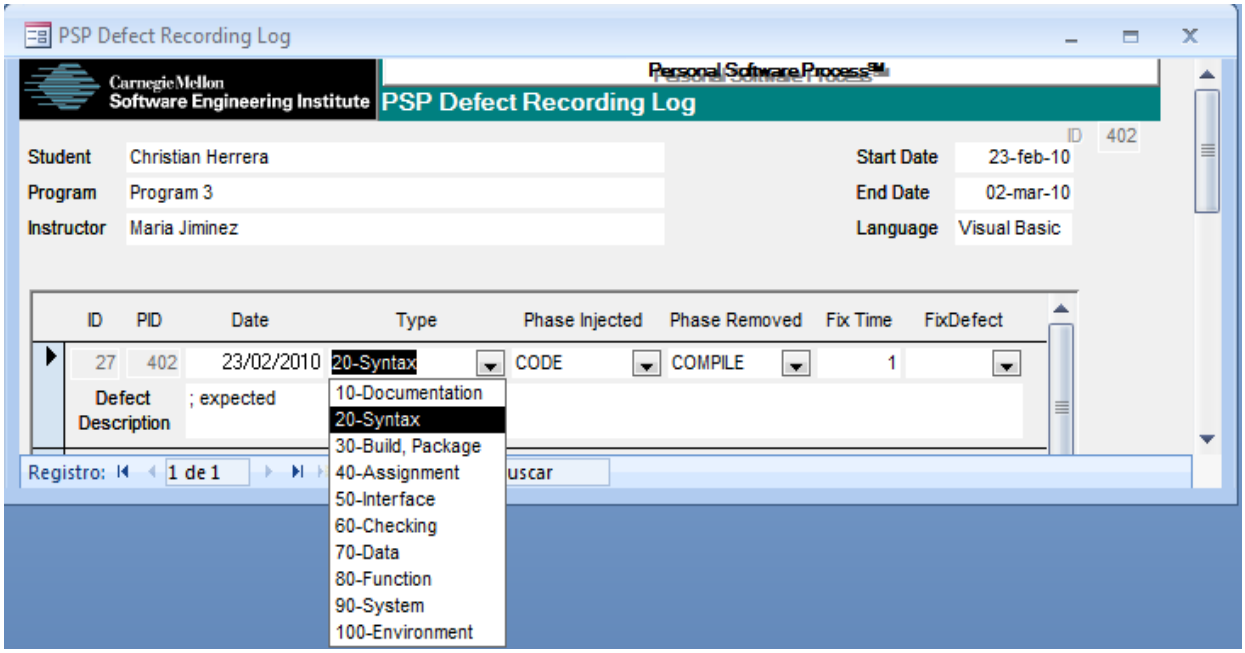

Figura 6 - Formulario para registro de defectos

El cálculo del tiempo de ***Find and fix*** (encontrar y corregir) depende de la fase**.** En el caso de las Revisiones de código y en las Revisiones de diseño, el tiempo se comienza a registrar una vez que el defecto es encontrado, por lo que solo incluye el tiempo de corregir dicho error. El tiempo de encontrarlos corresponde al tiempo en fase, menos el tiempo de la corrección; a diferencia de otras fases, donde se calcula como el tiempo de encontrar el defecto, más corregirlo.

Se registran medidas y estimaciones de **tamaño**. Usualmente, la unidad de medida para el tamaño es líneas de código lógico (LOC lógico). Hacen referencia a elementos del lenguaje, a diferencia del LOC físico que corresponde a líneas de texto. Para contabilizar LOCs lógicos se debe utilizar un **estándar de codificación** del lenguaje y uno de **conteo de elementos**. En caso de no existir se debe definir uno. Los LOCs se registran en el resumen del plan. De los LOCs y el tiempo en fase se deriva el esfuerzo y la productividad. Existe una categorización que diferencia los tipos de LOCs según el origen del código, código nuevo o agregado, base[4] modificado, eliminado, reutilizado, y total.

4 Si se parte de un código ya existente, loc base es el tamaño original.

Capítulo 3 - Elección de Herramienta y sistema gestor de base de datos

Los datos a utilizar en este trabajo provienen de aplicar el *PSP.* Estos se ingresan mediante la herramienta que da soporte al ingreso de datos *PSP Student Workbook,* descripta en el capítulo anterior. Tanto la aplicación como el almacenamiento de los datos se realizan en Microsoft Access.

Cada estudiante utiliza una instalación independiente de la herramienta con su propia base de datos. Al final del curso se tiene por cada uno de ellos una base de datos Access con todos los datos ingresados.

Al contar con un conjunto de bases de datos distribuidas, para realizar un análisis estadístico, se debe disponer de un mecanismo que permita acceder a los datos de forma integrada. Con el fin de simplificar este mecanismo, tanto para este como para trabajos futuros, se genera una integración física de los datos. Para ello se utiliza: una estructura y un sistema gestor de base de datos, y un proceso que migre los datos desde las bases individuales Access a la nueva base que almacena los datos de forma integrada. A partir de ahora se utilizará el termino “base individual” para hacer referencia a cada una de las bases individuales Microsoft Access y “base integrada” para la base a la cual se migran e integran los datos.

3.1 - Introducción

En trabajos anteriores se construyeron dos herramientas que migran datos desde las bases individuales a una nueva base de datos que los almacena de forma integrada. Una es AllPSPStudentData_v1, que está desarrollada en Java y utiliza como gestor de base de datos HSQLDB. Mediante esta, se integra una parte de la estructura de las bases individuales para realizar reportes gráficos específicos en el estudios anteriores [6] [3]. La otra está desarrollada en Genexus, “Herramienta Genexus” y a diferencia de la anterior migra todos los datos a una base de datos con motor MySQL.

Para la nueva solución, se compararon tres sistemas de gestores de base de datos: Microsoft Access, MySQL y HSQLDB. Se restringe a ellos porque son los que utilizan las herramientas.

Estructura de las bases individuales

La estructura de la base Access soporta el almacenamiento de todos los datos ingresados por los sujetos, pero no tiene definidas relaciones entre tablas. Únicamente tiene definidas tablas con sus tipos de datos. En el Anexo 1 - Estructuras de las bases Microsoft Access y MySQL, en la columna “TABLA EN ACCESS” se pueden visualizar los nombres de las tablas.

3.2 - Herramienta y estructura de la base

3.2.1 - Herramienta AllPSPStudentData_v1

En primera instancia, se investiga la herramienta desarrollada en Java, AllPSPStudentData_v1. Se cuenta con: código fuente de la herramienta, base de datos HSQLDB con los datos de los estudiantes ya cargados y algunos scripts que generan y cargan tablas.

Sus principales funcionalidades son:

- Crear la estructura de la base de datos integrada y cargar las tablas “codigueras”.
- Migrar parcialmente las bases individuales. Dado un directorio que contiene los archivos de las bases individuales, migra parte de su contenido a la base integrada.
- Generar reportes gráficos utilizando los datos cargados.

Estructura de la base de datos integrada (HSQLDB)

La estructura de la base integrada, se basa en la de las bases individuales, descartando algunas tablas que no eran necesarias en los trabajos a realizar, y agregando otras para facilitar la ejecución de consultas.

La estructura está conformada por: tablas que son una proyección de las tablas originales -que se encuentran en las bases individuales-, tablas descriptivas "codigueras" y otras tablas auxiliares. Estas últimas

corresponden a sumarizaciones o resúmenes de datos.

Dentro de las "codigueras" se encuentran:

- *Defectype:* Representa los tipos de defectos que pueden existir en el código. Estos se seleccionan al registrar un defecto, ver Figura 6. Sus campos son: identificador, nombre y detalle.
- *Phases:* Representa las fases del PSP, los campos son: identificador, nombre corto y nombre de la fase.

A continuación se detallan las tablas que se cargan con los datos de las bases Access:

- *Users:* Representa los sujetos (estudiantes) que utilizan *PSP Student Workbook*. A diferencia de la base individual, agrega un identificador para cada sujeto y quita el campo que la asocia con la tabla *UserProfiles*. El resto de la información -nombre, iniciales, instructor del curso y organización a la que pertenece el estudiante- se conserva.
- *UserProfiles:* Representa el perfil del estudiante. De la base individual, únicamente conserva el identificador de la tabla, el lenguaje en el que realiza los programas y agrega el identificador del estudiante para asociarlo con *Users*.
- *Programsize:* Representa el tamaño de los programas que realiza cada estudiante durante el curso. Contiene un identificador para el registro, identificador para asociarla con *Users*, identificador del programa y el tamaño en LOCs de código agregado o modificado. Deja por fuera los datos históricos, planificados y actuales de los programas respecto a otras categorías de LOCs. Son los datos registrados mediante el resumen del plan (Figura 4).
- *Logdetail:* Representa los defectos encontrados, Figura 6. Contiene: identificador del usuario, identificador del registro, programa, fase en que se inyecta y remueve el defecto, tipo de defecto, tiempo de *find and fix*, descripción y fecha-hora en la cual se encuentra. No sufre modificaciones respecto al de la base individual.

El resto de las tablas, *Stuprodefphainjrem* y *Totaldefects* corresponden a tablas auxiliares que contienen un resumen de las anteriores. La primera unifica los datos ingresados, mientras que la última sumariza los datos registrados en la primera, agrupando por tipo de defecto.

La herramienta presentada realiza una integración reducida de las tablas individuales. Una de las tablas que no migra es *Logtdetail*, la cual contiene el tiempo dedicado a cada fase en cada programa. Dicha tabla es necesaria para el análisis a realizar. Por lo que, para reutilizar al AllPSPStudentData_v1 se deben hacer modificaciones tanto en la estructura de la base de datos como en el código.

3.2.2 - “Herramienta Genexus”

La segunda opción tenida en cuenta, es una herramienta desarrollada en Genexus. Esta, dado un directorio que contiene las bases individuales, se encarga de migrarlas de forma completa a una nueva base de datos. En este caso, el motor utilizado es MySQL. A diferencia de AllPSPStudentData_v1 no se cuenta con una versión portable de la herramienta, sino que se dispone de una máquina virtual con la instalación de la herramienta y de la base de datos.

Al ejecutarla se detecta que algunas de las bases no se migran. Al no disponer de una licencia Genexus no es posible investigar el problema.

Estructura de la base de datos integrada (MySQL)

La estructura de la base de datos integrada MySQL, está definida con las mismas tablas que la base de datos individual, a excepción de algunas diferencias necesarias para la correcta integración de los datos. Tiene tablas descriptivas y tablas de datos cuyas estructuras son iguales a las bases individuales. La diferencia con la base Access es que agrega un campo de identificador único en cada tabla para referenciar a un sujeto que ingresa datos en el *PSP Student Workbook*.

3.2.3 - Conclusión

Se opta por reutilizar la herramienta AllPSPStudentData_v1, pero combinándola con la estructura de la base MySQL que utiliza la “Herramienta Genexus”. La “Herramienta Genexus” se descarta ya que usarla implica dejar bases de datos individuales sin cargar. No fue posible investigar el motivo al no disponer de la licencia necesaria. AllPSPStudentData_v1 no requiere de licencias y Java es un lenguaje ampliamente utilizado, lo cual se alinea con la idea de reutilización. Otra opción es generar una herramienta desde cero, sin embargo como ya se tenía implementada la interfaz de usuario, la obtención de las bases Access y la generación de algunos reportes gráficos, la opción es descartada.

3.3 - Elección del sistema gestor de base de datos

Existen muchos sistemas gestores de base de datos, sin embargo la elección se restringe únicamente a tres de ellos: Access, HSQLDB y MySQL. Estos coinciden con los que utilizan las herramientas investigadas.

Para la decisión se consideran los siguientes factores:

- Soporte de bases de datos relacionales.
- Que sea libre, no se necesite una licencia paga para su uso.
- Estándar, para facilitar su reutilización en trabajos futuros es deseable que sea ampliamente difundido.
- Performance, los tiempos de respuesta a las consultas realizadas sobre los datos sean aceptables.
- Experiencia de usuario.

En primer lugar se analiza **Access**, ya que al estar las bases individuales implementadas en él, supone un impacto menor al cambiar la estructura. Se descarta por los siguientes motivos:

- Es una base de datos comercial. Este es el principal motivo que condiciona su elección ya que es una limitante importante a la hora de reutilizarla.
- No es multiplataforma. Si bien se puede acceder en modo lectura en sistemas operativos que no sean de Microsoft es compleja su manipulación.
- No es estable. Es sensible a la manipulación de datos por medio de herramientas externas, generando errores de acceso y configuración de los archivos.

HSQLDB y MySQL comparten las siguientes características deseables en el gestor buscado:

- No se requieren licencias para su utilización, lo cual es esencial para no limitar su uso.
- Estándares, HSQLDB no es tan difundida como MySQL pero ambas tienen como base SQL estándar.
- Son multiplataforma.
- Son gestores para bases de datos relacionales. Esto es importante porque las bases individuales también lo son y ello simplifica la migración.

HSQLDB se descarta por su performance en el entorno de ejecución. Al ejecutar consultas que recuperan muchos registros, se presentan importantes demoras en las respuestas. Y en algunos casos, tanto el programa para su gestión como la base de datos dejan de ejecutarse.

Se evalúan las siguientes ventajas:

- No presenta requisitos especiales sobre el tipo de sistema operativo. Dado que trabaja sobre Java, el único requisito es tener instalada la *Java Virtual Machine* (JVM).
- Es un desarrollo Java que se puede integrar fácilmente con herramientas generadas en Java y con un IDE ampliamente difundido como lo es Eclipse.

Al investigar la degradación de la performance se concluye que sucede porque la base de datos y el entorno incorporado para su gestión están desarrollados en Java, y la memoria disponible para la JVM no es suficiente.

MySQL es el gestor elegido por las ventajas mencionadas, agregando los siguientes motivos:

- Performance: en comparación con HSQLDB el retorno de las mismas consultas era prácticamente inmediato.
- Es un gestor ampliamente difundido del cual hay disponible mucha información.
- Se cuenta con experiencia en la utilización del mismo tanto académica, como comercialmente.

Capítulo 4 - Diseño de la solución para la integración de los datos

En este capítulo se presenta la solución implementada para la integración de los datos. La solución integra en una única base de datos (MySQL) todas las bases individuales (Access). Está conformada por una estructura y un gestor de base de datos, una herramienta que migra los datos y un proceso de integración.

4.1 - Proceso de integración

El proceso de integración completo se compone de tres etapas. En la primera, mediante la ejecución de *scripts,* se crea la estructura de la base de datos. Luego de generada esa estructura, se cargan datos en las tablas descriptivas "codigueras"; esta acción se realiza ejecutando una serie de *scripts*. Finalmente, se utiliza la herramienta implementada en Java, AllPSPStudentData_v2, cuya función principal migrar los datos de las bases individuales a la nueva base de datos integrada.

En la Figura 7 se presenta la arquitectura de la solución. El proceso es aplicado de forma completa cuando no se tiene creada la estructura de la base de datos. En caso de que se tenga creada, solo se deben migrar los datos utilizando AllPSPStudentData_v2. Los *scripts* de datos (que cargan las tablas descriptivas) pueden ser ejecutados utilizando un gestor de base de datos o por medio de la propia herramienta.

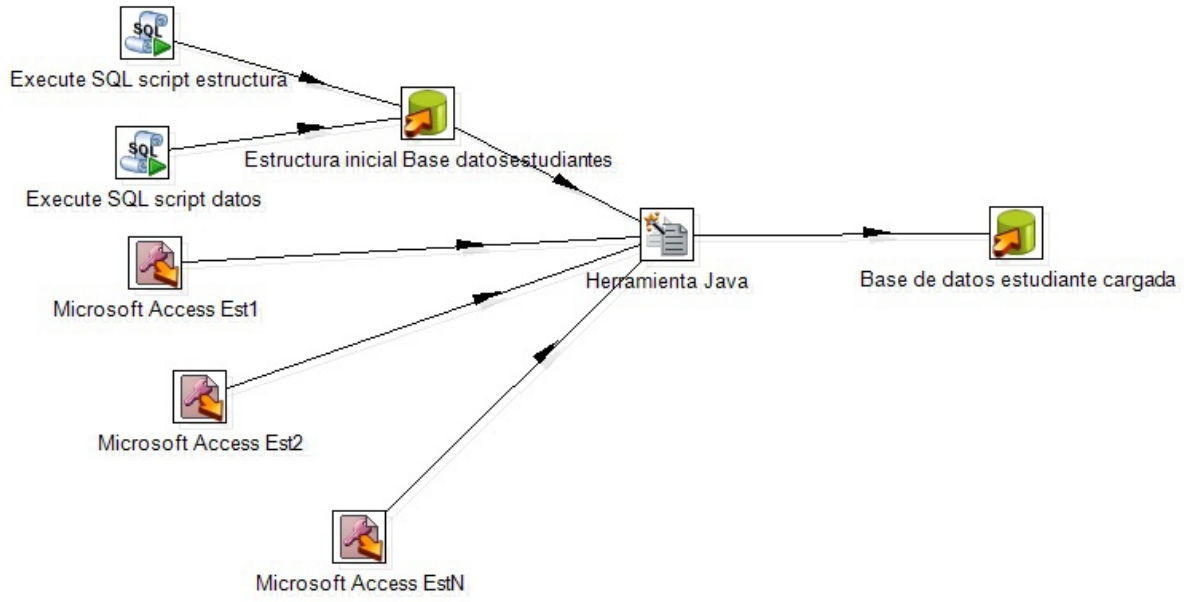

Figura 7 - Arquitectura de la solución

4.2 - Base de datos MySQL

Para crear la estructura de la nueva base de datos integrada, se reutiliza la estructura de la base de datos MySQL de la “herramienta Genexus”. Dado que esta última no tiene definida ninguna regla de integridad referencial, como ser las claves foráneas (FK), se realiza una investigación para poder deducirlas y así lograr un mejor diseño en la nueva estructura. Para determinar los cambios a realizar, se observan los datos, la nomenclatura y las columnas de las tablas. Por otro lado se revisan las notas de la estructura MySQL de la “herramienta Genexus”. Como resultado se definen las reglas de integridad referencial faltantes. En el Anexo 1 - Estructuras de las bases Microsoft Access y MySQL se puede observar la correspondencia entre las tablas de las bases individuales y la base integrada de AllPSPStudentData_v2.

Por cuestiones de compatibilidad entre las claves, se modifican algunos tipos de datos: algunos campos para que acepten el valor “*null”* y otros que son “*default null”* se cambian por “*not null”* (porque no es posible

definir FK sobre campos que sean por defecto vacíos). El detalle de las modificaciones realizadas se puede visualizar en Anexo 2 – Detalle de las tablas MySQL de AllPSPStudentData_v2 y los cambios realizados en la estructura de la "herramienta Genexus.

4.2.1 - Estructura

Se presenta en la Figura 8 el esquema de base de datos utilizado para integrar los datos provenientes de las bases de datos Access completadas durante los cursos. Por legibilidad se omiten las claves primarias y foráneas, y tipo de datos. Se pueden observar dos tablas: totaldefects y stuprodefphainjrem, estas no corresponden a la estructura necesaria para la migración, se agregan con el fin de simplificar el diseño de algunas consultas sobre los datos cargados.

