Ingeniare. Revista Chilena de Ingeniería
ISSN: 0718-3291
facing@uta.cl
Universidad de Tarapacá
Chile

Salinas, Erick; Cerpa, Narciso; Rojas, Pablo
Arquitectura orientada a servicios para software de apoyo para el proceso personal de software
Ingeniare. Revista Chilena de Ingeniería, vol. 19, núm. 1, junio, 2011, pp. 40-52
Universidad de Tarapacá
Arica, Chile

Ingeniare. Revista chilena de ingeniería, vol. 19 Nº 1, 2011, pp. 40-52

Arquitectura orientada a servicios para software de apoyo para el proceso personal de software

A service oriented architecture for the implementation of the personal software process

Erick Salinas[1] Narciso Cerpa[1] Pablo Rojas[1]

Recibido 5 de mayo de 2010, aceptado 16 de marzo de 2011
Received: May 5, 2010 Accepted: March 16, 2011

RESUMEN

El presente trabajo describe una arquitectura orientada a servicios para un software que tiene como objetivo facilitar la implementación de un Proceso Personal de *Software* en un equipo de desarrollo u organización. Entre las características que posee este software y que son relevantes de mencionar están las de entregar extensibilidad e independencia, esto se ve reflejado en la facilidad para agregar nuevas herramientas al proceso de desarrollo de *software* integradas al Proceso Personal de *Software* con un máximo de independencia de sistemas operativos y lenguajes de programación.

El software implementado realiza la recolección de los datos necesarios para el Proceso Personal de *Software* casi completamente automática, considerando que el administrador solamente clasifica los errores que pueden ocurrir cuando se utiliza algún lenguaje de programación en particular, entre otras pequeñas tareas. Esta facilidad de uso hace que la implementación del Proceso Personal de *Software* se realice exitosamente con un bajo esfuerzo requerido por los integrantes del equipo de desarrollo.

Palabras clave: Proceso personal de software, arquitectura orientada a servicios, proceso de software, servicios Web, Java.

ABSTRACT

This work describes a service oriented architecture of a software application that facilitates the implementation of the Personal Software Process by a development team or an organization. Some of the characteristics of this software and which are important to mention are extensibility and technical environment independence. These characteristics facilitate the process of adding new tools to the software development process integrating them to the Personal Software Process independently of the operating systems and programming languages being used.

The implemented software undertakes the data collection necessary to the Personal Software Process almost automatically, since the administrator must only classify the errors that may occur when a particular programming language is used, among other small tasks. This ease of use approach helps to make the implementation of the Personal Software Process a success, requiring a low effort by the software development team members.

Keywords: Personal software process, service oriented architecture, software process, Web services, Java.

[1] Facultad de Ingeniería. Universidad de Talca. Merced 437. Curicó, Chile. E-mail: esalinasz@gmail.com; ncerpa@utalca.cl; pabrojas@utalca.cl

INTRODUCCIÓN

El desarrollo de *software* implica mucho más que escribir instrucciones de programación y ejecutarlas en un computador. Se requiere cumplir los requisitos del cliente a un costo y de acuerdo a una planificación preestablecida. Para tener éxito y obtener productos de calidad, los ingenieros de *software* deben regirse por un proceso de desarrollo de calidad [5,30-31].

Debido a que el costo total de desarrollo de *software* lo constituye en un 70% el equipo de desarrollo, se hace necesario mejorar las habilidades y hábitos de trabajo para que los ingenieros de *software* realicen de mejor manera las actividades del proceso [25].

Las métricas del proceso de desarrollo de *software* y del producto son una medida cuantitativa que permite tener una visión profunda de la eficacia del proceso y de los proyectos que se ejecutan utilizándolo como un marco de trabajo. La eficacia de dicho proceso se mide indirectamente, es decir, se extrae un conjunto de métricas con el objetivo de medir características de tareas específicas del proceso de ingeniería de *software*.

Dentro de este grupo de métricas algunas se pueden considerar como privadas para desarrolladores, las cuales se ajustan con el enfoque del Proceso Personal de *Software* (*Personal Software Process*^{SM} PSP) [23].

Watts Humphrey, consciente que la mejora del proceso de desarrollo de *software* puede y debe empezar en el nivel individual, comenzó en 1989 el desarrollo del Proceso Personal de *Software*, como producto de la inquietud de aplicar el *Modelo de madurez de capacidades* (CMM) a pequeños proyectos [16].

El Proceso Personal de *Software* nace como un acercamiento estructurado y disciplinado para el desarrollo de *software*, cuya efectividad en el ámbito académico e industrial ha sido documentada en numerosos reportes técnicos y artículos de revistas especializadas [12]. Este proceso proporciona a los ingenieros de *software* un conjunto de formularios, guías y estándares que les ayudan a estimar y planificar su trabajo, y que puede ser usado en muchos de los aspectos relacionados con éste [31].

El PSP requiere una recopilación y análisis de métricas con un elevado nivel de detalle, lo cual no es algo trivial. En cualquier proyecto real, el empleo de herramientas de apoyo al Proceso Personal de *Software* se convierte en un elemento muy importante a considerar [36].

Las herramientas de apoyo al PSP que actualmente se encuentran disponibles han evolucionado hasta encontrarse en una generación que es capaz de recolectar de forma automática ciertas métricas [17], aunque aún no es posible encontrar un *software* que cumpla con facilitar directamente la implementación del Proceso Personal de Software junto con la recolección de datos de forma totalmente automatizada.

En el presente documento se presenta una arquitectura de *software* que ofrece un marco estructural para la construcción de una herramienta de *software* que facilite el uso de las fases del Proceso Personal de *Software* por parte de los desarrolladores.

Los objetivos específicos de dicha herramienta son:

- Capturar la mayor cantidad de información necesaria de forma automática y con mínima participación del usuario.
- Ser independiente de cualquier entorno de desarrollo y permitir una variedad de lenguajes de programación, brindando la capacidad de extender su uso.
- Facilitar la recuperación de información y la generación de informes y estadísticas de forma personalizada.

A continuación se presenta el marco teórico que sustenta este trabajo, presentando una descripción del PSP y sus componentes básicos; además se muestra un conjunto de herramientas que implementan PSP con sus respectivas características, las que dan los principales requisitos a este sistema.

Posteriormente se describe la arquitectura (orientada a servicios) del sistema mencionando, el funcionamiento tanto del servidor como de los distintos clientes y plug-ins o add-ons para los distintos IDE. Por último, se presentan las principales funcionalidades del sistema y la validación de esta arquitectura, para finalizar con las conclusiones de este trabajo.

MARCO TEÓRICO

El uso de métricas es una característica importante de todas las disciplinas de ingeniería. En un marco de trabajo de ingeniería, las métricas se refieren a estándares de las medidas usados para cuantificar aspectos específicos de un proceso, de un producto o de un proyecto de ingeniería [23]. La recolección de medidas es el primer paso en el orden para conocer cómo se controla y mejora el proceso de desarrollo de software [24].

Las métricas que se extraen del proceso, como por ejemplo medición de tiempo y esfuerzo, tienen usos privados y públicos [22]. Es por esto que algunas de estas métricas recopiladas deberían ser privadas para el desarrollador y servir como indicador sólo para él. En esta filosofía de datos de procesos privados centra su enfoque el PSP, reconociendo por tanto que la mejora del proceso de desarrollo de software puede y debe comenzar en el nivel individual [23].

El Proceso Personal de *Software*

El PSP entrega a los ingenieros un marco de referencia de disciplina personal para mejorar su trabajo y realizarlo con alta calidad [31], con el propósito de ayudarlos a aprender y practicar aquellos métodos para producir *software* que son más efectivos para ellos [29]. Entendiendo como principio fundamental que con un proceso de calidad los productos derivados de éste serán también de calidad [31].

El PSP consiste en un conjunto de métodos, formularios y guías (*scripts*) que muestran a los desarrolladores de *software*, la forma de planificar, medir y administrar su trabajo [31]. Este proceso puede ser introducido mediante un curso y su libro guía [28]. Está diseñado para ser usado con cualquier lenguaje de programación o metodología de diseño y puede ser utilizado en muchos aspectos del desarrollo de *software* [31].

Cuando los datos históricos del PSP son recolectados y mantenidos el desarrollador será capaz de comprender en qué gasta su tiempo, dónde y por qué introduce defectos y cuánto le toma encontrarlos, corregirlos y prevenirlos [29].

Se debe tener presente que el PSP no resuelve los problemas que tienen los estudiantes y profesionales en el desarrollo de *software*, pero los puede ayudar y guiar bastante en el establecimiento de una práctica disciplinada que puede ser analizada y mejorada [27].

El desarrollo de PSP está basado en parte en los principios de manejo de calidad de W. Edwards Deming y Joseph M. Juran cuyos objetivos son los de analizar y mejorar el trabajo personal [32].

Después del desarrollo del CMM, Watts S. Humphrey decidió aplicar estos principios para escribir pequeños programas, debido en parte a que mucha gente preguntaba cómo aplicarlo a organizaciones pequeñas o a trabajo de equipos pequeños, comprobando entonces que los principios de Deming y Juran eran totalmente aplicables al trabajo de los ingenieros de *software* de manera individual como en equipo [31].

El diseño del PSP se basa en los siguientes principios de planeación y de calidad [28]:

- Cada ingeniero es esencialmente diferente; para ser más precisos, los ingenieros deben planificar su trabajo y basar sus planes en sus propios datos personales.
- Para mejorar constantemente su desempeño, los ingenieros deben utilizar personalmente procesos bien definidos y medidos.
- Para desarrollar productos de calidad, los ingenieros deben sentirse personalmente comprometidos con la calidad de sus productos.
- Cuesta menos encontrar y arreglar errores en la etapa inicial del proyecto que encontrarlos en las etapas subsecuentes.
- Es más eficiente prevenir defectos que encontrarlos y arreglarlos.
- La manera correcta de hacer las cosas es siempre la manera más rápida y más barata de hacer un trabajo.