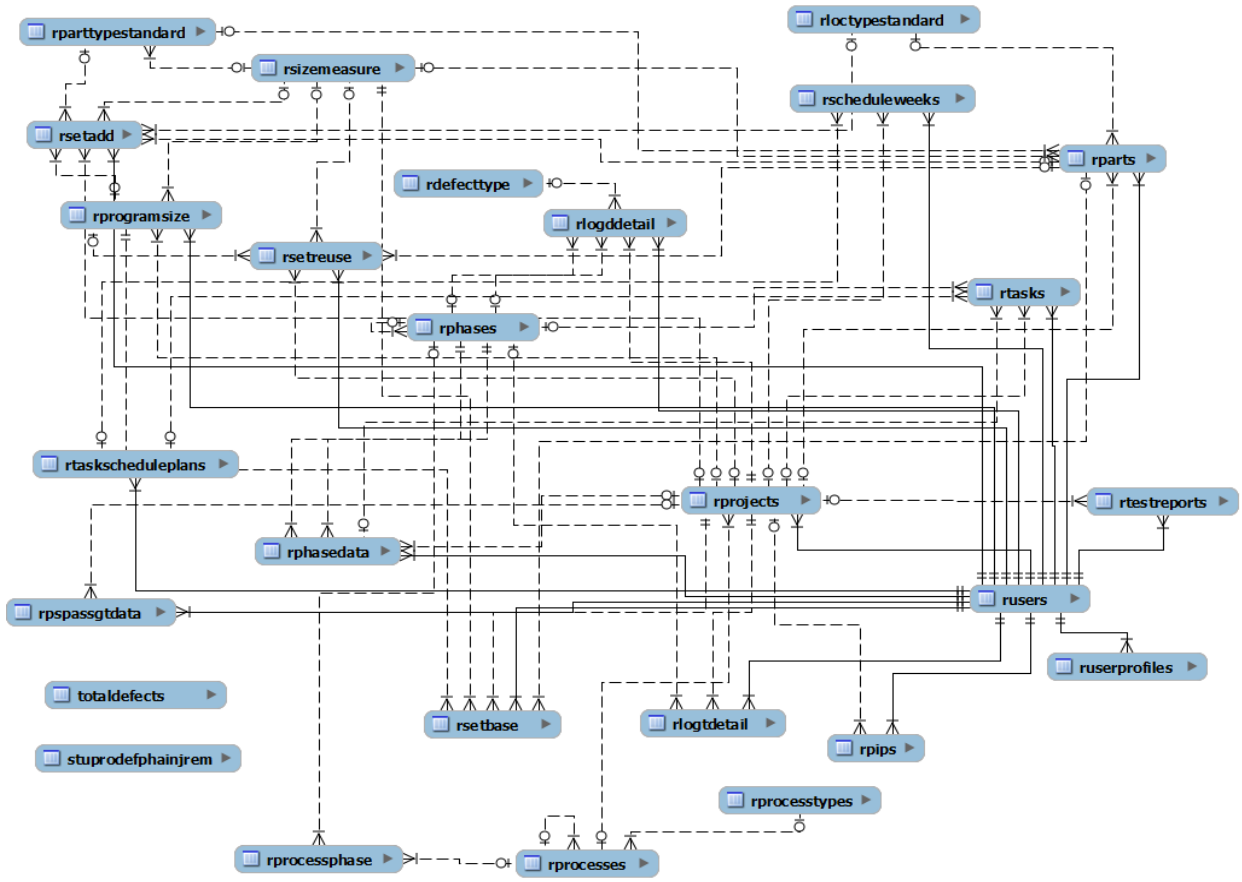

Figura 8 - Esquema de la base de datos MySQL de la herramienta

El alcance del proyecto no abarca la migración completa de los datos, se limita a la carga de los necesarios para obtener determinados resultados estadísticos. La porción del esquema que se carga se puede visualizar en la Figura 9. Se pueden ver las tablas, claves primarias pk (identificadas con un símbolo de llave) y claves foraneas fk (identificadas en los indices como fk_nombretabla...).

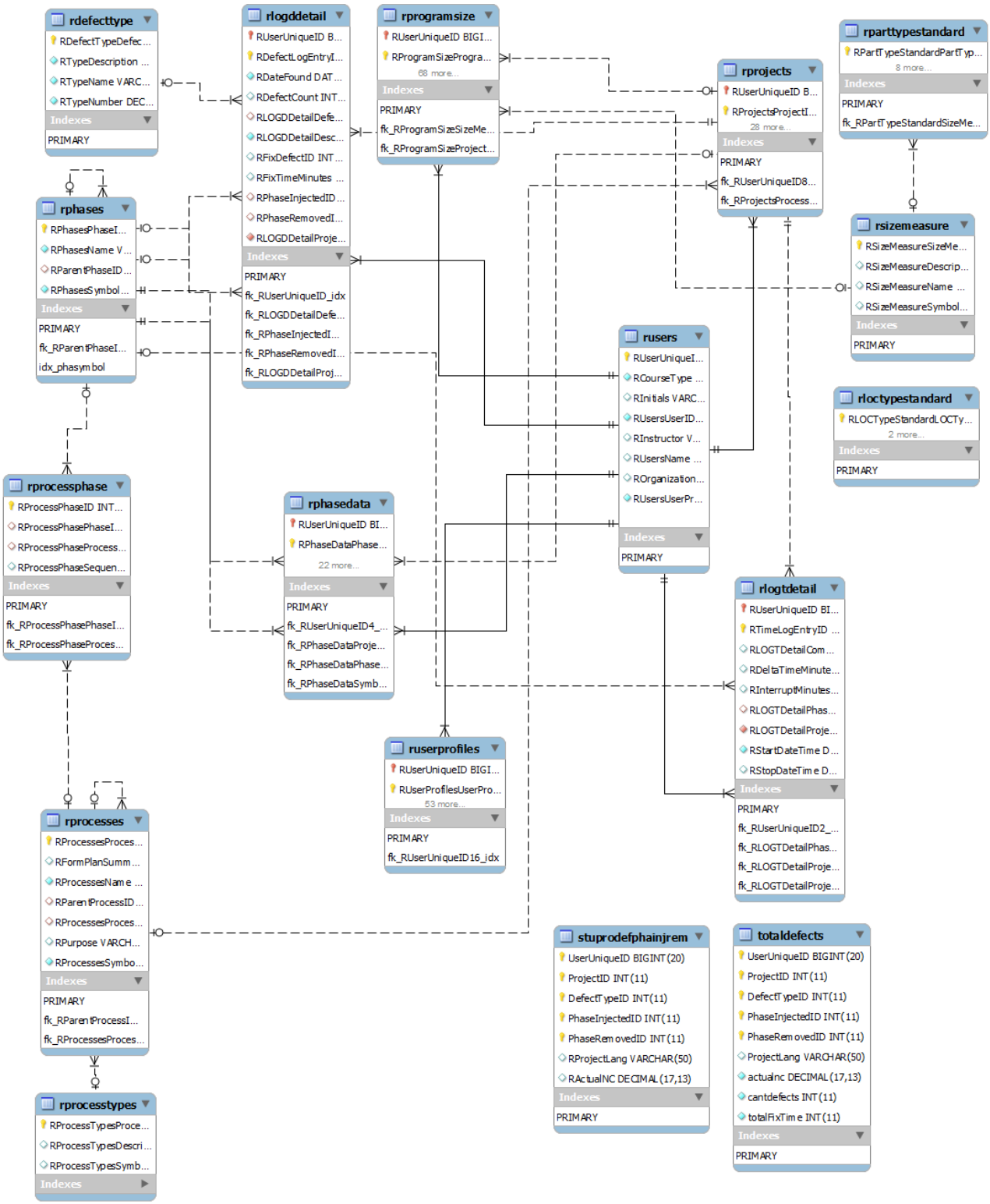

Figura 9 - Vista de la porción cargada de la base de datos MySQL

Las **tablas** se clasifican en dos grupos según su función: tabla de datos (migrados) y tablas auxiliares. El primero corresponde a tablas que contienen los datos tal cual se registraron en las bases individuales. El otro, a tablas generadas realizando agrupaciones: se crean para simplificar el diseño de las consultas sobre los datos.

Tablas de datos

Hay dos tipos de tablas: las descriptivas o "codigueras", que tienen la misma estructura y comparten los mismos datos para todos los sujetos, y las de registros, que únicamente comparten la misma estructura ya que los datos ingresados dependen de los registros realizados. Como se puede observar en las Figura 8 y Figura 9, en general, las tablas de registros tienen claves foráneas para relacionarse con las "codigueras". Los Cuadro 1 y Cuadro 2 muestran una breve descripción de cada tabla.

Nombre tabla	Descripción	Datos cargados
rdefecttype	Representa los tipos de defectos.	Tipos de defectos: *Documentation,Syntax,Build,Package,Assignment,Interface,Checking,Data,Function System, Environment, Others*
rphases	Representa las fases del PSP y otras actividades.	PLAN,DLD,CODE,COMPILE,UT,PM,DLDR, CR,AD *(After Development), Entry Criteria (PSP workbook, PIPs for process needs & Design and code review checklists), Analysis of data, Planning, Create Report, Postmortem, Exit Criteria*
rprocesses	Representa las versiones del proceso.	PSP0,PSP0.1,PSP1,PSP1.1,PSP2,PSP2.1 ,*Report*
rprocessphase	Especifica las fases consideradas para cada versión del proceso, corresponde a una relación entre (rphases y rprocesses).	Se carga con el identificador de la fase, el identificador de la versión del proceso y un secuencial indicando el orden de la fase en el proceso
rloctypestandard	Representa los tipos de LOCs	Tipos LOCs: *B (Base), M (Modified), D (Deleted), A (Added), R (Reused), N (New and Changed), T (Total), NR (New Reused), BA (Base Additions), NO (New Objects), E (Estimated Proxy Size)*
rsizemeasure	Representa los tipos de medidas de tamaño.	LOC
rparttypestandard	Representa las distintas partes (objetos) que componen un programa y el tipo de medida con la que se mide su tamaño.	Las partes son medidas todas con LOCs: *Calc (Calculation object), Data(Data object), IO (IO object), Logic (Logic object) , Set-up (Set-up object) , Text(Text object)*
rprocesstypes	Corresponde a tipos de procesos que engloban las versiones .	*Personal Software Process I* *Personal Software Process II*

Cuadro 1 - Tablas descriptivas o "codigueras"

Nombre tabla	Descripción	Datos cargados
rusers	Representa un usuario del *PSP Student Workbook.*	Contiene datos básicos: identificador de usuario, nombre, nombre del instructor, organización a la que pertenece el usuario etc.
rusersprofile	Contiene información del perfil del usuario	Experiencia en determinados roles en el desarrollo de software ,en lenguajes de programación, etc.
rprojects	Representa los ejercicios (programas) que realizan los usuarios.	Tiene la fecha de inicio y fin, si fue completado o no, que versión del proceso se aplico, etc
rphasedata	Asocia datos de cada fase realizada en un programa con la fase y el programa.	Contiene información actual, histórica y planificada acerca de tiempos y defectos en cada una de las fases de cada ejercicio
rlogtdetil	Contiene los tiempos insumidos en cada fase.	Se registra los tiempo de inicio y fin, interrupciones y el intervalo real de tiempo y los asocia con la fase y el proyecto.
rlogddetail	Representa los defectos encontrados. Cada registro se relaciona con el programa y la fase en que se registra.	Contiene: identificador del usuario, identificador del registro, programa, fase en que se inyecta y remueve el defecto, tipo de defecto, tiempo de *find and fix*, descripción y fecha-hora en la cual se encuentra.
rprogramsize	Representa medidas de los programas que realiza cada usuario.	Se registra información actual, histórica y planificada del tamaño de cada uno de los programas. En dicha tabla se registran las cantidades considerando la clasificación de los tipos de LOCs.

Cuadro 2 - Descripción de las tablas cargadas de la base de datos

Tablas auxiliares

Al diseño de la estructura que almacena los datos de las bases individuales, se le adiciona un par de tablas que contienen agrupaciones y agregaciones de algunos datos.

stuprodefphainjrem: Contiene las combinaciones de usuario, programa, tipo de defecto, fase inyección y remoción de defectos, lenguaje utilizado en el proyecto. A cada tupla le asocia el tamaño medido en LOCs *New and Changed* del proyecto (programa).

totalDefects: La agrupación la realiza mediante la misma combinación que en la tabla stuprodefphainjrem, y agrega la sumarización de la cantidad de defectos y el tiempo de *find and fix* a cada tupla.

El Anexo 2 – Detalle de las tablas MySQL de AllPSPStudentData_v2 y los cambios realizados en la estructura de la “herramienta Genexus contiene un detalle de las **claves primarias** de la estructura generada.

4.2.2 - Generación de la estructura y carga de codigueras

La estructura de la base es generada mediante la ejecución de *scripts*. Están diseñados para que la estructura sea generada sobre el gestor de base de datos MySQL. Previo a generar la estructura, es necesario tener instalado un servidor de base de datos MySQL y alguna herramienta para facilitar su gestión. En este caso se utiliza MySQL Workbench.

En primera instancia se crea el esquema "datosestudiantes", con todas las tablas y sus relaciones. Luego se cargan las tablas descriptivas o “codigueras”: una vez completada esta etapa, la base de datos está pronta para que se puedan migrar los datos. Los *scripts* que cargan datos de las “codigueras” pueden ser

ejecutados utilizando AllPSPStudenData_v2 o por medio de una herramienta de gestión de base de datos.

En caso de que se quiera cambiar el nombre al esquema en la base de datos, solo basta con modificar el script que genera la estructura y en AllPSPStudenData_v2, los parámetros del acceso a la base de datos.

El *script* que crea la estructura de la base de datos es *datosestudiantesschema.sql*. Crea tablas, los tipos de datos, claves primarias y foráneas, y un secuencial que es utilizado para asignarle un identificador único a cada uno de los sujetos (usuarios).

La carga de las tablas "codigueras", es donde se carga la información que se encuentra por defecto en las bases de datos de cada uno de los sujetos. Se realiza con los *scripts* indicados a continuación.

- *RDefectType.sql:* Carga la tabla rdefecttype que contiene los tipos de defectos sobre los cuales los estudiantes realizaron la clasificación.
- *RLOCTypeStandard.sql:* Inserta en la tabla rloctypestandard los tipos de LOCs.
- *RPartTypeStandard.sql:* Carga la tabla rparttypestandard que contiene la clasificación de las partes (objetos) que componen el programa.
- *RPhases.sql:* Carga la tabla rphses que corresponde a las fases del PSP.
- *RProcesses.sql:* Inserta los datos en la tabla rprocesses, que contiene las diferentes versiones del PSP.
- *RProcessPhase.sql:* Carga la tabla rprocessphase, que almacena la relación entre las versiones del PSP y las fases que se aplican en cada versión.
- *RProcessTypes.sql:* Carga la tabla rprocesstypes que contiene los tipos de procesos definidos.
- *RSizeMeasure.sql:* Inserta los datos en la tabla rsizemeasure, que indica los tipos de medida que se utilizan para medir los programas.

Cargan únicamente las tablas que no deben tener variaciones entre las bases de datos de cada estudiante. Sin embargo al momento de cargar algunos datos de los estudiantes, se presentan algunas inconsistencias, que se mostrarán más adelante en este informe. Una vez ejecutados todos los *scripts,* el esquema queda pronto para migrar los datos de las bases individuales.

4.3 - Herramienta AllPSPStudentData_v2 utilizada para la carga de los datos

El objetivo de esta última etapa es migrar los datos ingresados por los sujetos. AllPSPStudentData_v2 migra parte de los datos almacenados en las bases individuales a la nueva base de datos MySQL. En un principio, se opta por reutilizar AllPSPStudentData_v1 para potenciar algunas de sus funcionalidades extras, como la generación de gráficos. Esta última función se termina eliminando, ya que todo lo relacionado con análisis estadístico se realizará utilizando herramientas con mayor potencial ya desarrolladas. Por ejemplo, el software estadístico R.

4.3.1 - Especificación

La principal función de la herramienta es migrar los archivos de las bases individuales a la base integrada. Por medio de la interfaz, se selecciona un directorio que contiene los archivos. Se realiza una recorrida en profundidad (*DFS, Depth First Search*) del directorio, donde se seleccionan uno a uno los archivos Access y se migra una porción de los datos a la nueva base de datos. En la Figura 10 se muestra la interfaz principal de la herramienta.

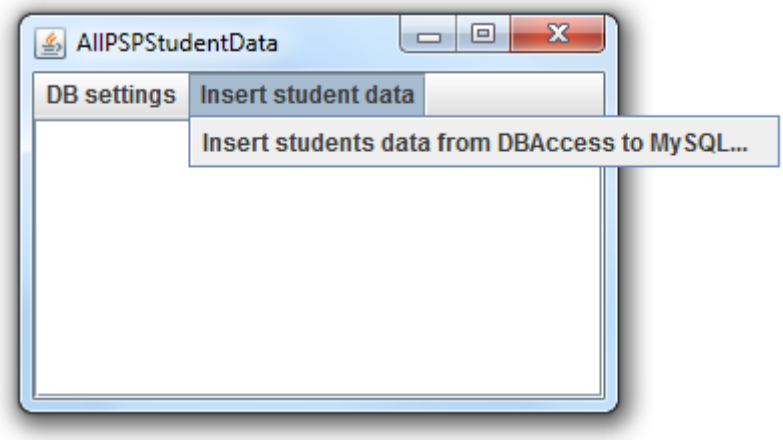

Figura 10 - Herramienta AllPSPStudentData_v2

Una vez que selecciona una base de datos individual (Access), se cargan algunas de sus tablas. En el Cuadro 3 se muestran las tablas que se cargan durante la migración (integración) de datos y las dependencias entre ellas y las "codigueras". De las dependencias surge el orden en que se migran las tablas.

ORDEN DE CARGA	METODO	RELACION CON TABLA					
rusers	Java						
rusersprofile	Java	rusers					
rprojects	Java	rusers				rprocesses	
rphasedata	Java	rusers	rprojects				
rlogtdetail	Java	rusers	rprojects	rphases			
rlogddetail	Java	rusers	rprojects	rphases			rdefecttype
rprogramsize	Java	rusers	rprojects		rsizemeasure		

Cuadro 3 - Dependencia entre tablas y orden de carga

Otra de las funcionalidades que tiene la herramienta es ejecutar los *scripts* que contienen los datos de las "codigueras" en la base de datos. Al igual que en el caso anterior se debe seleccionar el directorio que contiene los scripts.