La estructura del PSP comienza a partir de una declaración de los requisitos, luego el primer paso del proceso es el de planificación [31], en donde se genera a partir de la respectiva guía, el método de trabajo y el plan para el registro de los datos. Mientras los desarrolladores cumplen la guía (*script*), registran sus datos, como por ejemplo el tiempo utilizado en esa etapa y los datos de defectos. Al

concluir su trabajo, durante la etapa de *post mórtem* completan, con la información extraída del proceso, el Formulario de Resumen del Plan. El PSP contiene cuatro elementos básicos:

Guías (scripts): son una descripción de nivel-experto para guiar el proceso. Contienen el propósito u objetivo del proceso, el criterio de entrada, cualquier guía general, consideraciones de uso o limitaciones, fases o pasos a efectuar, medidas de proceso, criterios de calidad y condiciones de finalización.

Formularios: proveen un conveniente y consistente marco de trabajo para recolectar y retener datos. Especifican los datos requeridos y donde estos deben ser registrados.

Medidas: son las medidas de cuantificación del proceso y el producto.

Estándares: entregan una precisa y consistente definición que guía el trabajo, junto con la recopilación y uso de datos. Permiten aplicar mediciones uniformes a través de múltiples proyectos y comparaciones entre unos y otros.

El PSP introduce los conceptos de proceso en una serie de pasos. Cada nivel, de un total de seis, incluye todos los elementos del paso anterior con uno o dos adicionales.

Las actividades y métodos utilizados dentro del PSP son los siguientes:

Recolección de datos: Las mediciones del PSP están definidas por el paradigma *Goal-Question-Metric*. Este es el tiempo en que los desarrolladores gastan en cada fase del proceso, los defectos introducidos y encontrados en cada una de ellas y el tamaño en líneas de código fuente (SLOC, *Source Lines Of Code*) del producto desarrollado. Todos estos datos son recolectados en cada una de las fases y resumidos con la finalización del proyecto.

Estimación: Un importante aspecto de la ingeniería de *software* es la habilidad de hacer estimaciones del proceso y atributos del producto. El esfuerzo de estimación es en general tedioso, pero existen varios métodos para estimar en la literatura como por ejemplo el modelo COCOMO y métodos basados en el enfoque de Albretch de puntos de función [13]. La técnica utilizada en PSP para realizar estimaciones de tamaño se denomina PROBE, acrónimo de *PROxy Based Estimating*, empleando *proxys* u objetos como base, para luego aplicar técnicas estadísticas y ajustar la estimación probable del tamaño a partir de estimaciones pasadas [21]. El método PROBE del Proceso Personal de *Software*, introducido en la etapa PSP1, es posiblemente el más extensamente usado enfoque analítico para estimar el esfuerzo personal, pero en la práctica no precisamente es el más certero [18].

Manejo de defectos: En el PSP la calidad del producto es medida en términos de "defectos", en donde un defecto es cualquier cosa dentro del programa que deba ser cambiada para permitir que sea propiamente desarrollado, mejorado o usado. Los defectos pueden estar en el código, diseño, requisitos, especificaciones o en la documentación; y existe un estándar que los define por categorías [12]. En el PSP todos los defectos son contados, con el objetivo de encontrarlos y eliminarlos lo antes posible. Los ingenieros aprenden a seguir y analizar los defectos, recolectando los datos de las fases en las que fueron introducidos y removidos, describiendo los tipos de defectos y el tiempo de corrección.

Administración del Rendimiento: El rendimiento es la principal medida de calidad del PSP. El rendimiento total del proceso es el porcentaje de defectos encontrados y corregidos antes que los desarrolladores comiencen a compilar y probar sus programas. Aunque la calidad del *software* incluye más que defectos, el PSP se enfoca en la detección y prevención de errores, puesto que encontrar y corregir éstos absorbe más tiempo y costo en el desarrollo.

Controlar el costo de la calidad: Para gestionar la calidad del proceso, el PSP usa tres mediciones de costo de calidad:

- *Costos de estimación*: tiempo del desarrollo utilizado en el diseño y revisión del código,
- *Costos de fallas*: tiempo utilizado en compilar y probar; y
- *Costos de prevención*: tiempo utilizado en la prevención de defectos antes que éstos ocurran.

Otra medida de costo de calidad es la conocida como razón de estimación de fallas (appraisal-to-failure-

ratio) A/FR, la cual puede ser calculada dividiendo el costo de estimación por el costo de fallas. La medición de A/FR es relativa al esfuerzo utilizado en remover tempranamente un defecto, con el objetivo de mejorar el rendimiento y es considerado junto al porcentaje de defectos removidos antes de la primera compilación, como uno de los factores críticos que afectan al PSP [35].

Herramientas de apoyo para la implementación de PSP

Existen iniciativas satisfactorias en el plano académico, con sus respectivos métodos, para enseñar un proceso de *software* en cursos básicos utilizando el PSP [37], en que se reconoce lo prematuro que es en estos casos introducir completamente sus conceptos [8, 19, 26]. También se pueden encontrar experiencias positivas aplicando PSP, aunque con algunas adaptaciones, en otros cursos como por ejemplo, introducción a base de datos [34].

Estudios han reportado excelentes resultados en la industria con la adopción de PSP [6], coincidiendo con publicaciones del SEI (*Software Engineering Institute*) [33], como el de Pat Ferguson [16], del mismo modo se pueden encontrar otros estudios con experiencias positivas en el plano académico [3-4].