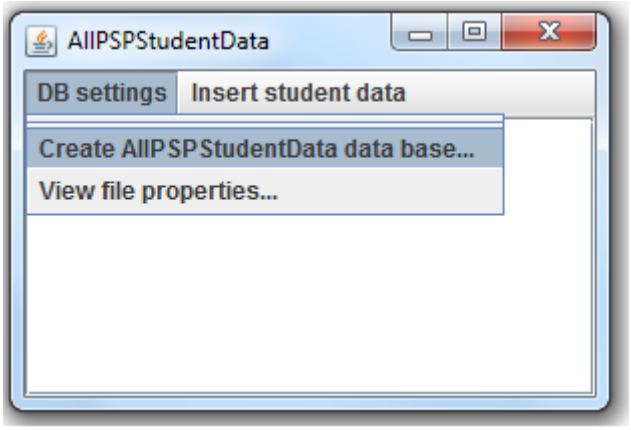

Figura 11 - Herramienta AllPSPStudentData_v2 DB *Settings*

Adicionalmente, otra funcionalidad de la herramienta es permitir modificar los paramentos de configuración para el acceso a las bases de datos individual (BD *Source*) e integrada (BD *Dest*). En la **Figura 12** se visualiza la pantalla mediante la cual se pueden modificar o visualizar. Estos datos se guardan en el archivo de configuración **file.properties.**

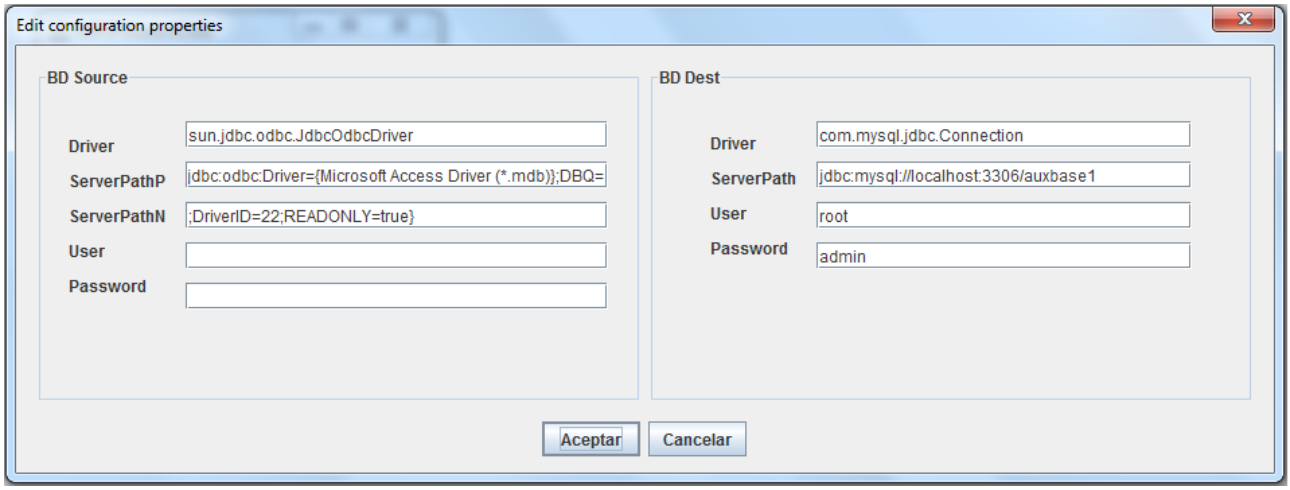

Figura 12 - Pantalla para modificar los parámetros de acceso a la base de datos

file.properties

Es un archivo en el cual se almacenan los parámetros configurables de un programa, en este caso los de las bases de datos a las cuales accede. Este debe estar ubicado en el directorio raíz de la herramienta.

Identificador del estudiante

AllPSPStudentData_v1, para proporcionarle un identificador al estudiante, utiliza una función que genera un número randómico (randomUUID). En AllPSPStudentData_v2, el identificador es generado por un secuencial retornado por la base de datos al momento de insertar un nuevo registro en la tabla rusers.

Log general

La herramienta genera un archivo de log donde se detallan los errores generales que ocurren en tiempo de ejecución. El archivo se crea en el directorio raíz de la ejecución, su nombre es log_yyyy-MM-dd_HH_mm.txt; donde yyyy corresponde al año, MM al mes y dd al día que se crea,y HH la hora y, mm los los minutos.

Logs de migración de datos

AllPSPStudentData_v2 despliega un log en la pantalla y genera un archivo de *log*. El archivo se crea en el directorio raíz del proyecto, su nombre es logInsertAllPspStudent_yyyy-MM-dd_HH_mm.txt; donde yyyy corresponde al año, MM al mes y dd al día que se crea,y HH la hora y, mm los los minutos.

En el mismo se registran:

- Excepciones que ocurran durante la ejecución ya sea de tipo entrada-salida, sql, etc.
- Un resumen del resultado de la migración de cada una de las bases. Registra información útil, cuando se produce algún error al momento de insertar los datos. Se puede ver la especificación de su formato en el Anexo 4 - Descripción y formato del archivo de log.

4.3.2 - Arquitectura e Implementación

El diseño corresponde a una arquitectura en capas, presentación, lógica y datos. Cada capa se encuentra distribuida en varios paquetes que contienen las clases implementadas. A continuación se especifican las clases involucradas en la implementación. Se utiliza *class* para indicar las clases no modificadas, *newclass* para las clases nuevas y *chgclass* para las modificadas.

4.3.2.1 - Capa de presentación

package *AllPSPStudenData.coreUI:*

- *chgclass* AllPSPStudenData: Constructora de la interfaz principal que da el acceso a las distintas funcionalidades de la herramienta. Se modifica para que le brinde al usuario las opciones de: migrar las bases de datos, cargar las “codigueras” (en caso de que no lo estén) y acceder a la pantalla que muestra los parámetros de configuración.
- *newclass* PropertiesShowPanel: Construye el panel que muestra y permite modificar los parámetros de configuración.

4.3.2.2 - Capa lógica

package AllPSPStudentData.Data.core:

- *newclass* AllPSPStudentDataImpl: Proporciona una interfaz de acceso a los diferentes métodos de la herramienta. Funciona como un patrón de diseño *Facade* para las clases que implementan los métodos. Es invocado desde AllPSPStudenData. También es la que se encarga de instanciar al file.properties para obtener la configuración.

package AllPSPStudentData.DataBaseCreation:

- *chgclass* InsertAllStudenDataUnderDirectory: Es la encargada de recorrer en profundidad el directorio (seleccionado por el usuario), el cual contiene los archivos .mdb a cargar. A medida que obtiene cada uno invoca a InsertStudentData, que es la clase encargada de realizar la carga de cada base individual a la base integrada. A dicha clase le pasa como parámetro la conexión a la base individual y a la base integrada y el *buffer* para generar el *log*.

4.3.2.3 - Capa de datos

package dataBaseBasics:

- *class* DataBaseBasics: Es la clase abstracta que es implementada por DataBaseBasicsMSAccess y DataBaseBasicsMySQL.
- *class* DataBaseBasicsMSAccess: Se encarga de setear el *driver* para la conexión a la base de datos Access y establece la conexión a la misma.
- *newclass* DataBaseBasicsMySQL: Se encarga de setear el *driver* para la conexión a la base de datos MySQL y crea la conexión hacia la misma.
- *newclass* DataBaseAccessProperties: Es la que brinda el acceso al **file.properties,** permitiendo a las otras clases su obtención, carga y modificación.

package AllPSPStudentData.DataBaseCreation:

- *chgclass* CreateAllPSPStudentDataBase: Se encarga de ejecutar los *scripts* de carga los datos de las “codigueras”. Se le pasa como parámetros la configuración de la base de datos integrada. Es invocada por AllPSPStudentDataImpl.
- *newclass* InsertStudentData: Recibe como parámetro la conexión a las bases de datos Access e integrada (MySQL) y se encarga de la migración de los datos de una a otra. Toma del archivo .mdb los datos de las tablas a migrar y los inserta en la nueva estructura. A su vez escribe en el *log* el detalle del resultado de la carga de cada uno de los registros.

- La clase está compuesta por siete procedimientos, uno por cada una de las tablas a cargar, la nomenclatura utilizada para los nombres de los mismos hace referencia a la tabla que se carga. Se ejecutan de forma secuencial, siendo la primera tabla que se carga la que contiene información básica de los estudiantes (rusers). Se realiza de esta forma para obtener al momento de insertar el identificador del usuario, el cual es utilizado para relacionar la tabla rusers con el resto de las tablas.

Durante el proceso de integración de datos se descubrieron problemas de calidad en los mismos, estos se detallan en el Anexo 5 - Calidad de datos y problemas durante la integración.

Capítulo 5 - Análisis

Desde su primera publicación en 1995 el *Personal Software Process (PSP)* ha sido objeto de análisis. El fin detrás de los estudios es responder qué tan beneficiosa resulta su aplicación para generar software de calidad y el impacto en su costo.

El objetivo de este capítulo es presentar el análisis estadístico realizado y, con él, validar que la migración utilizando la herramienta creada, *AllPSPStudentData_v2,* es de utilidad. La herramienta utilizada para el análisis es el software estadístico R.

El análisis complementa dos trabajos anteriores, uno de ellos es *Quality is Free, Personal Reviews Improve Software Quality at No Cost*, en el cual se estudia la eficiencia de la fases de Revisión de diseño y codificación para remover defectos y su costo [3]. El otro, *Demostrating the Impact of the PSP on Software Quality and Effort: Eliminating the Programming Learning Effect* muestra que la mejora de la calidad del software está dada por seguir un proceso disciplinado como lo es el *PSP* y no por programación repetitiva [4]. En cada uno, se estudian datos recabados en diferentes ámbitos de aplicación. En el primero, los datos corresponden a cursos dictados del *PSP* y en el segundo, a experimentos controlados. La diferencia entre ambos radica en que en los experimentos no se aplicaron los métodos del *PSP,* pero sí registraron datos y realizaron los mismos ejercicios y en el mismo orden.

Para este segundo objetivo, el análisis estadístico de los datos, se plantean dos subobjetivos. El primer subobjetivo, consiste en evaluar qué tan efectivas son las fases de Revisión de diseño y codificación para remover defectos, y cuál es su costo. El segundo subobjetivo, consiste en analizar si la mejora en la performance del estudiante en los programas desarrollados, aplicando el *PSP,* es producto de la aplicación de un proceso disciplinado (como lo es el *PSP),* o por la creación de varios programas en el mismo dominio de aplicación de forma consecutiva (programación repetitiva).

5.1 - Efectividad de las fases de Revisión para remover defectos

En el primer subobjetivo del análisis estadístico, la efectividad se estudia observando el comportamiento de la inyección y remoción de defectos en las distintas fases del proceso. Se compara la cantidad de defectos encontrados y removidos en las fases de Revisión respecto al resto de las fases. La remoción de defectos tiene un costo asociado. En este caso se calcula, para cada una de las fases, el costo promedio de remover un defecto.

5.1.1 - Conjunto de datos

Los datos utilizados corresponden a estudiantes que realizaron cursos del *PSP* durante el periodo comprendido entre octubre de 2005 y mayo de 2010. Los cursos fueron dictados por el *Software Engineering Institute (SEI)* de la universidad *Carnegie Mellon (CMU)* o por *partners del SEI.* Participaron un total de 130 estudiantes. El curso se compone de ocho ejercicios (programas) los cuales deben ser realizados por el estudiante de forma individual a lo largo de este. En el curso se trabajan de forma gradual todas las versiones del proceso.

Los participantes no son estudiantes entrenados en la aplicación del proceso, sino que lo están aprendiendo. Por tal motivo, es factible que comentan “errores” al tomar medidas y/o ingresar los datos. Por lo cual aquellos estudiantes cuyos registros generan incertidumbre en cuanto a su correctitud se excluiran del análisis.

Cada estudiante que participa en un curso, decide el lenguaje de programación a utilizar a lo largo del mismo. Esto es para que el factor aprendizaje no afecte las métricas. La diversidad de lenguajes utilizados, se transforma en un factor importante a controlar al momento de realizar comparaciones. La sintaxis del lenguaje hace que varíen de forma considerable las métricas registradas. Por ejemplo, no es lo mismo diseñar y codificar una misma solución en Java que en Sql. Como no es de interés tener en cuenta las variaciones que ello pueda generar, se opta por controlar dicho factor. Se seleccionan los datos de estudiantes que utili-

zaron cuatro lenguajes de similares características y que representan más del 72% de las muestras. Los lenguajes seleccionados son: Java, C#, C++ y C. Esta selección restringe el estudio a 94 estudiantes.

Los ocho ejercicios corresponden a programas que deben producir, durante los cuales registran las medidas solicitadas en las guías del proceso. Dado que las métricas necesarias para el análisis están asociadas a las fases de Revisión de código y diseño, se consideran los ejercicios en los cuales se haya aplicado el proceso en forma completa: *PSP2.0* (a partir de la versión 2.0 es que se agregan las fases de Revisión). Dicha versión se aplica en los últimos tres ejercicios del curso. Por lo tanto, los datos considerados corresponden a los registrados en los últimos tres programas de cada estudiante: 6, 7 y 8.

5.1.2 - Comportamiento de la inyección y remoción de defectos

Se busca determinar qué tan efectivas son las fases de Revisión de diseño y codificación para remover defectos y cuál es el costo asociado.

Para estudiar el comportamiento en la inyección de defectos, se calcula el promedio del porcentaje de los defectos inyectados en cada una de las fases por cada estudiante en los últimos tres programas. El cálculo se realiza considerando las fases de Diseño, Revisión de diseño, Codificación, Revisión de código, Compilación y Pruebas unitarias. Se tuvieron en cuenta 92 estudiantes en vez de los 94 seleccionados, ya que dos de ellos fueron descartados por no haber registrado defectos en ninguno de los tres programas.

El promedio más alto de inyección se da en las fases de Diseño y Codificación, con 46,4% y 52.4%, respectivamente. Las fases restantes comprenden un 1,2%. Esto muestra que las fases en las que más se inyecta son aquellas en las que se genera el producto, ya que la tarea principal en las restantes es revisar y corregir lo producido.

Debido a que casi todos los defectos son inyectados en las fases de Diseño y Codificación, se estudia el comportamiento de la remoción de estos, ya que la cantidad de defectos inyectados en las otras fases es despreciable. Se calcula el porcentaje de los defectos removidos en cada una de las fases sobre el total de removidos, considerando de forma separada los que se inyectaron en Diseño, a los de Codificación. En primera instancia se calcula, para cada estudiante, el porcentaje de remoción de defectos en cada fase según el total de los defectos removidos. Finalmente, se promedian los valores obtenidos por todos los estudiantes por fase.

Para los cálculos se tienen disponibles los resultados de 92 estudiantes, pero algunos de ellos son descartados. En el caso de los defectos que fueron inyectados en Diseño, se consideran únicamente 83, ya que 9 de ellos no registraron remoción de defectos que hayan sido inyectados en Diseño. En cuanto a los inyectados en Codificación, se consideran los datos de 80 estudiantes; 10 de ellos fueron descartados por la misma razón y los 2 restantes, porque los registros generaban incertidumbre en cuanto a su correctitud, ya que la mayoría de los defectos inyectados en Codificación quedaron sin remover en las etapas consideradas (se removieron en la propia etapa).

En el Cuadro 4, se presenta el porcentaje promedio de remoción de defecto de los estudiantes, según las fases en que fueron inyectados. Como se puede observar en el caso de los inyectados en Diseño, el mayor porcentaje se da en la Revisión del diseño, así como también en el caso de los inyectados en Codificación, la mayor remoción se produce en la Revisión de codificación. Por lo cual a simple vista, se puede afirmar que las fases de Revisión son muy eficaces para encontrar defectos inyectados en la fase que se está revisando. De todas formas, para poder contestar qué tan beneficioso es encontrar defectos en ellas, hay que tener en cuenta el costo en el cual se incurre. El costo es considerado como el tiempo promedio dedicado a remover un defecto en dicha fase. En ambos casos coincide en que la siguiente fase en que se encuentran y remueven más defectos, para ambos casos, es en las Pruebas unitarias.

FASE DE INYECCIÓN	FASE DE REMOCIÓN (%)				
	Revisión de diseño	**Codificación**	**Revisión de codificación**	**Compilación**	**Pruebas unitarias**
Diseño	53,4	9,6	8,9	2,5	25,7
Codificación			62	16,6	21,4

Cuadro 4 - Porcentaje de remoción de defectos por fase

5.1.3 - Costo de remover los defectos inyectados, según *Time in phase*

En esta sección se analizan las diferencias entre los costos promedios que lleva remover un defecto en una determinada fase. Como se vió anteriormente, menos del 2% de los defectos son inyectados en alguna fase que no sea Diseño o Codificación. Por lo tanto para calcular el costo, se consideran únicamente aquellos defectos que son inyectados en Diseño o Codificación.

Las fases consideradas para la remoción de los defectos inyectados son: Revisión del diseño (DLDR), Revisión de código (CR), Compilación (Comp) y Pruebas unitarias (UT). Para cada estudiante, se calcula el costo de remover un defecto en cada una de las fases mencionadas, para luego poder realizar el promedio de los costos obtenidos. Se excluyen los estudiantes que no removieron defectos en dichas fases y aquellos cuyos datos ingresados generaban incertidumbre en cuanto a su correctitud (la cantidad de defectos inyectados era muy superior al total de removidos). Esto genera que el tamaño de la muestra a considerar para el cálculo del promedio es variable en cada una de las fases. Para DLDR se dispone de datos de 64 estudiantes mientras que 75 en CR, 44 para Comp y 72 en UT.

El costo de remover un defecto en una determinada fase es calculado como: el cociente entre el tiempo dedicado a la fase y la cantidad de defectos removidos en la misma.

El Cuadro 5, muestra el costo promedio de remover los defectos inyectados en cada una de las fases estudiadas.

	Revisión de diseño	**Revisión de codificación**	**Compilación**	**Pruebas unitarias**
Costo promedio	41,8	27,7	4,1	38,9

Cuadro 5 - Costo (en minutos) de remover defectos inyectados en Diseño y Codificación según fase de remoción

Como se puede observar, el costo promedio más bajo se produce en Compilación. Por otro lado, el costo en Revisión de codificación es significativamente inferior al de las Pruebas unitarias y a la Revisión de diseño, mientras que entre estas últimas es muy similar. Es importante tener en cuenta que no se está comparando el costo de remover un mismo defecto en cada una de las fases, sino que la comparación corresponde al costo de remover en determinada fase aquellos defectos que no fueron removidos en las fases previas. No se considera la fase de Codificación ya que por la forma que se calcula el costo, sería un error hacerlo. En este caso particular, la búsqueda de defectos no es una actividad propia de dicha fase. Su principal actividad es la de generar el código. Por lo tanto, para poder dar un resultado más exacto, se debe quitar, al tiempo en la fase, el tiempo que se estuvo escribiendo el código. Como este dato no está disponible, el cálculo no es realizado.

A diferencia de Codificación, la principal tarea en las otras fases es encontrar y corregir defectos y, salvo alguna excepción, no se realizan otras tareas. Como excepción puede ocurrir que en la fase de Pruebas unitarias se diseñen las pruebas, si no se diseñaron con anterioridad.

Teniendo en cuenta lo mencionado, se puede observar que el costo más bajo se da en la Compilación. Esto puede deberse a que los tipos de defectos que se encuentran en esta fase son más rápidos de resol-

ver. Estos defectos son generalmente de sintaxis del lenguaje, por lo cual considerando que los compiladores proporcionan “ayudas” para corregir el error cometido, la corrección resulta más simple. Siguiéndole en costo se encuentra la Revisión de codificación y por último los costos más altos se dan en la Revisión de diseño y en las Pruebas unitarias.

Teniendo en cuenta la eficiencia que tienen las Revisiones para encontrar defectos, se podría optar por asumir el costo para lograr un producto con menos defectos. En las Pruebas unitarias el costo es similar, pero se encuentran menos de la mitad de los defectos que en las Revisiones. Por tal motivo, se podría afirmar que es preferible realizar Revisiones antes que las Pruebas unitarias. Sin embargo, hay que tener cuidado con esta afirmación porque en estos resultados no presentamos ninguno relacionado con tipos de defectos encontrados, y puede suceder que en las Pruebas unitarias se detecten tipos de defectos que en las otras no se detectan.

5.2 - Análisis de la mejora en la *performance* del estudiante al aplicar *PSP*

En *The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers* se concluye que la aplicación del *PSP* mejora la calidad del producto [5]. Dicho de otra forma, mejora la performance del individuo ya que produce software de mejor calidad.

Humphrey, en el *PSP*, propone que para gestionar y así mejorar la calidad del trabajo realizado, por lo tanto la performance, no solo se debe medir la calidad sobre el producto, sino que se debe medir la calidad sobre el proceso [2].

Los ejercicios (programas) creados por los estudiantes durante los cursos de *PSP* tienen el mismo dominio de aplicación. Esto hace que entre ellos tengan funciones similares y en varios casos son reutilizados unos en otros -programación repetitiva-. Lo que conduce a preguntarse si la mejora se debe a la aplicación del *PSP* o se debe a la programación repetitiva.

El segundo subobjetivo del análisis estadístico, consiste en analizar si la mejora en la performance del estudiante, en los programas desarrollados aplicando el *PSP*, es producto de aplicar un proceso que guía su trabajo (*PSP*) o de la programación repetitiva. Para determinar la causa de dicha mejora, se estudia qué ocurre en los experimentos donde solo está presente la programación repetitiva. Se observa la evolución de dos medidas de calidad, para ver si esta mejora o no.

Para medir la calidad en la performance del sujeto, el *PSP* plantea varias medidas. Entre ellas, la relación entre el tiempo de permanencia en dos fases, *phase-time ratios*, esta es sugerida en el *PSP* [2].

En este trabajo, la observación de la evolución en la calidad de la performance se hace sobre dos medidas (indicadores). Una mide la calidad sobre el producto y la otra mide la calidad sobre el proceso. La medida que aplica sobre la calidad del producto, es la densidad de defectos en las Pruebas unitarias (UT), calculada como la cantidad de defectos encontrados cada mil líneas de código. La otra aplica sobre el proceso, es la relación entre el tiempo dedicado al Diseño (TDLD) y el tiempo dedicado a la Codificación (TCod), definida como: ratio TCod/TDLD. El *PSP* propone una la relación (ratio) 1 a 1 entre el tiempo en Codificación y Diseño, o sea por cada unidad de tiempo empleada en Codificación debe emplearse una en Diseño [2].

Para el análisis estadístico se realiza un contraste de hipótesis. Se plantea una hipótesis nula (H0) y una alternativa (H1). En el test aplicado a la densidad de defectos y al ratio (TCod/TDLD), se compara el valor de cada una de las medias en un programa respecto a otro realizado anteriormente.

5.2.1 - Conjunto de datos

El conjunto de datos se obtuvo en tres experimentos realizados en la Facultad de Ingeniería de la Universidad de la República. A diferencia de los cursos, en los tres experimentos no se enseño todo el proceso del *PSP*, sino que únicamente se incluyeron las versiones del proceso PSP0 y PSP0.1. En ambas versiones se registran datos pero no se aplican las prácticas y técnicas del proceso. Al ser el nivel más básico es comparable con realizar ejercicios sin un proceso, ya que las guías utilizadas solo indican los registros que se deben realizar. Por lo cual, a diferencia del trabajo *The personal software process: an empirical study of the impact of PSP on individual engineers*, el único factor que interviene es el de la programación repetitiva [5].

Los participantes son estudiantes de grado de Ingeniería en Computación con conocimientos en lenguajes de programación. Se realizaron un total de tres experimentos, en los años 2012, 2013 y 2014. En cada uno, los estudiantes realizaron una serie de ocho ejercicios: en el primero aplicaron el PSP0 y en el resto la versión PSP0.1. Tanto los ejercicios, como los *scripts* y la herramienta utilizada para el registro de datos son los mismos que se utilizan en los cursos. En el experimento desarrollado durante el 2012 participaron 12 estudiantes, en el de 2013 participaron 10 y en el 2014 la participación fue de 14.

5.2.2 - Estudio de la densidad de defectos en UT

En el trabajo que estudia cómo el *PSP* mejora de la calidad del software producido, se puede observar que a medida que se avanza del PSP0 a PSP2 la densidad de defectos encontrados en UT disminuye [5]. Este resultado está afectado por dos factores: el proceso y la programación repetitiva. Con el objetivo de controlar uno de los factores, es que se realiza el estudio con datos de experimentos donde solo interviene como factor la programación repetitiva.

Este estudio consiste en observar si existe variación de la densidad de defectos a lo largo de los programas creados en el experimento. Si esta corresponde a un descenso en la cantidad, existe una mejora en la calidad del producto desarrollado, por lo tanto en la performance del estudiante. Este mismo estudio se realizó antes, únicamente, con los datos del primer experimento [4]. Este trabajo lo amplía para dos experimentos más.

La densidad de defectos en UT se define como la cantidad de defectos encontrados en UT cada mil líneas de código (def UT/KLOC).

Test estadístico

Para analizar variaciones en la densidad de defectos encontrados en UT, se realiza una comparación de a pares de programas. Se comparan las muestras obtenidas para un programa respecto a otro realizado anteriormente. Cada muestra corresponde a un conjunto formado por la densidad de defectos en UT para cada estudiante en el programa.

Las hipótesis nula y alternativa planteadas son:

H0: No existe variación significativa en la densidad de defectos encontrados en UT en el programa j con respecto al programa i. Donde i, j varían de 1 a 8 e i<j.

H1: Existe variación significativa en la densidad de defectos encontrados en UT en el programa j con respecto al programa i. Donde i, j varían de 1 a 8 e i<j.

Los estudiantes son los mismos en ambos programas, por lo tanto las muestras para cada programa están relacionadas. Considerando que se está ante muestras apareadas, que el tamaño de la muestra es pequeño y que a priori no se conoce en qué sentido se puede dar la variación, el test estadístico a realizar es el “Test de rangos de Wilcoxon de dos colas para muestras apareadas”. Es un test no paramétrico que se ejecuta para comparar dos muestras cuando no se puede suponer normalidad.

Las hipótesis para el test aplicado a la densidad de defectos en UT son:

H0: Mediana(densidad de defectos en UT i) = Mediana(densidad de defectos en UT j)

H1: Mediana(densidad de defectos en UT i) <> Mediana(densidad de defectos en UT j)

Donde i,j son los programas del 1 al 8, i < j

Se define el nivel de significancia (α o pvalue) igual a 0.05, a partir de este valor la aplicación del test no rechaza la hipótesis nula. Por lo tanto, si el *pvalue<=0.05* la hipótesis nula (H0) es rechazada y si *pvalue>0.05* no es posible rechazarla.

Resultados y discusión

La presentación de los resultados está dividida en cuatro casos, uno por cada uno de los tres experimentos y un cuarto donde se trabaja con los datos de todos los experimentos en forma conjunta.

En primera instancia se realiza un estudio de la normalidad de las muestras en cada programa, para confirmar que no la cumplan. En los gráficos *Box and Whisker* presentados en Figura 13 se observa que la mediana se acerca al primer o al tercer cuartil, por lo cual las muestras no cumplen con una distribución normal.

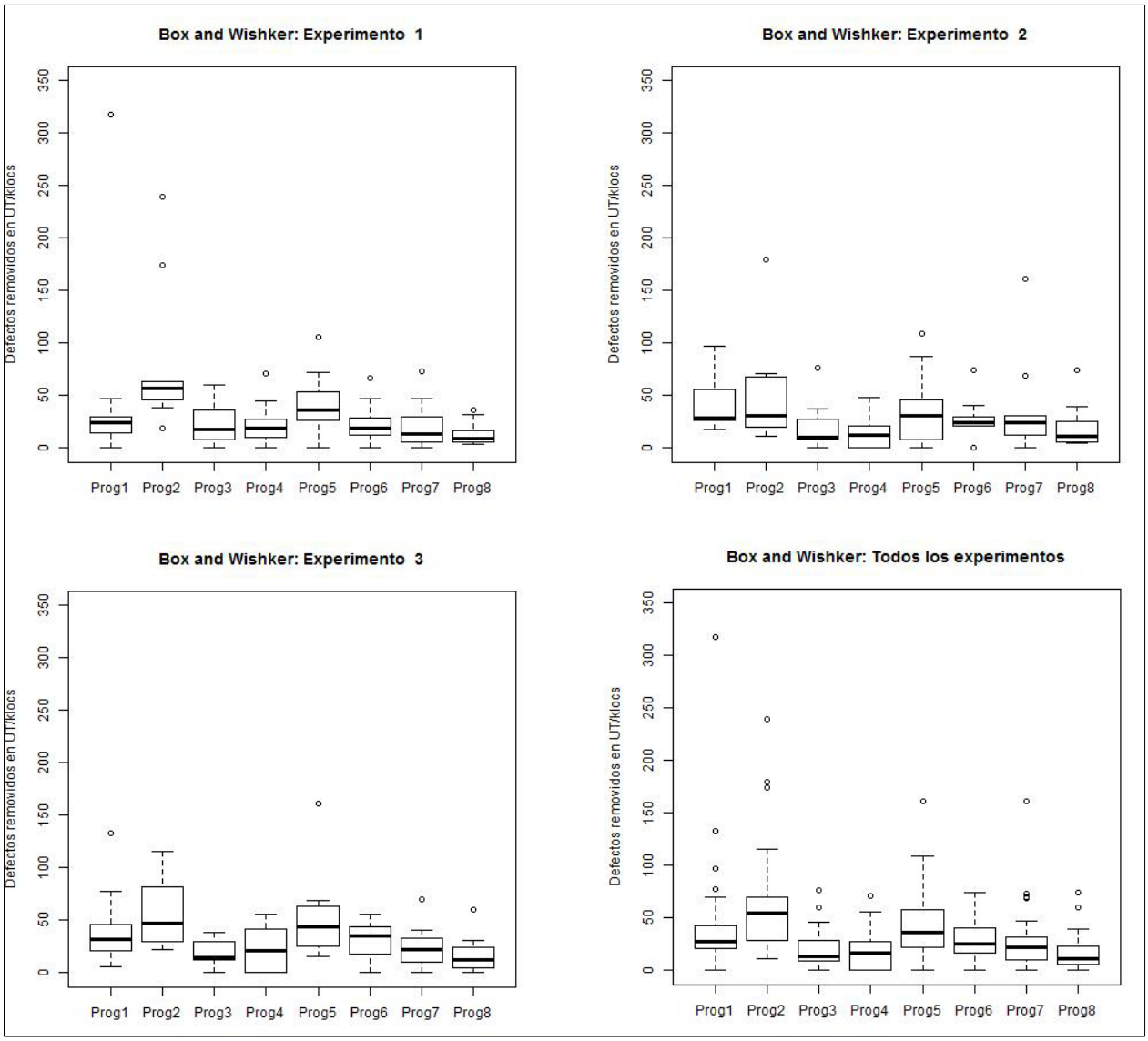

Figura 13 - ***Box and Whisker para densidad de defectos en UT***

En segunda instancia se aplica el test de rangos de Wilcoxon de dos colas para muestras apareadas a cada par de programa. Cuando la hipótesis nula es rechazada interesa conocer en qué sentido se da la variación, para ello se realiza una comparación de medianas entre el par de muestras:

- Si Mediana(densidad de defectos en UT i) > Mediana(densidad de defectos en UT j), mejoró la calidad ya que se llega a UT con menos defectos a medida que se avanza en los ejercicios.
- Si Mediana(densidad de defectos en UT i) < Mediana(densidad de defectos en UT j), empeoró la calidad ya que se llega a UT con más defectos a medida que se avanza en los ejercicios.

Experimento 1: Media, mediana, iqr de la densidad de defectos en UT								
	Prog 1	Prog 2	Prog 3	Prog 4	Prog 5	Prog 6	Prog 7	Prog 8
MEDIANA	24,5499	56,9763	18,1218	18,4807	36,3757	18,4034	13,7824	8,5839
IQR	14,4147	16,7818	27,7706	17,2674	27,3426	15,9071	23,1838	10,8321
MEDIA	46,8442	76,2998	22,4452	22,0444	41,9786	22,4160	21,4278	12,8208
Experimento 2: Media, mediana, iqr de la densidad de defectos en UT								
	Prog 1	Prog 2	Prog 3	Prog 4	Prog 5	Prog 6	Prog 7	Prog 8
MEDIANA	28,1248	30,7927	10,0060	11,7909	30,9804	24,0999	24,1546	10,8540
IQR	29,6311	47,2424	19,6741	20,4082	38,0237	9,0036	18,8644	19,1084
MEDIA	41,2803	51,2493	19,3482	15,7975	36,8450	28,0911	37,9180	19,7080
Experimento 3: Media, mediana, iqr de la densidad de defectos en UT								
	Prog 1	Prog 2	Prog 3	Prog 4	Prog 5	Prog 6	Prog 7	Prog 8
MEDIANA	32,0551	46,5926	14,4620	20,7976	43,2749	35,0986	22,2332	12,5106
IQR	24,6747	52,5622	17,7839	41,2371	37,6501	26,2369	22,6193	19,5941
MEDIA	39,4877	58,5157	17,7201	21,1082	49,4430	31,6455	23,8460	16,6101
Todos los experimentos: Media, mediana, iqr de la densidad de defectos en UT								
	Prog 1	Prog 2	Prog 3	Prog 4	Prog 5	Prog 6	Prog 7	Prog 8
MEDIANA	27,8872	54,7957	13,7757	16,5845	36,3757	25,3530	21,9807	10,5733
IQR	21,9207	40,5590	20,1005	27,7832	36,4198	24,2194	21,4311	17,6508
MEDIA	42,4378	62,4253	19,7474	19,9451	43,4554	27,5817	26,9488	16,2075

Cuadro 6 - **Mediana, media e intercuartil para la densidad de defectos en UT**

Finalmente se calcula el *effect size,* medida para cuantificar la variación entre cada par de muestras. El cálculo se realiza en los casos en que la H0 es rechazada. Para medirlo se utiliza la medida estandarizada, *d* de Cohen, la cual clasifica el *effect size* en: insignificante, pequeño, medio y grande. En el Anexo 6 - Cálculo de la d de Cohen para la densidad de defectos en UT se muestran los cuadros del cálculo del *d* de Cohen.

A continuación se presentan cuatro cuadros donde se muestran los resultados obtenidos luego de aplicar el test a cada uno de los casos. En cada cuadro se muestran los resultados de aplicar el test de rangos de Wilcoxon de dos colas para muestras apareadas, comparación de las medianas y la *d* de Cohen entre cada par de programas.

Para cada par, se muestra el valor del *pvalue* del test de Wilcoxon. Las celdas en que no se puede rechazar la hipótesis nula (*pvalue* > 0.05) están marcadas con gris. Si la hipótesis nula es rechazada (*pvalue* <= 0.05) se marcan con tonos de verde o rojo. Al producirse una mejora en la calidad, dada por la densidad de defectos es indicada con tonos de verde y en caso contrario con tonos de rojo. Este resultado surge de la comparación de medianas. Las tonalidades de los colores indican la significancia de la *d* de Cohen: cuanto más oscuro es el tono más significativa es la magnitud de la diferencia. En el Cuadro 7 se presenta la leyenda para interpretar los cuadros.

d Cohen		p<=0,05 y mei>mej	p<=0,05 y mei<mej	p>0,05
INSIGNIFICANTE	d<0,2			
PEQUEÑO	d<0,5			
MEDIO	d<0,8			
GRANDE	d>0,8			

Cuadro 7 - **Leyenda para interpretar los cuadros del test de Wilcoxon, comparación medianas y d Cohen**

En el Cuadro 8, se puede observar que es estadísticamente significante la mejoría que se produce en todos los programas (excepto el 1) con respecto al programa 2. En el artículo que se está complementando, el cual estudia si la mejora en la performance se debe a la programación repetitiva o al *PSP,* se plantea que esto puede ser consecuencia de que las características de dicho programa difieren de los otros [4]. El programa 2 es el único donde no se resuelve un problema matemático, es un programa que cuenta las líneas

de código de un programa (LOCs).

La mejoría del programa 6 y 8 respecto al 5 también es estadísticamente significante, mientras que la hipótesis nula no puede ser rechazada con respecto a 3, 4 y 7. Por otro lado en el programa 8 se producen algunas mejorías estadísticamente significantes.

En general se puede observar que no hay una mejora continua al avanzar en los ejercicios. Considerando los ejercicios 3, 4, 6 y 7 no es posible detectar diferencias significativas con el resto de los ejercicios. El 8 es el que presenta más mejorías respecto a otros ejercicios (1,2 y 5).

TABLA TEST WILCOXON, DE EXPERIMENTO: 1							
programa	2	3	4	5	6	7	8
1	0,028056	0,722108	0,157939	0,346522	0,136097	0,388186	0,006040
2		0,006040	0,002873	0,018603	0,002218	0,009633	0,002218
3			0,753684	0,084379	0,937473	0,753684	0,272095
4				0,116664	0,929153	1,000000	0,136097
5					0,015022	0,084379	0,006040
6						0,929153	0,084379
7							0,209427

Cuadro 8 - Test de Wilcoxon para la densidad de defectos en UT, experimento 1

En el Cuadro 9 se observa que los programas 3, 4 y 8 con respecto al 1 y al 2 presentan diferencias estadísticamente significantes. En el caso anterior, se planteó el motivo por el cual pueden existir diferencias respecto al programa 2. En cuanto al programa 1 se observa que respecto a los otros programas no es posible detectar diferencias.

Se observa que el programa 5 respecto al 3 y 4 no solo no mejora, sino que empeora de forma medianamente significativa, la densidad de defectos en UT. Lo mismo sucede en el programa 7 con respecto al 4.

Sintetizando, nuevamente se puede observar que en la comparación entre programas no es posible detectar diferencias significativas que lleven a pensar en una real variación en la densidad de defectos en UT. Por otro lado, contrariamente a lo que sucede en los cursos de *PSP,* en el avance de los ejercicios se puede observar algunos casos en que el cambio producido es negativo. De todas formas se produce alguna mejoría significativa del programa 8 respecto al resto. Uno de los motivos por el cual puede suceder, es que por más que no se aplique el proceso, sí está midiendo su trabajo. Esto puede resultar en que sea más consciente de los errores que comente.

TABLA TEST WILCOXON, DE EXPERIMENTO: 2							
programa	2	3	4	5	6	7	8
1	0,646462	0,016605	0,021824	0,284503	0,202622	0,284503	0,012515
2		0,005062	0,006910	0,168807	0,114128	0,284503	0,005062
3			0,441268	0,038152	0,109745	0,092601	0,959354
4				0,038152	0,066316	0,007686	0,386271
5					0,678402	0,798859	0,046853
6						0,798859	0,114128
7							0,139414

Cuadro 9 - Test de Wilcoxon para la densidad de defectos en UT, experimento 2

En el Cuadro 10 se observa nuevamente, que respecto al programa 2, varios programas han tenido mejoras estadísticamente significativas, como es el caso de los programas 3,4,7 y 8.