Sin embargo, también se encuentran publicaciones que reportan una pobre adopción en la industria [14] como también en lo académico [11]. Las principales razones identificadas en la pobre adopción del Proceso Personal de *Software* son: el largo tiempo de entrenamiento de 150 a 200 horas [14] y la dificultad de la recolección de datos, que a su vez pueden conducir a problemas con la calidad de estos [1].

Según expertos [2] que utilizan y enseñan el PSP en su forma original, éste carece de un adecuado conjunto de herramientas. Completar los documentos de trabajo a mano implica mucho esfuerzo y existe la probabilidad de ingresar errores significativos en la recolección y posterior análisis de los datos, independientemente del potencial que posea la información.

En carreras universitarias en donde se enseña el PSP, los estudiantes solo utilizan éste en el curso donde se impartió. Ellos no lo continúan utilizando en sus cursos posteriores, confirmando el hecho de que cuando trabajan solos la motivación por ocupar PSP es nula o muy débil [9-10]. Si se les pregunta el por qué no lo utilizan, estos siempre responden quejándose por la dificultad del registro de los datos [20].

Aunque el Proceso Personal de *Software* puede ser probado como una técnica útil, su resultado está subordinado a la disponibilidad de los datos [7].

Se recomienda que se utilice una herramienta que soporte PSP [9], no solamente como ayuda, sino para obtener un alto grado de análisis de la calidad de los datos, aunque la utilización de ésta no será una "bala de plata" que resuelva todos los problemas [2].

Las herramientas que soportan PSP pueden dividirse en tres generaciones [17]. En la primera generación los desarrolladores imprimían los formularios y los completaban manualmente junto con los registros de tamaño, esfuerzo y defectos. Adicionalmente utilizaban formularios de datos para estimar el proyecto y asegurar la calidad. Este enfoque presenta un substancial gasto de tiempo para completar los formularios.

Durante la segunda generación se utilizan herramientas automatizadas para apoyar PSP como por ejemplo *LEAP* [41], *PSP Studio* [39], *PSP Dashboard* [40] *y PSP Tool* [1]. Estas herramientas tienen el mismo enfoque de interacción con el usuario. Muestran los campos de entrada donde registran el esfuerzo, tamaño y la información de defectos. También presentan varias herramientas de análisis si son requeridas por el usuario, alivianando su trabajo. Debido a que esta perspectiva no elimina la continua necesidad de cambio de contexto entre el desarrollo del producto y el registro del proceso, no fue ampliamente adoptada.

Finalmente se presenta una tercera generación en donde se busca recolectar el mayor grado de información posible de forma automática y discreta. El primer representante fue el desarrollo de *Hackystat* [17], el cual puede recolectar métricas de forma automática, aunque no se encuentra enfocado en el Proceso Personal de *Software*.

La recolección de datos en forma manual es poco confiable [1] y su uso requiere por parte de los

desarrolladores continuos cambios de contexto entre el desarrollo del producto y el registro del proceso [24]. Por lo tanto se hace necesario una herramienta que sea lo más completamente automatizada posible y no invasiva, relacionada con la recolección de datos [1, 7, 24] y su posterior análisis. Los datos a recolectar se pueden dividir en primarios; como lo son el tiempo, el tamaño y los datos de defectos; y secundarios que son los derivados de los primeros, como por ejemplo: defectos removidos y la eficiencia a la fecha en porcentaje [15].

Junto a *Hackystat*, se encuentran herramientas como PROM [38], PSP.NET [15] y PSPA [20], las que presentan una arquitectura parecida, puesto que cada sistema cuenta con un servidor central que recolecta la información que es enviada por extensiones agregadas a un IDE (*Integrated Development Environment*) particular. Uno de los principales inconvenientes que presentan algunas de estas herramientas es que su uso se encuentra limitado a las instituciones donde éstas fueron desarrolladas, por lo que su diseño y principales componentes no han sido descritos ni detallados fuera de este ámbito.

De forma general se puede mencionar que *Hackystat* y PROM, se diferencian de PSP.NET y PSPA en que estas herramientas tienen su enfoque principal en la recolección de métricas y no en ser una implementación completa del PSP. Un problema de esta perspectiva es que el PSP cubre completamente el proceso de desarrollo de *software*, mientras que un IDE típicamente se centra sólo en la implementación [20].

Los sistemas descritos anteriormente comparten varias características, y entre las más importantes se pueden mencionar:

- Se basan específicamente en arquitecturas cliente-servidor u orientada a servicios.
- Implementan sensores dentro de extensiones (*plugins)* para recolectar y enviar los datos desde el cliente al servidor central.
- Utilizan estándares como el protocolo SOAP (*Simple Object Access Protocol*) y XML (*eXtensive Markup Language*), para la transferencia y el manejo de la información respectivamente.
- Recurren a extensiones dentro de herramientas de desarrollo con la finalidad de extender el rango de *software* capaz de operar con el sistema.

En cuanto a las principales falencias que presentan las herramientas, éstas son primero relativas a la privacidad en la transmisión de los datos. Esto se debe a que tienen una orientación de arquitectura distribuida con el fin de operar sobre una red, como lo es Internet. Con la excepción de *Hackystat*, en estas herramientas el registro de tiempo junto con el de defectos, en algunas ocasiones, no pueden ser recolectados de forma automática, por lo tanto se debe presentar la posibilidad de ingresarlos o catalogarlos, en el caso de defectos, de forma manual.