A diferencia de los experimentos anteriores, en este se podría afirmar que la significancia en la variación es más alta, ya sea por una mejora o por un empeoramiento en la calidad del programa.

Nuevamente se muestra un empeoramiento en la densidad de defectos en UT del programa 5 respecto al 3 y 4. A su vez en los programas 6,7 y 8 se produce una mejoría mediana y altamente significativa respeto al 5. Esto puede deberse a alguna característica particular que tenga dicho programa que al no aplicar técnicas de revisión para detectar con anterioridad los defectos, los mismos no puedan ser detectados hasta llegar a las Pruebas unitarias. Por otro lado en el programa 8 se produce una importante mejoría respecto a cuatro de los programas, y que se viene reiterando desde los otros experimentos. Esto da lugar a plantear que por más que no exista una mejora continua significativa al avanzar en los ejercicios, al llegar al último de ellos sí se produzca una mejora. Tal vez por el hecho de que el estudiante ha asimilado los errores que ha cometido en los ejercicios previos.

TABLA TEST WILCOXON, DE EXPERIMENTO: 3							
programa	2	3	4	5	6	7	8
1	0,096197	0,006319	0,198123	0,220899	0,826091	0,064039	0,018567
2		0,000982	0,001887	0,396726	0,055533	0,002865	0,000982
3			0,700703	0,000982	0,027708	0,300290	0,649644
4				0,018567	0,198012	0,972125	0,463071
5					0,035465	0,015654	0,001887
6						0,220899	0,025843
7							0,124043

Cuadro 10 - Test de Wilcoxon para la densidad de defectos en UT, experimento 3

El Cuadro 11 muestra el resultado de considerar todos los experimentos de forma conjunta. Como se puede observar, continúa existiendo una mejora significativa del resto de los programas con respecto al programa 2; mientras que con respecto al programa 1 existe una insignificante o leve mejoría, exceptuando en la comparación con el programa 2. Por otro lado en los programas 7 y 8, sobre todo en el 8, se observan mejorías respecto a los anteriores pero únicamente un par de ellas son significativas estadísticamente. También en dichos programas respecto a los ejercicios 3 y 4 no hay diferencias que puedan ser detectadas. Esto acompaña la teoría planteada en el experimento anterior, que tal vez por el hecho de tomar medidas y registrar al llegar a los últimos programas pueda darse alguna diferencia. De todas formas, es una teoría que únicamente con el análisis realizado no se puede confirmar. Se continúan observando diferencias donde empeora de forma estadísticamente significante la densidad de defectos.

TABLA TEST WILCOXON, TODOS LOS EXPERIMENTOS							
programa	2	3	4	5	6	7	8
1	0,008699	0,001758	0,005423	0,499323	0,116169	0,020918	0,000010
2		0,000000	0,000000	0,008699	0,000029	0,000075	0,000000
3			0,713190	0,000028	0,030564	0,192241	0,294517
4				0,000314	0,050279	0,249126	0,359021
5					0,005097	0,008699	0,000003
6						0,611626	0,001084
7							0,015545

Cuadro 11 - Test de Wilcoxon para la densidad de defectos en UT, todos los experimentos

Resumen

En suma, considerando los experimentos realizados, no se detecta una mejora continua en la densidad de defectos en UT en el avance de los ejercicios, como la que ocurre en los cursos del *PSP*. Observando los cuadros, se puede ver que en muchos casos no se detectan diferencias significativas. En donde las hay, es en otros programas respecto al ejercicio 2 y al 5. Esto conduce a pensar que se pueda deber a particularidades en las características de ambos programas. De todas formas no es una afirmación que se pueda tomar con total veracidad.

En el último ejercicio se detecta mejora, en algunos casos significativa, respecto a los anteriores. Por lo tanto, más allá de que no exista una mejora continua, si puede existir alguna mejoría al llegar al final de la serie de ejercicios. Uno de los motivos por los cuales puede surgir, es por el simple hecho de que los estudiantes no están habituados a medir su trabajo y que tan solo el hecho de registrar los errores que cometen, van asimilándolos y por lo tanto al final dejan de cometer algunos de ellos. Esto es una teoría, no tiene porqué ser lo que realmente sucede. Se podría realizar algún estudio analizando los tipos de errores que cometen a medida que se avanza a ver si se detecta esa diferencia. Por ejemplo, viendo si tal vez algún tipo de error aparece de forma menos frecuente al avanzar.

De todas formas las mejoras que se producen no son continuas tal cual presenta el trabajo que estudia la mejora de la calidad en los cursos de PSP [5]. Esto lleva a concluir que la mejora viene dada por la aplicación de las técnicas del PSP y no por la repetición de programas.

5.2.3 - Evolución del ratio: tiempo de Codificación/tiempo en Diseño

Para que el proceso realizado por el individuo al crear un programa sea de calidad, el *PSP* propone que el tiempo dedicado a Codificación (TCod) y el tiempo dedicado a Diseño (TDLD) deben estar dados por una relación 1 a 1 [2]. O sea, por cada unidad de tiempo dedicada a Codificación debe dedicarse una a Diseño.

A partir de las versiones PSP2.0 y PSP2.1 se incluyen *templates* de diseño, notación y medidas para gestionar la calidad del producto. Esto hace que el sujeto, a medida que avanza en los ejercicios, se acerque al valor sugerido.

Este estudio consiste en observar si existe una variación en el ratio TCod/TDLD a medida que se evoluciona en la creación de programas durante el experimento. En caso de existir, se analiza qué tan significativa es la variación y si esta provoca una mejora o empeora la calidad del proceso. Para ver si existe variación se comparan de a pares de programas.

Dependiendo del tiempo dedicado a cada fase, el valor del ratio (TCod/TDLD) puede resultar: uno, mayor que uno o menor. Al comparar dos programas donde, en ambos casos, el valor del ratio es menor que uno, si el ratio en determinado programa es menor que en del programa anterior, esto implica que la diferencia entre los tiempos aumentó. Por lo tanto empeora la calidad del proceso realizado por el sujeto; en caso contrario, mejoró. En cambio, si en ambos programas el valor del ratio es mayor que uno, cuando el ratio de un programa es menor que el del programa anterior, implica una mejoría ya que la diferencia entre ambos se hace más chica; en caso, contrario empeora.

Test estadístico

Para analizar si existen variaciones en el ratio (TCod/TDLD), se comparan las muestras obtenidas de a pares de programas. Se compara el valor en un programa respecto a otro realizado anteriormente. El conjunto de muestras utilizados para cada programa, son los ratios calculados para cada estudiante en cada programa.

Las hipótesis nula y la hipótesis alternativa planteadas son:

H0: No existe variación significativa en el ratio TCod/TDLD,en el programa j respecto al i. Donde i, j varían de 1 a 8 e i<j.

H1: Existe variación significativa en el ratio TCod/TDLD, en el programa j respecto del programa i. Donde i, j varían de 1 a 8 e i<j.

Como las muestras están apareadas, el tamaño de la muestra es pequeño y, a priori, no se conoce el sentido de la relación, se aplica el Test de rangos de Wilcoxon de dos colas para muestras apareadas, con nivel de significancia α=0.05. Por lo tanto, si el pvalue <= 0.05, la hipótesis nula (H0) es rechazada y si pvalue > 0.05 no es posible rechazarla.

Las hipótesis para el test aplicado al ratio TCod/ TDLD son:

H0: Mediana(ratio TCod/TDLD i) = Mediana(ratio TCod/TDLD j)

H1: Mediana(ratio TCod/TDLD i) <> Mediana(ratio TCod/TDLD j)

Donde i,j son los programas del 1 al 8, i < j

Resultados y discusión

La presentación de los resultados está dividida en cuatro casos, uno por cada uno de los tres experimentos y un cuarto donde se trabaja con los datos de todos los experimentos en forma conjunta.

Previo a realizar el test de Wilcoxon se estudia la normalidad de las muestras. En las gráficas de la Figura 14, se observa que la mediana no se encuentra equidistante de los dos cuartiles como se espera en una distribución normal, o sea el 50% de la muestra no está bajo la normal.

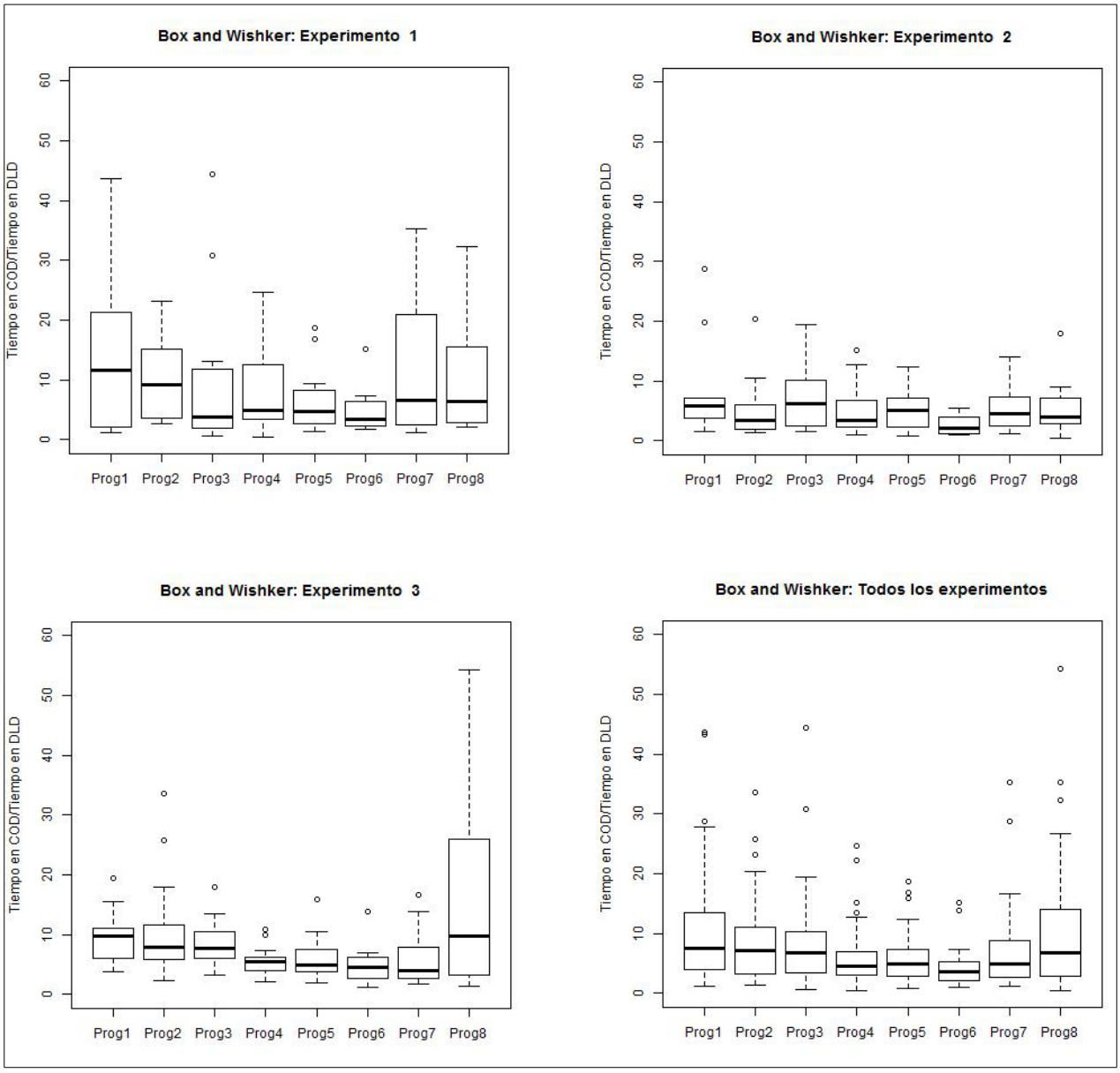

Figura 14 - ***Box and Whisker para tiempo COD/DLD por programa***

Dado que la muestras no cumplen con el criterio de normalidad, es posible aplicar el Test de rangos de Wilcoxon de dos colas para muestras apareadas a cada par de programa. Dicho test solo observa la variación, pero no el sentido en que se da. Por lo tanto, en caso de que la hipótesis nula sea rechazada, se comparan medianas.

Se realiza la comparación considerando los siguientes cuatro casos :

- Si mediana(ratio TCod/TDLD i) < 1 y mediana(ratio TCod/TDLD j) < 1:
  - Si mediana(ratio TCod/TDLD i) > mediana(ratio TCod/TDLD j), empeoró la calidad del proceso desarrollado, esto es consecuencia de que a medida que se avanza en los ejercicios se distancian más de la relación 1 a 1.
  - Si mediana(ratio TCod/TDLD i) < mediana(ratio TCod/TDLD j), mejoró la calidad del proceso desarrollado.
- Si Mediana(ratio TCod/TDLD i) > 1 y mediana(ratio TCod/TDLD j) > 1:
  - Si mediana(ratio TCod/TDLD i) > mediana(ratio TCod/TDLD j), mejoró la calidad del proceso desarrollado, esto es porque a medida que se avanza se acerca mas de la relación 1 a 1.
  - Si mediana(ratio TCod/TDLD i) < mediana(ratio TCod/TDLD j), empeoró la calidad del proceso desarrollado.

Cualquier otra casuística, se analiza en forma particular dado que a priori no es tan directa la generalización.

En el Cuadro 12 se muestran las medianas con las que se realiza la comparación. Como se puede observar la mediana del ratio TCod/TDLD en cada programa de cada experimento es mayor que uno. Por lo tanto el test de comparación de medianas se hace sobre el caso en que: mediana(ratio TCod/TDLD i) > 1 y mediana(ratio TCod/TDLD j) > 1 .

Experimento 1: Media, mediana, iqr de la tasa (tCOD/tDLD)								
	Prog 1	Prog 2	Prog 3	Prog 4	Prog 5	Prog 6	Prog 7	Prog 8
MEDIANA	11,5738	9,1288	3,6859	4,9121	4,7479	3,3640	6,4972	6,2846
IQR	19,0990	11,4591	9,8905	9,0134	5,5534	4,0358	18,5420	12,7618
MEDIA	15,0925	9,8153	9,9191	8,3696	6,5833	18,5461	50,5987	13,6814
Experimento 2: Media, mediana, iqr de la tasa (tCOD/tDLD)								
	Prog 1	Prog 2	Prog 3	Prog 4	Prog 5	Prog 6	Prog 7	Prog 8
MEDIANA	5,7987	3,3726	6,2466	3,4522	5,0111	2,1078	4,5030	3,9089
IQR	3,4170	4,1576	7,6936	4,4218	4,9614	2,7719	4,7786	4,2740
MEDIA	8,4063	5,5722	7,7272	5,5209	5,1764	2,7499	5,4205	5,6536
Experimento 3: Media, mediana, iqr de la tasa (tCOD/tDLD)								
	Prog 1	Prog 2	Prog 3	Prog 4	Prog 5	Prog 6	Prog 7	Prog 8
MEDIANA	9,6876	7,8015	7,5926	5,4753	4,8184	4,5694	3,9079	9,7713
IQR	4,9151	5,9412	4,4453	2,3778	3,6845	3,6478	5,2091	22,7717
MEDIA	9,6360	10,9073	8,2417	5,5960	6,1023	4,8345	5,9331	15,5089
Todos los experimentos: Media, mediana, iqr de la tasa (tCOD/tDLD)								
	Prog 1	Prog 2	Prog 3	Prog 4	Prog 5	Prog 6	Prog 7	Prog 8
MEDIANA	7,5833	7,1344	6,7206	4,5915	4,8184	3,6667	4,8884	6,7257
IQR	9,5941	7,9170	6,9430	3,9076	4,5127	3,2648	6,2463	11,1071
MEDIA	11,1554	9,0086	8,6698	6,5255	6,0027	8,9400	21,1006	12,0665

Cuadro 12 - Mediana, media e intercuartil para la tasa (TCod/TDLD)

En los casos en que la hipótesis nula es rechazada, se calcula el valor del *effect size* por medio de la d de Cohen. En el Anexo 7 – Cálculo de la d de Cohen para la tasa TCod/TDLD se pueden observar los cuadros del cálculo del d de Cohen, sobre el cual se realiza la clasificación.

En los siguientes cuadros se presentan los resultados obtenidos para cada par de programas. Se presenta un cuadro por experimento y otro en el que se consideran los tres experimentos de forma conjunta. Al igual que en la sección anterior, en los cuadros se agrupan los resultados obtenidos al aplicar el test de rangos de Wilcoxon de dos colas para muestras apareadas, comparación de las medianas y la d de Cohen entre cada par de programas. Los colores y tonos son aplicados de la misma forma que en el caso anterior. En el Cuadro 7 se encuentra la leyenda para interpretar los cuadros.

El Cuadro 13 presenta los resultados de aplicar Wilcoxon para cada par de programa con la hipótesis

para el ratio: TCod/TDLD. El programa 4 es el único que presenta una mejora con respecto a algún programa, el primero. Se observa que el ratio en los programas 7 y 8 empeora con una pequeña significancia estadística respecto al programa 3. En el caso del programa 7, también lo hace respecto del 6. Mientras que el 8 respecto al 5 empeora con una significancia más alta. Sin embargo, en el resto de los programas no se presenta ninguna diferencia significativa. Dado que el valor de la mediana(ratio TCod/TDLD i) es distinto de uno en todos los casos, conduce a pensar que los estudiantes, a medida que avanzan, no consideran equiparar el tiempo que se dedica a ambas fases.

TABLA TEST WILCOXON, DE EXPERIMENTO: 1							
programa	2	3	4	5	6	7	8
1	0,432768	0,059739	0,041389	0,099481	0,272095	0,813945	0,813945
2		0,239317	0,637870	0,157939	0,182338	0,753684	0,753684
3			0,637870	0,937473	0,753684	0,012063	0,009633
4				0,432768	0,875329	0,116664	0,209427
5					0,480177	0,116664	0,049860
6						0,034170	0,182338
7							0,182338

Cuadro 13 - Test de Wilcoxon para la tasa (TCod/TDLD), experimento 1

En el Cuadro 14 se observa que el programa 6 presenta mejoras respecto a los anteriores, siendo las mismas mediana o altamente significativas, mientras que en el resto no se nota una diferencia estadísticamente significativa. Lo que conduce a pensar que el programa 6 sea un programa “especial”. Se puede decir que el ratio:TCod/TDLD en comparación con los anteriores programas se hace más cercano al valor uno. En los programas 7 y 8 se observa que aumenta el ratio respecto al programa 6, lo cual es coherente con la mejoría que se produce en el programa 6 respecto a los anteriores.

TABLA TEST WILCOXON, DE EXPERIMENTO: 2							
programa	2	3	4	5	6	7	8
1	0,092601	0,798859	0,059336	0,284503	0,006910	0,202622	0,028417
2		0,114128	0,878482	0,959354	0,046853	0,721277	0,721277
3			0,005062	0,059336	0,006910	0,059336	0,139414
4				0,646462	0,012515	0,721277	0,507624
5					0,028417	0,575062	0,507624
6						0,036658	0,028417
7							0,798859

Cuadro 14 - Test de Wilcoxon para la tasa (TCod/TDLD), experimento 2

En el Cuadro 15 del experimento 3, nuevamente aparece una mejora significativa que se produce en el programa 6, aunque en este caso únicamente respecto a los programas 1 al 3. Se observa que los programas 4 y 5 mejoran significativamente respecto al primero, mientras que el 4 tiene una diferencia respecto al 3 y el 5 respecto al 2. Considerando que la mayoría de los programas no presentan diferencias estadísticamente significativas, se observa que no existe ninguna mejora continua en el ratio. Nuevamente el programa 8, comparado con tres de los otros programas, muestra un aumento significativo en el ratio. Esto puede deberse a que en el programa 8 se reutiliza parte del 6, por lo tanto, parte de los tiempos que se incurriría en realizar el diseño desde cero está incluido en los tiempos del programa 6.