La Tabla 1 muestra las características principales, anteriormente mencionadas, que debe cumplir una herramienta de apoyo a PSP para facilitar su utilización e implementación, junto con el cumplimiento de éstas por los sistemas mencionados.

Tabla 1. Principales características de las herramientas de apoyo a PSP.

Características Principales	**Hackystat**	**PROM**	**PSP.NET**	**PSPA**
Enfoque centrado en PSP	NO	NO	SÍ	SÍ
Recolección automática de datos	SÍ	SÍ	SÍ	SÍ
Privacidad de datos	SÍ	NO	SÍ	SÍ
Generación de reportes PSP	NO	NO	SÍ	SÍ
Portabilidad	SÍ	SÍ	NO	NO
Control real del tiempo y defectos	NO	NO	NO	NO
Privacidad en los datos transmitidos	SÍ	NO	NO	NO
Flexibilidad en la recolección de datos	SÍ	SÍ	SÍ	NO
Facilidades para extender su uso	SÍ	SÍ	SÍ	NO

ARQUITECTURA DEL SISTEMA

Debido a que las herramientas necesitan ser independientes de los entornos de programación (para automatizar la recolección de métricas), sistemas operativos, y lenguajes de programación, se optó por continuar el camino que han tomado los sistemas existentes.

Se pretende, por tanto, utilizar los beneficios que entrega la denominada Arquitectura Orientada a Servicios (SOA, *Service Oriented Architecture*) la que permite a distintos sistemas, escritos en diferentes lenguajes de programación y de distintas plataformas tecnológicas, transformarse en servicios débilmente acoplados y altamente interoperables. De manera puntual se utilizará una implementación de esta arquitectura denominada *Servicios Web*. No es la intención del presente trabajo el presentar o proponer una variación de esta arquitectura, pero sí realizar una implementación tradicional de servicios Web.

El modelo básico de la arquitectura propuesta se puede observar en la Figura 1, en donde la comunicación entre la parte cliente y servidor (en este caso se pueden considerar como contenedores de servicios) se realiza mediante el protocolo SOAP.

Servicios del cliente: En la Figura 1 se observa una herramienta de *software* que permite el ingreso manual de los datos del proceso y la

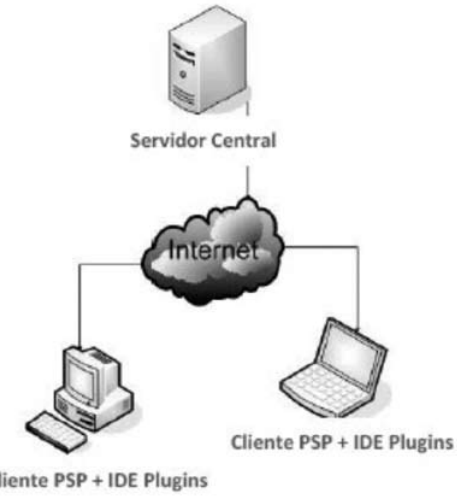

Figura 1. Modelo inicial arquitectura del sistema.

administración de proyectos, denominada "Cliente PSP" y extensiones para IDE, los cuales recolectan y envían las métricas.

Servicios del servidor: El servidor cuenta con el documento WSDL (*Web Services Description Languaje*) que define la interfaz de comunicación de los servicios.

Arquitectura del Sistema

La arquitectura propuesta en la Figura 2 está dividida en las siguientes tres capas:

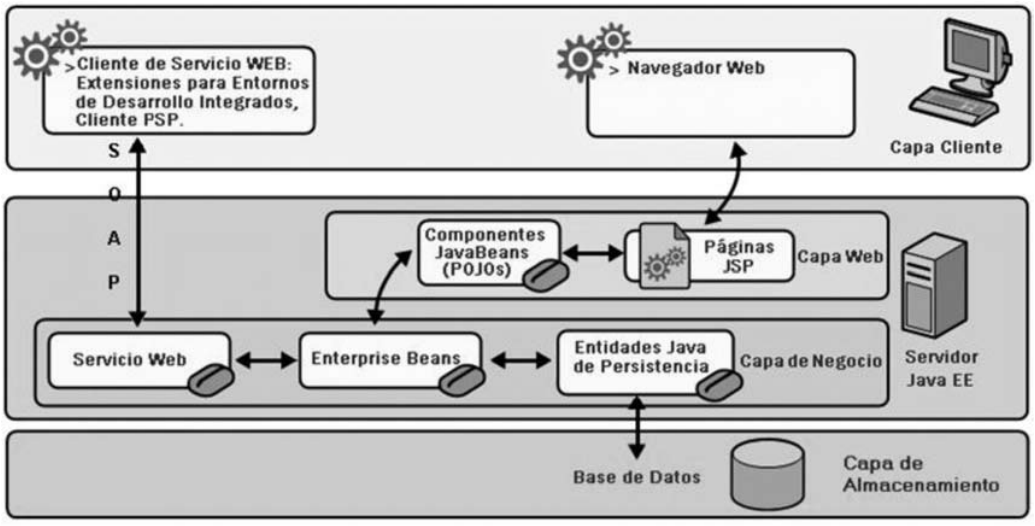

Figura 2. Arquitectura Integral del sistema.