TABLA TEST WILCOXON, DE EXPERIMENTO: 3							
programa	2	3	4	5	6	7	8
1	0,649644	0,151956	0,033047	0,019223	0,019223	0,074736	0,649644
2		0,310897	0,074736	0,039243	0,023130	0,115852	0,600179
3			0,046399	0,054624	0,015906	0,115852	0,310897
4				0,916512	0,196051	0,151956	0,039243
5					0,463071	0,310897	0,074736
6						0,310897	0,033047
7							0,010747

Cuadro 15 - Test de Wilcoxon para la tasa (TCod/TDLD), experimento 3

El resultado del test de Wilcoxon mostrado en el Cuadro 16 corresponde a la unificación de todos los experimentos. En este caso se muestran mejorías de los programas 3, 4, 5 y 6 respecto a programa 1, pero ninguna estadísticamente significativa. Por otro lado el programa 5 muestra una pequeña mejoría respecto al programa 2 y al 3. Mientras que el programa 6 muestra una mejoría insignificante respecto a los programas anteriores, los programas 7 y 8 presentan un insignificante aumento del ratio. Analizando el programa 8, nuevamente se detecta una diferencia respecto al 4 y al 5, el ratio en dichos programas es menor que en el programa 8. El motivo puede estar dado por lo expuesto en el experimento 3.

TABLA TEST WILCOXON, TODOS LOS EXPERIMENTOS							
programa	2	3	4	5	6	7	8
1	0,279687	0,036035	0,000741	0,001403	0,000741	0,101440	0,376441
2		0,376441	0,149482	0,024836	0,001403	0,385343	0,947763
3			0,064192	0,024836	0,001403	0,755646	0,136092
4				0,588844	0,043943	0,634790	0,020027
5					0,061871	0,461085	0,007590
6						0,003551	0,001758
7							0,201400

Cuadro 16 - Test de Wilcoxon para la tasa (TCod/TDLD), todos los experimentos

Resumen

En resumen, considerando que para cada programa en cada experimento el valor de la mediana del ratio es mayor que uno y no observando en los cuadros variaciones estadísticamente significativas, se puede decir que no existe una mejora continua en el ratio TCod/TDLD. Esto es por no poder rechazar la hipótesis nula o porque, en los casos en que se rechaza, la diferencia no es amplia. El caso particular se da en el programa 8 donde el ratio aumenta respecto a los programas 4, 5 y 6. Esto puede ser consecuencia de que en dicho programa se reutiliza código de otro, por lo cual los tiempos dedicados a Codificación y Diseño tienen una brecha más amplia, ya que “parte del trabajo está realizado”.

En cuanto a los cursos del *PSP* la mejora se produce sobre todo a partir del programa 7 (PSP2.0) en adelante, que es cuando el estudiante comienza a realizar Revisiones de diseño, utilizar *templates* de diseño, notación y técnicas de verificación especificas de diseño. Lo que conducen a que el estudiante tenga más consideraciones en la fase de desarrollo.

Nuevamente, pero utilizando otra medida se concluye que la programación repetitiva no es la causa de la mejora de la performance del individuo, sino que lo es el *PSP.*

5.2.4 - Estudio de la proporción N (TCod=N*TDLD) por programa

Con los resultados de la sección anterior se concluyó que no existe una variación estadísticamente significativa en el ratio TCod/TDLD solo por realizar programación repetitiva. A su vez, considerando los valores obtenidos en la mediana del ratio para cada programa, se observó que la misma no era cercana al valor propuesto, sino que era mayor.

Esto motiva a realizar un estudio adicional, el cual no fue planteado dentro de los objetivos. El mismo consiste en observar el poco tiempo que los estudiantes dedican a la fase de Diseño, en comparación al dedicado a la fase de Codificación. Esto se hace mediante el estudio de la proporción N (TCod=N*TDLD) que los relaciona, o sea, se busca hallar dicho valor N. A diferencia de los análisis estadísticos anteriores, en este caso no se observa la evolución de la proporción (no se comparan valores entre pares de programas), sino que se comparan los tiempos dedicados a cada fase en un mismo programa.

Test estadístico

Las variables dependientes sobre la que se realiza el estudio son los tiempos de Codificación y Diseño. El test realizado es aplicado cuatro veces, donde el N varía de 2 a 5. El estudio estadístico corresponde a un contraste de hipótesis. En este caso el programa es fijo y se compara entre dos conjuntos de muestras del mismo programa: tiempo invertido en la fase de Codificación por cada estudiante en el programa i y tiempo de Diseño multiplicado por un factor N para el mismo programa i. A ello se agrega un estudio, donde se consideran las muestras de todos los programas juntos.

Las hipótesis nula y alternativa planteadas para cada experimento son:

H0: El tiempo de Codificación en i es menor o igual que N veces el tiempo en Diseño.

H1: El tiempo de Codificación en i es mayor que N veces el tiempo en Diseño.

Donde i corresponde a uno de los programas del 1 al 8, o al agrupamiento de las muestras de los ocho programas.

En este caso el test a realizar corresponde al “Test de rangos de Wilcoxon de una cola para muestras apareadas”. La prueba a realizar es de una cola porque asumimos el sentido de la diferencia (menor o igual), las muestras están apareadas porque los pares muestreados corresponden a un mismo programa y son los mismos estudiantes. Se opta por Wilcoxon porque el tamaño de la muestra es pequeño.

Las hipótesis para el test aplicado a la proporción entre los tiempo de Codificación y Diseño para cada programa son:

H0: Mediana(TCod i) <= Mediana(N*TDLD i)

H1: Mediana(TCod i) > Mediana(N*TDLD i)

Donde i corresponde a uno de los programas del 1 al 8, o al agrupamiento de las muestras de los ocho programas.

El nivel de significancia α se toma con valor 0.05.

Resultados y discusión

La presentación de los resultados está dividida en cuatro casos, uno por cada uno de los tres experimentos y un cuarto donde se trabaja con los datos de todos los experimentos en forma conjunta.

Previo a la aplicación del test, se realizó un estudio de la normalidad para verificar que las muestras de cada programa no cumplan con dicha distribución, dado que es uno de los requisitos para aplicar el test. En Figura 15 se grafica para cada programa el tiempo en Diseño y el tiempo en Codificación. Se observa que la media de ambos tiempos se encuentra en algunos casos más cerca del primer o en otros del tercer cuartil, pero nunca centrada. En este caso, se grafica el tiempo en Diseño sin multiplicarlo por el factor N pero en caso de hacerlo se obtendría el mismo resultado ya que al multiplicar toda la muestra por el mismo factor no varía su distribución.

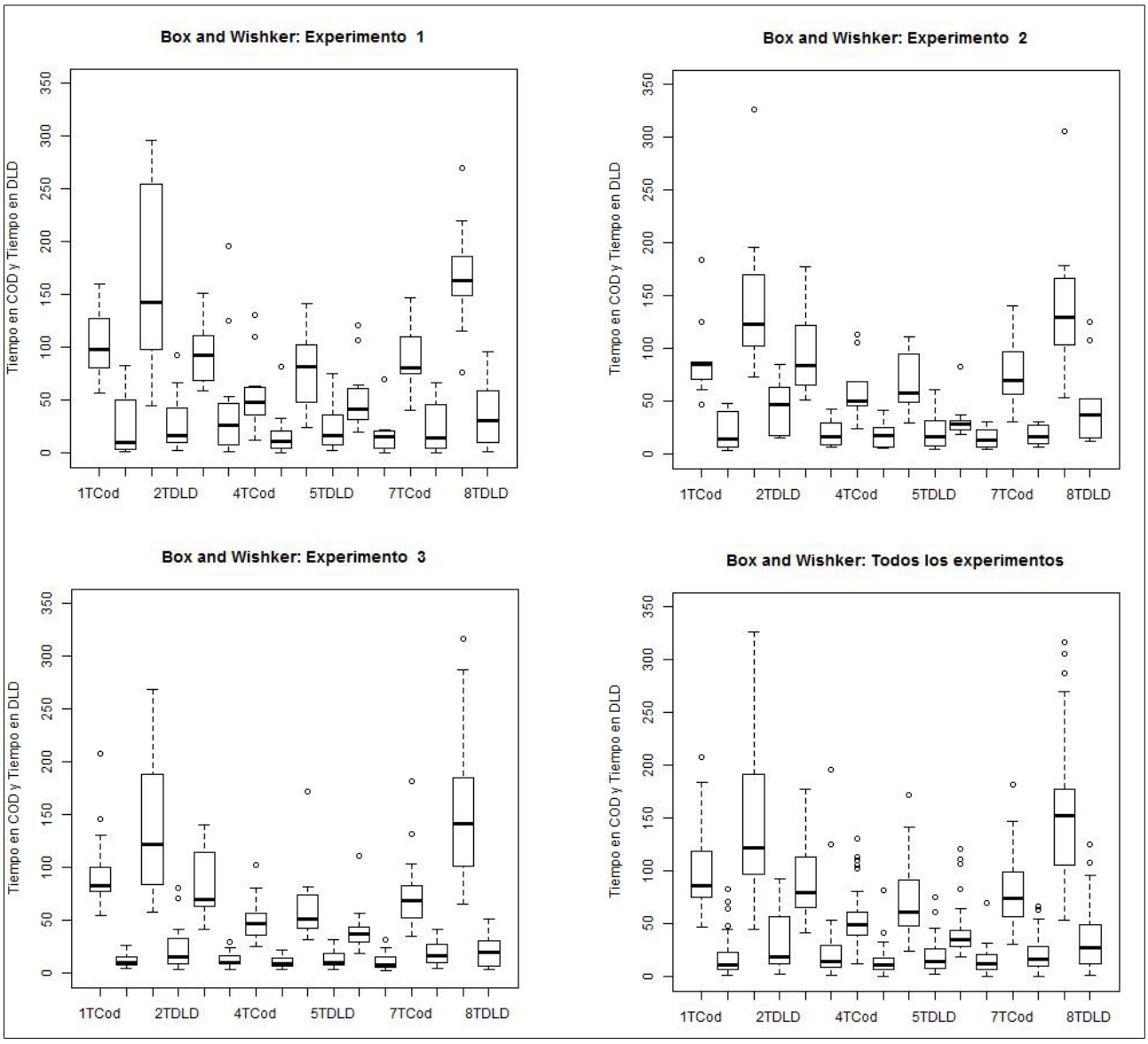

Figura 15 - *Box and Whisker para tiempo COD y tiempo DLD por programa*

Luego del estudio de la normalidad se realiza el test de Wilcoxon, el test se aplica para cada programa de cada experimento con N=2, 3, 4 y 5. A continuación se muestran los resultados obtenidos para N=3 y N=4. En los Cuadro 17 y Cuadro 18 se presentan los resultados de aplicar el test de Wilcoxon con hipótesis direccionales (una cola) en cada programa y la agrupación de todos los programas, tomando como muestras: TCod y N*TDLD, en cada uno. Las celdas contienen el *pvalue* calculado. Las que se encuentran pintadas de color gris indican que no es posible rechazar la hipótesis nula (*pvalue>0.05*). Dicho de otra forma no es posible rechazar que el tiempo de Codificación sea menor o igual que N veces el tiempo dedicado al Diseño. Las de color rojo representan los casos en que la hipótesis nula ha sido rechazada (*pvalue<=0.05*), o sea, que el tiempo que se dedica a Codificación en el programa es mayor que N veces el dedicado a Diseño.

N = 3								
TABLA TEST WILCOXON, DE EXPERIMENTO: 1								
1	2	3	4	5	6	7	8	TODOS
0,173261	0,003825	0,437665	0,104713	0,119658	0,318935	0,153911	0,024930	0,104713

TABLA TEST WILCOXON, DE EXPERIMENTO: 2								
1	2	3	4	5	6	7	8	TODOS
0,101311	0,439241	0,057064	0,253812	0,323231	0,898689	0,120561	0,222293	0,253812

TABLA TEST WILCOXON, DE EXPERIMENTO: 3								
1	2	3	4	5	6	7	8	TODOS
0,000737	0,001488	0,000737	0,006552	0,013854	0,124432	0,066479	0,009612	0,000936

TABLA TEST WILCOXON, DE TODOS LOS EXPERIMENTOS								
1	2	3	4	5	6	7	8	TODOS
0,000929	0,000497	0,002964	0,005828	0,012418	0,390335	0,012954	0,002819	0,001776

Cuadro 17 - Test de Wilcoxon para la relación entre TCod y 3*TDLD

El Cuadro 17 muestra el test aplicado a N=3, se observa que en el experimento 2 no es posible rechazar la H0 para ninguno de los programas. Por otro lado en el experimento 1 se puede ver que los únicos que rechazan la hipótesis nula son los programas 2 y 8. Al compararlo con el Cuadro 18 que muestra los resultados para N=4 se observa que en ambos casos no es posible rechazar la hipótesis nula. Por lo cual podría considerarse que el N para ambos programas sería cercano al 4. Se puede ver que en tres de los experimentos el programa 8 rechaza la hipótesis nula por lo cual, relacionando con el resultado de la sección anterior, se puede observar que la diferencia en el programa 8 entre ambos tiempos es mayor.

En cuanto al experimento 3 y a la consideración de todos los experimentos juntos se puede ver que, a excepción de uno o dos programas, el resto rechaza la hipótesis nula para N=3. Mientras que para N=4 en la aplicación a todos los experimentos la hipótesis nula no puede ser rechazada. En el experimento 3 es rechazada en menos programas que para N=3.

Se omite incluir los cuadros para N=2 y N=5, pero se observa que las mismas son coherentes con los cuadros mostrados debido a que para N=2 la hipótesis nula es rechazada en los mismos programas que se rechaza en N=3 más otros. Mientras que para N=5 se rechaza tan solo en dos programas en el experimento 3: programa 1 y programa 3. Esto es lógico ya que se rechaza menos que en N=4 y los que se rechazan también se hace en N=4.

N = 4								
TABLA TEST WILCOXON, DE EXPERIMENTO: 1								
1	2	3	4	5	6	7	8	TODOS
0,437665	0,078970	0,759912	0,291460	0,562335	0,880342	0,593027	0,216384	0,318935

TABLA TEST WILCOXON, DE EXPERIMENTO: 2								
1	2	3	4	5	6	7	8	TODOS
0,287531	0,777707	0,222293	0,676769	0,712469	0,989088	0,479677	0,639362	0,639362

TABLA TEST WILCOXON, DE EXPERIMENTO: 3								
1	2	3	4	5	6	7	8	TODOS
0,001183	0,027312	0,002886	0,098025	0,110665	0,486063	0,458256	0,037368	0,023200

TABLA TEST WILCOXON, DE TODOS LOS EXPERIMENTOS								
1	2	3	4	5	6	7	8	TODOS
0,018756	0,106596	0,095042	0,283231	0,415691	0,974359	0,409315	0,095042	0,109642

Cuadro 18 - Test de Wilcoxon para la relación entre TCod y 4*TDLD

Resumen

Finalmente lo que se observa según los rechazos de hipótesis nula y los no rechazos, es que para el caso del experimento 1, el N que relaciona ambos tiempos para la mayoría de los programas sería N=3. En el test para N=2, la hipótesis nula ha sido rechazada en todos los programas excepto en el 3. En cuanto al experimento 2, el N tomaría el valor 3 para la mayoría de los programas aunque, por la poca variación en al-

gunos programas, podría pensarse que estaría entre 3 y 4. Por último en el experimento 3, el N estaría entre 3, 4 y 5 ya que están divididos los programas que rechazan y aquellos que no pueden rechazar las hipótesis para N=3 y N=5. A su vez, para N=5 hay dos programas que la continúan rechazando. En cuanto al resultado obtenido para todos los experimentos es posible afirmar que el N se encuentra entre 4 y 5, más cercano al 4. Esto se debe a que en el caso de N=3 la hipótesis nula es rechazada en casi todos los programas y por otro lado que la diferencia entre los *pvalue* para N=4 y N=5 son bastante amplios.

En los cursos del PSP, el valor propuesto se da cuando N=1. Como se observa en el caso de los experimentos el N es mayor, ya que se encuentra entre 3 y 4.

Esto refleja que los estudiantes normalmente dedican tres o cuatro veces menos de tiempo a Diseño que a Codificación. Probablemente, esto los lleva a desarrollar programas mal diseñados y con una alta densidad de defectos.

Capítulo 6 - Conclusiones y trabajo a futuro

En este capítulo se presentan las conclusiones y los trabajos a futuro.

6.1 - Conclusiones

En el mercado actual, los clientes tienen más exigencias sobre la calidad de los productos. Esto presenta un gran desafío para la industria del software, la cual debe producir software de calidad en el menor tiempo y al más bajo costo.

Dentro de las líneas de investigación del Grupo de Ingeniería de Software (Gris) se encuentra el estudio de procesos de desarrollo de software y su mejora. Uno de los procesos bajo estudio es el *PSP*.

El *Personal Software Process (PSP)* es un proceso que busca ayudar a gestionar y mejorar la forma en que el individuo realiza su trabajo, o sea su performance, para con ello crear productos de software de mayor calidad. En el *PSP* se propone que para gestionar, y así mejorar la performance, no solo se debe medir la calidad sobre el producto desarrollado, sino que se debe medir la calidad sobre el propio proceso.

Este trabajo consta de dos objetivos principales. El primer objetivo consiste en crear una solución que dé soporte a la integración de datos que se recolectaron de forma independiente durante la aplicación del *PSP*, con la herramienta *PSP Student Workbook.* A su vez, validar su funcionamiento con la migración de un conjunto de datos. El segundo objetivo, consiste en realizar un análisis estadístico sobre datos integrados utilizando la solución creada, validando que la integración realizada sea de utilidad. Este último, además consta de dos subobjetivos. El primer subobjetivo, consiste en evaluar qué tan efectivas son las fases de Revisión para remover defectos y cuál es el costo asociado. El segundo subobjetivo, consiste en analizar si la mejora en la performance del estudiante en los programas desarrollados aplicando el PSP, es producto de la aplicación del *PSP* o de la programación repetitiva.

Se generó un proceso de integración, que permite migrar las bases de datos individuales en una única base de datos integrada para facilitar el acceso a los datos. El proceso está compuesto por un conjunto de *scripts* que generan la estructura de la base de datos que permite integrar los datos, un gestor de base de datos MySQL y el desarrollo de una herramienta Java, AllPSPStudentData_v2. Esta, tiene como función principal migrar los datos contenidos en las bases individuales a una única base de datos integrada con motor MySQL. A su vez, por medio de la misma, se pueden ejecutar *scripts* para cargar datos en la base de datos integrada. Para verificar la correctitud en la migración, se realizó una comparación entre los datos de las bases individuales y los datos cargados en la base integrada, y se replicaron resultados que se encuentran en el trabajo *Quality is Free, Personal Reviews Improve Software Quality at No Cost* [3].