Capa cliente: En esta capa se encuentran las aplicaciones en las que el usuario del sistema puede realizar operaciones de visualización, ingreso, modificación o eliminación de información. Se encuentran las extensiones que comunican los Entornos de Desarrollo Integrado con el servicio Web y la aplicación Cliente PSP.

Servidor Java EE: Dentro de esta capa se encuentra la implementación de la lógica del negocio. Se puede dividir en dos subcapas:

Subcapa Web: Se presentan las páginas Web (utilizando el entorno de desarrollo Java Server Pages) desde las cuales el usuario puede interactuar con el sistema.

Subcapa de negocio: Utilizando la tecnología EJB 3.0 (*Enterprise Java Beans*) se implementa la lógica del negocio.

Dentro de este conjunto se ejecuta el servicio Web (generado a partir del WSDL) y las clases entidades, que la herramienta de mapeo objeto-relacional se encarga de persistir.

Capa de Almacenamiento: En la última capa se encuentra el servidor de base de datos en donde es almacenada la información generada por el sistema.

Para validar la arquitectura propuesta, se procedió a la implementación de un *software,* el que utiliza las siguientes herramientas en cada una de las capas descritas anteriormente:

Capa Cliente: Para satisfacer la necesidad de comunicación con el servidor de aplicaciones, se recurrió a herramientas provistas por las plataformas Java y .NET para su creación.

- *Cliente PSP*: Se utilizó el entorno de desarrollo de aplicaciones de escritorio basados en JAVA Eclipse RCP *Rich Client Plataform*).

- *Extensiones para Entornos de Desarrollos Integrados:* Se construyeron extensiones para los IDE *BlueJ, Eclipse™ y Visual Studio .NET™*.

Capa Servidor: Para implementar el servicio Web y su lógica de negocio, se optó por la tecnología libre llamada Java *Platform Enterprise Edition* o Java EE, la cual es una plataforma de programación (parte de la Plataforma Java) para desarrollar y ejecutar software de aplicaciones en lenguaje de programación Java con arquitectura de "n" niveles distribuida, basándose ampliamente en componentes de software modulares que se ejecutan sobre un servidor de aplicaciones.

Capa de Persistencia de Datos: La base de datos seleccionada para almacenar la información que se genera es MySQL *Community Server* 5.0, la versión gratuita de este sistema de gestión de base de datos.

Otras herramientas utilizadas para satisfacer las funcionalidades requeridas por la aplicación son las siguientes: *Diff, JasperReports, Proyecto Nebula, Apache Commons Math, TopLink JPA*.

FUNCIONALIDADES

Una de las principales características que posee esta arquitectura es la facilidad de integración de clientes recolectores de datos (sensores) que mediante mensajes envían los datos capturados a un servidor de servicios Web.

Métricas de Tamaño y Defecto

La recolección de datos de métricas de tamaño y de defectos se realiza de manera automática mediante plugins o add-ons para los distintos IDE. En este trabajo se implementaron plugins o add-ons para los IDE: *Eclipse, BlueJ y Visual Studio .NET*2005. Cabe señalar que la arquitectura soporta la creación de nuevos plug-ins o add-ons para nuevos IDE.

Clasificación de Defectos

Para la clasificación de defectos, el administrador realiza una categorización de éstos, para cada uno de los lenguajes de programación utilizados, especificando si son errores de compilación o de ejecución. Posteriormente el sistema de manera automática clasifica estos errores.

Estimaciones

Para realizar estimaciones mediante PSP se utiliza el método PROBE, el cual fue implementado por completo en esta herramienta, con el cual los desarrolladores pueden estimar los tamaños y

los tiempos de desarrollo para nuevos programas a partir de su información personal.

Generación de Reportes
El sistema entrega la opción de generar reportes de las principales métricas utilizadas por el PSP, como por ejemplo: métricas del proceso, de calidad, y de defectos por parte de usuarios y administradores. Cada usuario tiene la opción de crearlos a partir de sus propios proyectos, mientras que un administrador posee la facultad de crearlos a partir de la información de todos los usuarios del sistema.

Para la impresión de los reportes generados por *JasperReports*, se utiliza la infraestructura otorgada por el visor, el cual dependiendo del formato de salida puede ser un navegador Web o un visualizador de documentos de tipo PDF (*Portable Document Format*).

Medición de Tiempo
Para realizar la medición del tiempo empleado, el sistema considera dos estados para dicha medición en cada uno de los trabajos de un usuario. Los dos estados posibles son "medición de tiempo activa" y "medición de tiempo detenida". En este caso, el usuario es el que debe seleccionar en cuál de los estados se encuentra su trabajo. Es un proceso que se realiza de forma semiautomática.

VALIDACIÓN DE LA ARQUITECTURA

Con el objetivo de validar la arquitectura propuesta, se procedió a efectuar pruebas al *software* que la implementa.

Dentro de las capas servidor y cliente se realizaron pruebas unitarias a las clases generadas. Para esto se utilizó la herramienta que automatiza este tipo de pruebas para el lenguaje de programación Java denominada JUnit.

Para validar la funcionalidad otorgada por el servicio Web, se procedió a efectuar pruebas del total de operaciones disponibles, descritas en el documento WSDL, utilizando para este fin la aplicación soapUI. Estas pruebas tienen un carácter de integración, además de permitir conocer el rendimiento del servidor a las consultas efectuadas. A modo de ejemplo, para la prueba de acceso (login), como se aprecia en la Figura 3, el resultado obtenido a partir de las variables con valores válidos: $Properties#u1 (username) y $Properties#p1 (password), que representan el nombre de usuario y contraseña, respectivamente, fue satisfactorio, puesto que el servidor retornó la cadena de texto: *valid login*. Para una respuesta negativa el servidor retorna: *invalid login*.

Posteriormente, se aplicó el conjunto de casos de prueba, para validar cada una de las características implementadas. El conjunto de las pruebas funcionales aplicadas se puede apreciar en la Tabla 2. Para el registro automático de métricas se realizaron pruebas que consistieron en capturar defectos (tanto de compilación y en tiempo de ejecución), junto con rescatar el tamaño del código fuente, medido en SLOC, desde los IDE *BlueJ, Eclipse* y *Visual Studio.NET*. Se logró comprobar la efectiva captura de estos datos en el sistema. Dentro del registro de métricas, desde el Cliente PSP, se ingresó de forma

Figura 3. Prueba de acceso (login) del sistema.

Tabla 2. Grupo de características afectas a pruebas funcionales.

Grupo de características	**Descripción**
Registro automático de métricas	Registro de tamaño y defectos enviados desde Entornos de Desarrollo Integrados dentro de un proyecto perteneciente a un usuario.
Registro de métricas	Ingreso de tiempo (cronometrado por el usuario dentro del "Cliente PSP") y tamaño (a partir de selección de archivo fuente dentro del "Cliente PSP").
Manejo de proyectos	Administración de proyectos y respectivos módulos de usuario.
Manejo de bitácoras del PSP	Manejo de bitácoras de tiempo, tamaño y defectos. El usuario puede modificar atributos de tamaño, como por ejemplo SLOC base y SLOC reutilizadas.
Manejo de estimaciones	Generación de estimaciones de tamaño y tiempo de un usuario utilizando el método PROBE a partir de sus datos históricos.
Manejo de estándares del PSP	Administración de estándares del Proceso Personal de *Software* desde ambiente Web de administración.
Manejo de guías del PSP	Administración de guías del PSP por parte del administrador del sistema utilizando página Web correspondiente.
Manejo de reportes del PSP	El usuario puede generar reportes a partir de sus datos históricos de forma personalizada. Un administrador del sistema puede obtener reportes de usuarios determinados por él.
Manejo de usuarios del sistema	Un administrador puede manejar los usuarios que tienen acceso al sistema.
Manejo de sincronización de información	Si el usuario posee dos servidores, puede realizar una sincronización de éstos.

asistida el tiempo utilizado por el usuario durante un proceso de desarrollo completo, permitiendo comprobar el tiempo efectivo de su trabajo. Se realizaron diferentes estimaciones para diversos proyectos basándose en los propios datos históricos de distintos usuarios. Comprobando, que mientras más información real se posea, las estimaciones serán de mejor calidad, acercándolas más a la realidad. Se crearon y administraron proyectos de los distintos niveles del Proceso Personal de *Software*, validando el correcto funcionamiento de las bitácoras, resúmenes de planes de proyecto, propuestas de mejoramiento del proceso, entre otros elementos del PSP. Para los distintos proyectos administrados por usuarios, se obtuvieron los diferentes tipos de reportes disponibles, lo que permitió comprobar el correcto funcionamiento de esta característica. Finalmente, se logró sincronizar la información correspondiente a un usuario entre dos servidores que acceden a diferentes bases de datos. En lo que respecta a la administración del sistema se procedió a comprobar el correcto funcionamiento de las características relacionadas con el manejo de los elementos del PSP, como por ejemplo el manejo de guías y estándares.

Seguridad

El sistema tiene la capacidad de ofrecer seguridad en la transferencia de la información personal del usuario entre la capa cliente y servidor gracias a la transmisión mediante la capa socket seguro SSL (*Secure Sockets Layer*). Con este fin se debe importar un certificado, generado y proveído por el servidor, el cual es utilizado para encriptar la información que viaja entre ambos.

- *Seguridad del Cliente PSP (servicio Web*) Si el usuario desea puede activar la opción de transporte seguro al efectuar la comunicación entre el Cliente PSP con el servidor gracias a la transmisión mediante la capa socket seguro SSL (*Secure Sockets Layer*).

- *Seguridad en la navegación.* Para acceder a las páginas Web de la administración con contenido protegido se trabaja con un certificado digital, lo cual asegura la autentificación correcta del administrador.

Cabe hacer notar que el objetivo de esta validación es probar la implementación del conjunto de requisitos funcionales y no funcionales que se obtuvo a partir de la revisión bibliográfica y que se listó en Tabla 1. Este principalmente consistía en automatizar más funciones de recolección de datos (tiempo y defectos) y generación de informes que las herramientas existentes, proveer un enfoque centrado en PSP que sea portable y entregue privacidad, además de ser independiente del entorno. Todo esto se cumple a cabalidad.