Para validar que los datos que migra la herramienta son de utilidad para realizar análisis estadísticos, se realizó un estudio con dos conjuntos de datos de forma independiente. Un conjunto proviene de cursos del *PSP* y el otro de experimentos controlados.

Como resultado del análisis en torno al primer subobjetivo del análisis estadístico realizado, observando la inyección y remoción de defectos, donde se utilizan los datos provenientes de los cursos, surge que las fases en las que se inyectan más defectos, son en la fase de Diseño y Codificación. Mientras que en las que se remueven más son en las Revisiones. Esto nos da la pauta de cuán efectivas son las mismas para encontrar los defectos. Principalmente en encontrar defectos inyectados en la fase que es objeto revisar. En cuanto al costo promedio de remover un defecto en una determinada fase, se observa que exceptuando la fase de Compilación, el costo es muy similar. De los resultados obtenidos, se concluye que las Revisiones resultan más beneficiosas que las Pruebas, ya que es donde se encuentra la mayor cantidad de defectos a un costo similar al de las Pruebas.

El análisis estadístico realizado en el marco segundo subobjetivo, donde se evalúa si la mejora en la performance del estudiante se debe a la programación repetitiva, se infiere el mismo resultado obtenido en *Demostrating the Impact of the PSP on Software Quality and Effort: Eliminating the Programming Learning Effect* [4]. No se detecta una mejoría relevante al no aplicar el proceso de forma completa (en los experi-

mentos). Es más, en algunos casos se genera un empeoramiento y en otros no hay diferencia. Esto se concluye luego de analizar las dos medidas definidas para evaluar la calidad del trabajo. Las medidas son la densidad de defectos en las Pruebas unitarias (UT) y por el otro el ratio TCod/TDLD. La primera medida se aplica sobre el producto y la segunda sobre el proceso. Al estudiar la densidad de defectos en UT, no se detectan mejoras estadísticamente significativas. En los casos que se detecta una mejoría, se puede deber a que los estudiantes no están habituados a medir su trabajo y el hecho de estar midiendo y registrando los errores que cometen, ayuda a que los asimilen. De todas formas estas mejorías no son continuas, como las que surgen aplicando el proceso [5].

Por otro lado, al estudiar la evolución del ratio TCod/TDLD, a lo largo del proceso, nuevamente se detecta que las diferencias entre un programa y otro no son estadísticamente significativas. Estos últimos resultados muestran que si el estudiante no tiene un proceso que se lo indique, por sí solo, no considera mejorar factores que inciden en la calidad del producto, del proceso, y por consiguiente su performance. Los resultados obtenidos en el estudio del ratio TCod/TDLD, donde se detectó que la mediana del ratio para cada programa no era cercana al valor sugerido, motivaron a realizar un estudio adicional. El mismo consistió en observar el poco tiempo que dedican los estudiantes a diseñar en comparación con el que dedican a codificar. El estudio consistió en observar cómo se relacionan los tiempos invertidos en cada una de estas fases para cada programa de forma independiente. Se estudió la proporción N, definida como: TCodi <= N*TDLDi (siendo i el numero de programa) y N=2,3,4 y 5. De este surge que el N se ubica entre 3 y 4, mientras que de resultados obtenidos de cursos del *PSP* se propone el valor N=1 [2]. Esto refleja que los estudiantes normalmente dedican tres o cuatro veces menos de tiempo a Diseño que a Codificación. Probablemente, esto los lleva a desarrollar programas mal diseñados y con una alta densidad de defectos.

6.2 - Trabajo a futuro

Relacionado con la **solución proporcionada para la integración,** sería interesante realizar modificaciones a la herramienta para darle mayor robustez y que pase de ser una herramienta que “migra datos de una base a otra” a ser una “herramienta de análisis”. La estructura actual de la base soporta la migración de todos los datos recolectados, por lo que sería importante la migración de todos los datos recolectados en las bases individuales a la base integrada.

Otra mejora sería tener la opción de crear la estructura de la base de datos por medio de la misma.

La integración con el software estadístico R. De esta forma se puede automatizar la obtención de algunos reportes “estándar” y, por otro lado, que no sea necesario utilizar otra herramienta aparte para el análisis.

En cuanto al **análisis estadístico** podría resultar interesante realizar los mismos estudios con un volumen mayor de datos. A su vez, antes de realizar el análisis estadístico, efectuar un estudio de calidad de datos y posteriormente una limpieza de los mismos, principalmente en los datos provenientes de los cursos que fue donde se detecta la mayor incertidumbre en la certeza de los mismos y el descarte realizado fue más “artesanal”.

Anexo 1 - Estructuras de las bases Microsoft Access y MySQL

En la tabla presentada a continuación se muestra las equivalencias entre las tablas presentes en la base de datos origen Access de cada estudiante y la destino MySQL. Como se podrá ver en general, la diferencia se da en el agregado de claves primarias y foráneas que sirvan para relacionar y restringir los datos contenidos en las tablas al integrarlas.

TABLA EN ACCESS	TABLA EN MYSQL	DIFERENCIA MAS RELEVANTES
	rloctypestandard	
DefectType	rdefecttype	
LOGDDetail	rlogddetail	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
LOGTDetail	rlogtdetail	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
Parts	rparts	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
PartTypeStandard	rparttypestandard	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
PhaseData	rphasedata	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
Phases	rphases	
PIPs	rpips	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
Processes	rprocesses	Se agregan FK
ProcessPhase	rprocessphase	
ProcessTypes	rprocesstypes	
ProgramSize	rprogramsize	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
Projects	rprojects	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
PSPAssgtData	rpspassgtdata	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
ScheduleWeeks	rscheduleweeks	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
SETAdd	rsetadd	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
SETBase	rsetbase	Se agrega a la PK que identifica el usuario en la tabla rusers
SETReuse	rsetreuse	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
SizeMeasure	rsizemeasure	
Tasks	rtasks	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
TaskSchedulePlans	rtaskscheduleplans	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
TestReports	rtestreports	Se agrega a la PK que identifica el usuario en la tabla rusers, se agregan FK
UserProfiles	ruserprofiles	Se agrega a la PK que identifica el usuario en la tabla rusers
Users	rusers	Se agrega a la PK que identifica el usuario

Anexo 2 – Detalle de las tablas MySQL de AllPSPStudentData_v2 y los cambios realizados en la estructura de la "herramienta Genexus"

A continuación se visualizan los datos básicos de cada una de las tablas. También se muestran los cambios realizados en comparación con la estructura inicial, los mismos se encuentran marcados con *.

Tabla: rlogddetail	
Clave primaria (PK):	
RuserUniqueID* , RdefectLogEntryID	
Claves Foraneas (FK) *:	
fk_RLOGDDetailDefectTypeID (RLOGDDetail-DefectTypeID)	rdefecttype.(RDefectTypeDefectTypeID)
fk_RLOGDDetailProjectID (RUserUniqueID, RLOGDDetailProjectID)	rprojects.(RUserUniqueID, RProjectsProjectID)
fk_RphaseInjectedID (RPhaseInjectedID)	rphases.(RPhasesPhaseID)
fk_RphaseRemovedID (RPhaseRemovedID)	rphases.(RPhasesPhaseID)
fk_RuserUniqueID (RUserUniqueID)	rusers.(RUserUniqueID)
Tipo de datos*:	
RLOGTDetailProjectID, en vez de tener el valor *null* por defecto paso a no aceptar el valor *null*.	

Tabla: rlogtdetail	
Clave primaria (PK):	
RuserUniqueID* , RTimeLogEntryID	
Claves Foraneas (FK) *:	
fk_RLOGTDetailPhaseID (RLOGTDetailPha-seID)	rphases.(RPhasesPhaseID)
fk_RLOGTDetailProjectID (RUserUniqueID, RLOGDDetailProjectID)	rprojects.(RUserUniqueID, RProjectsProjectID)
fk_RUserUniqueID2 (RUserUniqueID)	rusers.(RUserUniqueID)
Tipo de datos*:	
RLOGTDetailProjectID, en vez de tener el valor *null* por defecto paso a no aceptar el valor *null*.	

Tabla: rparts	
Clave primaria (PK):	
RuserUniqueID* , RPartsPartID	
Claves Foraneas (FK) *:	
fk_RpartsLOCTypeID (RPartsLOCTypeID)	rloctypestandard.RLOCTypeStandardLOCTypeID
fk_RpartsPartTypeStandardID (RPartsPartTy-peStandardID)	rparttypestandard.RPartTypeStandardPartTypeStand
fk_RpartsProjectID (RUserUniqueID, RPartsProjectID)	rprojects.(RUserUniqueID, RProjectsProjectID)
fk_RpartsSizeMeasureID (RPartsSizeMeasu-reID)	rsizemeasure.RSizeMeasureSizeMeasureID
fk_RUserUniqueID3 (RUserUniqueID)	rusers.RUserUniqueID

Tabla: rparttypestandard	
Clave primaria (PK):	
RPartTypeStandardPartTypeStand	
Claves Foraneas (FK) *:	
fk_RpartTypeStandardSizeMeasureID (RPartTypeStandardSizeMeasureID)	rsizemeasure.RSizeMeasureSizeMeasureID

Tabla: rphasedata	
Clave primaria (PK):	
RuserUniqueID* , RPhaseDataPhaseDataID	
Claves Foraneas (FK) *:	
fk_RPhaseDataPhaseID (RPhaseDataPhaseID)	rphases.RPhasesPhaseID
fk_RPhaseDataProjectID (RUserUniqueID, RPhaseDataProjectID)	rprojects.(RUserUniqueID, RProjectsProjectID)
fk_RPhaseDataSymbol (RPhaseDataSymbol)	rphases.RPhasesSymbol
fk_RUserUniqueID4 (RUserUniqueID)	rusers.RUserUniqueID
Tipo de datos*:	
RPhaseDataSymbol, en vez de tener el valor *null* por defecto paso a no aceptar el valor *null*.	

Tabla: rphases	
Clave primaria (PK):	
RPhasesPhaseID	
Claves Foraneas (FK) *:	
fk_RparentPhaseID (RParentPhaseID)	rphases.RPhasesPhaseID
Tipo de datos*:	
RPhasesSymbol, se modifica el tamaño del tipo varchar, anteriormente era varchar(20) y se cambio por varchar(50).	

Tabla: rpips	
Clave primaria (PK):	
RuserUniqueID* , RPIPID	
Claves Foraneas (FK) *:	
fk_RPIPsProjectID (RUserUniqueID, RPIPsProjectID)	rprojects.(RUserUniqueID, RProjectsProjectID)
fk_RUserUniqueID6: (RUserUniqueID)	rusers.RUserUniqueID

Tabla: rprocesses	
Clave primaria (PK):	
RProcessesProcessID	
Claves Foraneas (FK) *:	
fk_RProcessesProcessTypeID: (RProcessesProcessTypeID)	rprocesstypes.RProcessTypesProcessTypeID
fk_RparentProcessID (RParentProcessID)	Rprocesses.RProcessesProcessID

Tabla: rprocessphase	
Clave primaria (PK):	
RProcessPhaseID	
Claves Foraneas (FK) *:	
fk_RProcessPhasePhaseID (RProcessPhase-PhaseID)	rphases.RPhasesPhaseID
fk_RProcessPhaseProcessID (RProcessPha-seProcessID)	rprocesses.RProcessesProcessID

Tabla: rprogramsize	
Clave primaria (PK):	
RuserUniqueID* , RProgramSizeProgramSizeID	
Claves Foraneas (FK) *:	
fk_RProgramSizeProjectID (RUserUniqueID, RProgramSizeProjectID)	rprojects.(RUserUniqueID, RProgramSizeProjectID)
fk_RProgramSizeSizeMeasureiD (RProgram-SizeSizeMeasureiD)	rsizemeasure.RSizeMeasureSizeMeasureID
fk_RUserUniqueID7 (RUserUniqueID)	rusers.RUserUniqueID

Tabla: rprojects	
Clave primaria (PK):	
RuserUniqueID* , RProjectsProjectID	
Claves Foraneas (FK) *:	
fk_RprojectsProcessID (RProjectsProcessID)	rprocesses.RProcessesProcessID
fk_RUserUniqueID8 (RUserUniqueID)	rusers.RUserUniqueID

Tabla: rpspassgtdata	
Clave primaria (PK):	
RuserUniqueID* , RPSPAssgtDataID	
Claves Foraneas (FK) *:	
fk_RPSPAssgtDataProjectID (RUserUniqueID, RPSPAssgtDataProjectID)	rprojects.(RUserUniqueID,RProjectsProjectID)
fk_RUserUniqueID9 (RUserUniqueID)	rusers.RUserUniqueID

Tabla: rscheduleweeks	
Clave primaria (PK):	
RuserUniqueID* , RScheduleWeekID	
Claves Foraneas (FK) *:	
fk_RscheduleWeeksProjectID (RUserUniqueID, RScheduleWeeksProjectID)	rprojects.(UserUniqueID, RProjectsProjectID)
fk_RScheduleWeeksTaskSchedulePlan (RU-serUniqueID,RScheduleWeeksTaskSchedulePlan)	rtaskscheduleplans.(RUserUniqueID, RTaskSchedulePlansTaskSchedule)
fk_RUserUniqueID10 (RUserUniqueID)	rusers.RUserUniqueID

Tabla: rsetadd	
Clave primaria (PK):	
RuserUniqueID* , RSETAddID	
Claves Foraneas (FK) *:	
fk_RSETAddLOCTypeID (RSETAddLOCTypeID)	rloctypestandard.RLOCTypeStandardLOCTypeID
fk_RSETAddPartID (RUserUniqueID, RSETAddPartID)	rparts.(RUserUniqueID, RPartsPartID)
fk_RSETAddPartTypeStandardID (RSETAddPartTypeStandardID)	rparttypestandard.RPartTypeStandardPartTypeStand
fk_RSETAddProgramSizeID (RUserUniqueID, RSETAddProgramSizeID)	rprogramsize.(RUserUniqueID, RProgramSizeProgramSizeID)
fk_RSETAddProjectID (RUserUniqueID, RSETAddProjectID)	Rprojects.(RUserUniqueID, RProjectsProjectID)
fk_RSETAddSizeMeasureID (RSETAddSizeMeasureID)	rsizemeasure.RSizeMeasureSizeMeasureID
fk_RUserUniqueID11 (RUserUniqueID)	rusers.RUserUniqueID

Tabla: rsetbase	
Clave primaria (PK):	
RuserUniqueID* , RSETBaseID	
Claves Foraneas (FK) *:	
fk_RSETBasePartID (RUserUniqueID, RSETBasePartID)	rparts.(RUserUniqueID, RPartsPartID)
fk_RSETBaseProgramSizeID (RUserUniqueID, RSETBaseProgramSizeID)	rprogramsize.(RUserUniqueID, RProgramSizeProgramSizeID)
fk_RSETBaseProjectID (RUserUniqueID, RSETBaseProjectID)	rprojects.(RUserUniqueID, RProjectsProjectID)
fk_RSETBaseSizeMeasureID (RSETBaseSizeMeasureID)	rsizemeasure.RSizeMeasureSizeMeasureID
fk_RUserUniqueID12 (RUserUniqueID)	rusers.RUserUniqueID

Tabla: rsetreuse	
Clave primaria (PK):	
RuserUniqueID* , RSETReuseID	
Claves Foraneas (FK) *:	
fk_RSETReusePartID (RUserUniqueID, RSETReusePartID)	rparts.(RUserUniqueID, RPartsPartID)
fk_RSETReuseProgramSizeID (RUserUniqueID, RSETReuseProgramSizeID)	rprogramsize.(RUserUniqueID, RProgramSizeProgramSizeID)
fk_RSETReuseProjectID (RUserUniqueID, RSETReuseProjectID)	rprojects.(RUserUniqueID, RProjectsProjectID)
fk_RSETReuseSizeMeasureID (RSETReuseSizeMeasureID)	rsizemeasure.RSizeMeasureSizeMeasureID
fk_RUserUniqueID13 (RUserUniqueID)	rusers.RUserUniqueID

Tabla: rtasks	
Clave primaria (PK):	
RuserUniqueID* , RTaskID	
Claves Foraneas (FK) *:	
fk_RTasksPhaseDataID (RUserUniqueID, RTasksPhaseDataID)	rphasedata.(RUserUniqueID, RPhaseDataPhaseDataID)
fk_RTasksPhaseID (RTasksPhaseDataID)	rphases.RPhasesPhaseID
fk_RTasksProjectID (RUserUniqueID, RTasksProjectID)	rprojects.(RUserUniqueID, RProjectsProjectID)
fk_RTasksTaskSchedulePlanID (RUserUniqueID, RTasksTaskSchedulePlanID)	rtaskscheduleplans.(RUserUniqueID, RTaskSchedulePlansTaskSchedule)
fk_RUserUniqueID14 (RUserUniqueID)	rusers.RUserUniqueID

Tabla: rtaskscheduleplans	
Clave primaria (PK):	
RuserUniqueID* , RTaskSchedulePlansTaskSchedule	
Claves Foraneas (FK) *:	
fk_RUserUniqueID15 (RUserUniqueID)	rusers.RUserUniqueID

Tabla: rtestreports	
Clave primaria (PK):	
RuserUniqueID* , RTestReportID	
Claves Foraneas (FK) *:	
fk_RTestReportsProjectID (RUserUniqueID, RTestReportsProjectID)	rprojects.(RUserUniqueID, RProjectsProjectID)
fk_RUserUniqueID16 (RUserUniqueID)	rusers.RUserUniqueID

Tabla: ruserprofiles	
Clave primaria (PK):	
RuserUniqueID* , RUserProfilesUserProfileID	
Claves Foraneas (FK) *:	
fk_RUserUniqueID17 (RUserUniqueID)	rusers.RUserUniqueID

Tabla: rusers
Clave primaria (PK):
RuserUniqueID*

Tabla: rdefecttype
Clave primaria (PK):
RDefectTypeDefectTypeID

Tabla: rloctypestandard
Clave primaria (PK):
RLOCTypeStandardLOCTypeID

Tabla: rprocesstypes
Clave primaria (PK):
RProcessTypesProcessTypeID

Tabla: rsizemeasure
Clave primaria (PK):
RSizeMeasureSizeMeasureID

Anexo 3 - Ejecución de la migración de datos desde cero

Se detalla el proceso que se debe realizar para migrar los datos desde la base Access a la base MySQL considerando que no se tiene creada la estructura de la base de datos. También se describe la plataforma sobre la cual se implementó.

Los pasos a seguir son:

1. Instalación y configuración de herramientas a utilizar
2. Creación de la estructura de la base de datos
3. Migración de datos

Instalación y configuración de herramientas

Para ejecutar las herramientas se utiliza un PC con un procesador i5 y 6 Gb de ram. El sistema operativo utilizado es Windows 7 de 64 bits.

Herramientas utilizadas

- Ide: Para la modificación, compilación y ejecución de la herramienta Java se utiliza Eclipse Indigo en su versión para web developers.
- Servidor de base de datos: MySQL 5.6.11.

Se debe ingresar en la página de MySQL http://www.mysql.com/downloads/ y descargar la distribución MySQL Enterprise Edition. de la página de Oracle. Luego para gestionar la base de datos se utiliza MySQL Workbench.

Generación de estructura

Mediante algún gestor de base de datos (en este caso se utiliza: MySQL Workbench) y utilizando la conexión creada para el servidor de base de datos se va a ejecutar el script: datosestudiantes.sql. Dicho *script* se encarga de crear toda la estructura de la base de datos: tablas, pk y fk.

La carga de la base de datos se realiza en dos instancias: en una primera, se cargan las tablas que tienen datos comunes a todos los participantes del proyecto. Generalmente estas tablas cumplen la función de "codigueras" para el resto de las tablas, por lo cual se generan claves foráneas que relacionarán el resto de las tablas con estas. El resto de las tablas son cargadas utilizando la herramienta Java que fue modificada para tal fin.

Carga mediante *scripts*

A continuación se detallan los *scripts* que deben ser ejecutados, utilizando algún gestor de base de datos. La ejecución es en el orden indicado para que no se produzcan violaciones de reglas de integridad.

Orden en que deben ser ejecutados los *scripts*:

1. RDefectType
2. RPhases
3. RSizeMeasure
4. RLOCTypeStandard
5. RProcessTypes
6. RProcesses
7. RProcessPhase
8. RpartTypeStandard

Carga mediante herramienta

Luego de ejecutar los *scripts* de carga la base de datos, esta queda pronta para poder ser accedida por la herramienta Java para la carga de los datos restantes.

Generación de tablas para análisis de datos (opcional):

Para analizar los datos cargados se crean algunas tablas de resumen, las cuales se generan y cargan mediante los siguientes scripts: TablaRes_StuPorDefPhaInjRem.sql y TablaRes_TotalDefects.sql.

Anexo 4 - Descripción y formato del archivo de *log*

La herramienta, cuando carga los datos, genera un archivo de *log* que contiene información del resultado de la migración de los datos.

Ejecución correcta

Si una base se migra sin inconvenientes, se genera en el *log* el cual contiene un par de líneas por cada estudiante procesado. Las mismas tienen el siguiente formato:

Student beeing processed "ubicación del archivo procesado"
generatedkey "identificador generado secuencialmente para el estudiante"

Ejecución con errores en la migración

Aparte de los renglones indicados anteriormente, para cada estudiante se agrega un detalle que resulte útil para analizar el motivo por el cual se genera dicho error. Para ello se genera un registro con el siguiente formato:

ERROR en "función en la cual se genera el error": "descripción corta del error"
Impresión de la sentencia que generó el error
DETALLE:
"descripción del detalle"

Nota: En el detalle se proporcionan datos que se procesan. Dentro de los datos, se encuentran: el identificador del estudiante (ID_ESTUD), qué programa se estaba cargando (PROJECTID), a qué fase pertenece el registro (PHASEID), etc.

Anexo 5 - Calidad de datos y problemas durante la integración

Discusión de inconsistencias encontradas en los datos

Al analizar el log generado se detecta que la mayoría de los errores se repetían en varios registros, la mayoría correspondían a violaciones de reglas de integridad.

Regla de integridad violada: rphasedata.RPhaseDataSymbol->rphases.RPhasesSymbol

La misma no es cumplida por unos 7 estudiantes, y se debe a que la tabla de fases, que tiene cargado el registro Access de dichos estudiantes, no coincide con la que se carga en la base MySQL. Como se menciona en secciones anteriores la tabla rphases es una tabla que se carga mediante un *script* asumiendo que todos los estudiantes tendrán definidas las mismas fases durante el curso. Sin embargo al analizar los datos se detecta que no era así para todos.

A modo de ejemplo, para clarificar lo expresado anteriormente se presenta uno de los casos. El caso corresponde al registro:
CLEANED-grading-psp2\2006-2H\Magellan-Oct2006\Program8 Assignment\PSPEng2v0.7normal_assm-pr-prog8-1\tmatthews_2.

A continuación se presentan las diferencias entre las fases que contiene dicho registro y la base MySQL:
Phases en registro access:

ID	SIMBOLO FASE	DESCRIPCION FASE
20	*Analyze*	*Analyze data*
21	*Write*	*Writing the report*
22	*Review*	*Review the report*

Phases en base MySQL:

ID	SIMBOLO FASE	DESCRIPCION FASE
20	*PSP workbook PIPs for process needs & Design and code review hecklists*	*Entry Criteria*
21	*Analysis of the data*	*Analysis of data*
22	*Layout of report*	*Planning*

Como podemos visualizar en la tabla mostrada a continuación (Cuadro 19) donde se visualizan los registros que no cumplen la regla, vemos que, excepto uno de los registros, el resto corresponden a una misma instancia del curso. Sin embargo, todos los estudiantes tienen diferentes fases definidas.

fk_RPhaseDataSymbol :datosestudiantes.rphasedata.RPhaseDataSymbol->rphases.RPhasesSymbol
CLEANED-grading-psp2\CLEANED-grading-psp2\2006-2H\Magellan-Oct 2006\Program 8 Assignment\PSPEng2v0.7normal_assm-pr-prog8-1\tmatthews_2
CLEANED-grading-psp2\CLEANED-grading-psp2\June 2006\PSPEng2\Oracle-Jun 2006\Program 8 Assignment\PSPEng2v0.7normal_assm-pr-prog8-1\hpark_1
CLEANED-grading-psp2\CLEANED-grading-psp2\June 2006\PSPEng2\Oracle-Jun 2006\Program 8 Assignment\PSPEng2v0.7normal_assm-pr-prog8-1\lxia_2
CLEANED-grading-psp2\CLEANED-grading-psp2\June 2006\PSPEng2\Oracle-Jun 2006\Program 8 Assignment\PSPEng2v0.7normal_assm-pr-prog8-1\rbairraj_2
CLEANED-grading-psp2\CLEANED-grading-psp2\June 2006\PSPEng2\Oracle-Jun 2006\Program 8 Assignment\PSPEng2v0.7normal_assm-pr-prog8-1\rkulkarn_1
CLEANED-grading-psp2\CLEANED-grading-psp2\June 2006\PSPEng2\Oracle-Jun 2006\Program 8 Assignment\PSPEng2v0.7normal_assm-pr-prog8-1\shanda_4
CLEANED-grading-psp2\CLEANED-grading-psp2\June 2006\PSPEng2\Oracle-Jun 2006\Program 8 Assignment\PSPEng2v0.7normal_assm-pr-prog8-1\ssahai_2

Cuadro 19 - REGISTROS QUE NO RESPETAN LA FK: fk_RPhaseDataSymbol

Sugerencia:

Teniendo en cuenta lo expresado anteriormente, podríamos en principio descartar dichas fases agregadas, ya que no es de interés evaluarlas porque no forman parte del proceso que queremos analizar. De to-

das formas queda sembrada una importante duda en cuanto a la confianza en el resto de la información contenida en los registros, ya que las fases no solo son distintas, sino que lo son entre estudiantes que realizaron un mismo curso.

Regla de integridad violada:

rprogramsize.RProgramSizeSizeMeasureiD->rsizemeasure.RSizeMeasureSizeMeasureID

Al igual que en el caso anterior los mismos 7 estudiantes presentan diferencias en otras de las tablas. En este caso corresponde a la tabla sizemesure. Cuando se carga la base MySQL se carga con el tipo de medida que se utiliza generalmente para medir el tamaño de un código (LOCs). Sin embargo dichos estudiantes agregan otras medidas.

A modo de ejemplo uno de los estudiantes tiene cargado como medidas:

ID	SIMBOLO	DESCRIPCION
1	LOC	*Count each LOC*
2	NUMSECTIONS	*Number of Paragraphs or Sections*
4	NUMCHART	*Number of Charts or Graphs*
5	LOT	*Number of Lines of Text*

Al revisar los casos en que se utilizaron las medidas que no corresponden a LOCs, se detecta que no son ingresadas dentro del marco de los 8 programas del curso sino que se registran en programas que no están dentro de los ejercicios del curso, en general corresponden a reportes.

Sugerencia:

En este caso se podrán descartar, ya que el análisis a realizar es dentro del marco de los programas definidos en curso. En el caso de optar por esta solución, no solo se tendrán que descartar dichos datos sino que también algunos registros de la tabla *projects*. Porque allí también figuran ingresados proyectos "extras" que no corresponden, como ser: "*Interim report*", "*final report*". Estos son distintos para cada estudiante y las medidas para el tamaño no son consideradas válidas.

Relacionado con lo mencionado anteriormente un punto importante que no se tuvo en cuenta al diseñar la base MySQL, es controlar la cantidad de proyectos ingresados por estudiante. Ya que muchos tienen ingresados proyectos más allá del programa 8. Sería interesante poder investigar el motivo por el cual se ingresaron y si fue parte del curso realizado por el estudiante. Aunque por el tipo de actividad también pudo haber sido registrados por los instructores.

Problemas con Access en algunos casos:

[Microsoft][Controlador ODBC Microsoft Access]Índice de descriptor no válido

Hay 4 registros de estudiantes que están fallando en la carga, el error se da cuando se tratan de obtener algunos de los campos Access. Se ha modificado el código intentando solucionarlo pero no se llega a una solución.

Sugerencia:

Para estos casos hay dos opciones a tomar debido a que no se encuentra la solución:

- No cargar las tablas de dichos registros
- Cargar los datos manualmente mediante *scripts*

Problemas de sintaxis en los insert en la base MySQL

Se presentaron inconvenientes con el caracter '\' ya que es interpretado como un caracter de escape y no como la representación de un caracter en sí.

Solución:

Se modifica el código para que cambie dicho caracter al identificarse su presencia.

Regla de integridad violada:

rprojects.RProjectsProcessID->rprocesses.RProcessesProcessID

La violación de esta regla se registra para un único estudiante, debido a que su registro tiene datos almacenados en la tabla processes que no coinciden con la rprocesses que se carga inicialmente en la base MySQL, agrega noveno proceso "*Report Process*".

Sugerencia:

En este caso se puede no tener en cuenta el proceso debido a que no hace referencia a ninguno de los procesos del PSP, aunque no se conoce el motivo de su inclusión.

Resoluciones de inconsistencias encontradas en los datos

Descripción
Tipo
Violación de regla de integridad
Detalle
Regla de integridad violada: rphasedata.RPhaseDataSymbol->rphases.RPhasesSymbol
Excepción lanzada
com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException
Error en el log
Cannot add or update a child row: a foreign key constraint fails (`datosestudiantes`.`rphasedata`, CONSTRAINT `fk_RPhaseDataSymbol` FOREIGN KEY (`RPhaseDataSymbol`) REFERENCES `rphases` (`RPhasesSymbol`) ON DELETE NO ACTION ON UPDATE NO ACTION)
Implementación de la solución
Dado que dicha violación no permite que se inserten los datos involucrados en la tabla rphases, se opta por captar los mismos e imprimirlos en el *log*. Por lo cual cuando se produzca el error se generan registros identificados como: ERROR en loadTableRPhasedata. Los mismos incluyen la siguiente información: ubicación del archivo que se está procesando, identificador del estudiante en la tabla rusers, identificador y nombre del proyecto asociado a la fase, identificador y símbolo de la fase así como también el proceso asociado al proyecto.

Descripción
Tipo
Violación de regla de integridad
Detalle
Regla de integridad violada: rprogramsize.RProgramSizeSizeMeasureiD->rsizemeasure.RSizeMeasureSizeMeasureID
Excepción lanzada
com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException
Error en el log
Cannot add or update a child row: a foreign key constraint fails (`datosestudiantes`.`rprogramsize`, CONSTRAINT `fk_RProgramSizeSizeMeasureiD` FOREIGN KEY (`RProgramSizeSizeMeasureiD`) REFERENCES `rsizemeasure` (`RsizeMeasureSizeMeasureID`)

Implementación de la solución
En este caso se procede de forma similar al anterior registrando dicho error en el *log*. El registro se identifica como: ERROR en loadTableRProgramsize e incluye: ubicación del archivo que se está procesando, identificador del estudiante en la tabla rusers, el identificador y nombre del proyecto asociado a la medida, la medida y el proceso asociado al proyecto.

Descripción
Tipo
Error generado por el *driver* de Access
Detalle
Se investigan los motivos del error y luego de varias pruebas se detecta que el error se genera porque en dichos casos la tabla projects contiene una columna menos. La columna que no se encuentra presente es: usersEnterQualityPlan.
Excepción lanzada
Problemas con Access en algunos casos: [Microsoft][Controlador ODBC Microsoft Access]Índice de descriptor no válido
Error en el log
ADVERTENCIA en loadTableRProjects: la tabla tiene 28 columnas y debería tener 29
Implementación de la solución
Dado que al generarse dicho error no se continúan cargando los registros del archivo, se procede a modificar el código para que contemple dicho caso y a su vez generar una advertencia en el *log*. Cuando se carga la tabla, primero se contabiliza cuantas columnas tiene, en el caso 28 el valor de dicha columna es cargado con el valor 2. Se toma dicho valor ya que la columna en caso de estar presente representa verdadero o falso con los valores 0 o 1. A su vez en el *log* se registra con una advertencia: "ADVERTENCIA en loadTableRProjects: la tabla tiene 28 columnas y debería tener 29".

Anexo 6 - Cálculo de la d de Cohen para la densidad de defectos en UT

d COHEN, DE EXPERIMENTO: 1							
programa	2	3	4	5	6	7	8
1	-0,388272	0,391124	0,396356	0,075947	0,391437	0,404851	0,553779
2		1,145202	1,147652	0,697047	1,145019	1,153911	1,386979
3			0,020805	-0,825012	0,001562	0,051035	0,633415
4				-0,824851	-0,019206	0,030057	0,578354
5					0,823965	0,831803	1,384087
6						0,049375	0,627279
7							0,513718

Cuadro 20 - Cálculo del d Cohen para la densidad de defectos en UT en el experimento 1

d COHEN, DE EXPERIMENTO: 2							
programa	2	3	4	5	6	7	8
1	-0,248927	0,906527	1,174893	0,140931	0,586252	0,088680	0,908623
2		0,812947	0,939146	0,326613	0,605935	0,272575	0,809490
3			0,176117	-0,574833	-0,415772	-0,501057	-0,016095
4				-0,738814	-0,679543	-0,623511	-0,199347
5					0,300738	-0,025443	0,569710
6						-0,273145	0,408798
7							0,495269

Cuadro 21 - Cálculo del d Cohen para la densidad de defectos en UT en el experimento 2

d COHEN, DE EXPERIMENTO: 3							
programa	2	3	4	5	6	7	8
1	-0,578257	0,872671	0,667320	-0,282101	0,294779	0,586519	0,881242
2		1,660599	1,375202	0,259039	1,023598	1,317410	1,637018
3			-0,202722	-1,146112	-0,921210	-0,402214	0,079549
4				-0,943183	-0,551625	-0,142649	0,247202
5					0,609846	0,875325	1,148060
6						0,437691	0,898096
7							0,429572

Cuadro 22 - Cálculo del d Cohen para la densidad de defectos en UT en el experimento 3

d COHEN, DE EXPERIMENTO: TODOS LOS EXPERIMENTOS							
programa	2	3	4	5	6	7	8
1	-0,385908	0,565146	0,555101	-0,022607	0,367946	0,355146	0,657690
2		1,152546	1,134908	0,448947	0,934786	0,870403	1,257992
3			-0,010874	-0,886722	-0,438756	-0,296300	0,211844
4				-0,861506	-0,408949	-0,281143	0,212588
5					0,586282	0,520695	1,034714
6						0,025645	0,659509
7							0,450170

Cuadro 23 - Cálculo del d Cohen para la densidad de defectos en UT en todos los experimentos

Anexo 7 – Cálculo de la d de Cohen para la tasa TCod/TDLD

d COHEN, DE EXPERIMENTO: 1							
programa	2	3	4	5	6	7	8
1	0,449960	0,355615	0,552122	0,738013	-0,095799	-0,355922	0,083737
2		-0,009641	0,199289	0,529244	-0,251623	-0,410801	-0,281726
3			0,137652	0,316079	-0,241316	-0,408223	-0,232256
4				0,257816	-0,291989	-0,425131	-0,376508
5					-0,345430	-0,443459	-0,523692
6						-0,305358	0,132381
7							0,369126

Cuadro 24 - Cálculo del d Cohen para la tasa TCod/TDLD en el experimento 1

d COHEN, DE EXPERIMENTO: 2							
programa	2	3	4	5	6	7	8
1	0,378140	0,089476	0,405994	0,479368	0,892914	0,439515	0,383221
2		-0,357229	0,009482	0,080481	0,648900	0,030383	-0,014760
3			0,398009	0,503974	1,103388	0,449199	0,367624
4				0,080037	0,761113	0,022869	-0,026685
5					0,850345	-0,064850	-0,107749
6						-0,895397	-0,766722
7							-0,051653

Cuadro 25 - Cálculo del d Cohen para la tasa TCod/TDLD en el experimento 2

d COHEN, DE EXPERIMENTO: 3							
programa	2	3	4	5	6	7	8
1	-0,171976	0,318712	1,081729	0,823973	1,208308	0,796075	-0,500811
2		0,366675	0,769360	0,665679	0,863145	0,668621	-0,351656
3			0,759138	0,525255	0,910993	0,518379	-0,623807
4				-0,149958	0,256734	-0,088094	-0,867659
5					0,348407	0,038730	-0,809676
6						-0,270583	-0,927790
7							-0,814511

Cuadro 26 - Cálculo del d Cohen para la tasa TCod/TDLD en el experimento 3

d COHEN, DE EXPERIMENTO: TODOS LOS EXPERIMENTOS							
programa	2	3	4	5	6	7	8
1	0,232194	0,254749	0,547689	0,634416	0,102618	-0,168646	-0,070475
2		0,040845	0,369932	0,477918	0,003270	-0,205835	-0,257782
3			0,289746	0,379958	-0,012747	-0,211304	-0,276941
4				0,103464	-0,117032	-0,248608	-0,492279
5					-0,143300	-0,257732	-0,550806
6						-0,196452	-0,136934
7							0,151994

Cuadro 27 - Cálculo del d Cohen para la tasa TCod/TDLD en todos los experimentos

Bibliografía

[1] W. Humphrey, The Personal Software Process,Tech. Rep. CMU/SEI-2000-TR-022 ,Software Engineering Institute, Carnegie Mellon University,Pittsburgh, Pa, USA,2000

[2] W. Humphrey, PSP: A Self-Improvement Process for Software Engineers,Book ,Addison-Wesley,USA,2005

[3] D. Vallespir, W. Nichols, Quality is Free, Personal Reviews Improve Software Quality at No Cost, Software Quality Professional, Vol. 18, Nro. 2,2016, A ser publicado

[4] D. Vallespir, F. Grazioli, L. Perez, S. Moreno, Demostrating the Impact of the PSP on Software Quality and Effort: Eliminating the Programming Learning Effect,Conference Proceedings ,TSP Symposium ,Dallas, Texas, USA,2013

[5] W. Hayes and J. Over, “The personal software process: an empirical study of the impact of PSP on individual engineers”,Tech. Rep. CMU/SEI-97-TR-001 ,Software Engineering Institute, Carnegie Mellon University,Pittsburgh, Pa, USA,1997

[6] D. Vallespir, W. Nichols, Analysis of design defects injection and removal in PSP,Conference Proceedings ,TSP Symposium 2011: A dedication to excellence,Atlanta, GA, USA,2011