CONCLUSIONES

En el presente documento se presentó una arquitectura de *software* que tiene como objetivo primordial, el satisfacer los requisitos de automatización y extensibilidad que una herramienta de *software* de apoyo a la implementación del Proceso Personal de *Software* debe poseer. Para cumplir con estas exigencias se optó por utilizar la denominada Arquitectura Orientada a Servicios ya que esta entrega las facilidades que permiten poder lograr interoperabilidad entre distintos Entornos de Desarrollo Integrado con un servidor central que administre y almacene la información.

Desde su presentación el PSP no ha logrado alcanzar una adopción importante dentro de la industria de *software* y el ámbito académico. Esto se debe principalmente, según estudios, a lo difícil que resulta recopilar una gran cantidad de datos a partir del trabajo realizado, lo que además conlleva a pérdida de tiempo.

El Proceso Personal de *Software* requiere una recopilación y análisis de métricas con un elevado nivel de detalle, lo cual no es algo trivial. En cualquier proyecto real, el empleo de herramientas de apoyo al PSP se convierte en un elemento muy importante a considerar. Con este objetivo durante el presente proyecto se analizaron un cúmulo de herramientas disponibles con el objetivo de comprender el enfoque de éstas, y extrayendo de dichas aplicaciones características que aportaron al diseño de nuestra solución.

El software desarrollado a partir de la arquitectura presentada en este documento contribuye con un conjunto de características que no proveen las otras herramientas de apoyo al PSP que fueron discutidas previamente en la sección del marco teórico. Algunas de estas características son: enfoque centrado en PSP, privacidad de datos, generación de reportes, portabilidad, privacidad de los datos transmitidos, flexibilidad en la recolección de datos, facilidades para extender su uso, y un alto grado de control real del tiempo y defectos.

Estas características permiten que el software propuesto entregue soporte automático para utilizar el PSP:

- Generando reportes y gráficos del PSP.
- Clasificando los defectos derivados de la compilación y ejecución de programas.
- Facilitando la medición de tiempo empleado por el usuario durante su proceso.
- Contando las líneas de código fuente (SLOC) de múltiples lenguajes, como por ejemplo Java, C y C# enviadas desde las extensiones agregadas a los distintos IDE.
- Suministrando soporte para estimación de tamaño y tiempo.

El conjunto de pruebas realizadas valida la arquitectura propuesta y verifica que la herramienta de soporte cumple con las características requeridas por el Proceso Personal de *Software* (ver Tabla 1).

En cuanto al uso que se le puede dar al software en el plano académico e industrial, se hace necesario señalar que éste debe ser utilizado por el desarrollador como una herramienta de apoyo a su proceso personal, con el objetivo de ayudarle a conocer su proceso de forma continua, para así poder optimizarlo y no utilizarlo como una simple herramienta de control.

Trabajo a Futuro

Como trabajo a futuro se puede proponer el uso de esta herramienta por alumnos de ramos iniciales de computación para comprobar su real efectividad en sus procesos de desarrollo de software. Una vez que la herramienta sea validada por alumnos, podrá ser utilizado como un aporte real a los profesionales del área.

Para el mejoramiento del software propuesto se plantea a futuro la integración de determinadas funciones específicas, como la planificación y seguimiento de proyectos, y herramientas para la interacción entre miembros que elaboran un mismo proyecto.

REFERENCIAS

[1] A.M. Disney and P.M. Johnson. "Investigating Data Quality Problems in the PSP". Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering, pp. 143-152. New York, NY, USA. 1998.

[2] A.M. Disneya and P.M. Johnson. "A Critical Analysis of PSP Data Quality: Results from a Case Study". Empirical Software Engineering. Vol. 4, Issue 4, pp. 317-349. 1999.

[3] B.R. von Konsky, J. Ivins and M. Robey. "Using PSP to Evaluate Student Effort in Achieving Learning Outcomes in a Software Engineering Assignment". Proceedings of the 7th Australasian conference on Computing education. 2005.

[4] C. Wohlin and A. Wesslen. "Understanding Software Defect Detection in the Personal Software Process". The Ninth International Symposium on Software Reliability Engineering. 1998.

[5] I. Sommerville. "Ingeniería de Software". Sexta Edición. Addison Wesley-Pearson Educación. 2002.

[6] I. Hirmanpour and S. Khajenoori. "Personal Software Process Technology: An Experiential Report". Proceedings of the Information Systems Education Conference (ISECON). 2000.

[7] I.D. Coman. "An Analysis of Developers Task using Low-Level, Automatically Collected Data". Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. 2007.

[8] J.I. Maletic, A. Howald and A. Marcus. "Incorporating PSP into a Traditional Software Engineering Course: An Experience Report". Proceedings 14th Conference on Software Engineering Education and Training, pp. 89-97. 2001.

[9] K. Venkatasubramanian, S.B.T. Roy and M.V Dasari. "Teaching and Using PSP in a Software Engineering course: An Experience Report". Proceeding of the Software Engineering Education and Training Annual Conference. Chennai, India. 2001.

[10] L. Prechelt, B. Unger and O. Gramberg. "Experience Report: Teaching and Using the Personal Software Process (PSP)". 1997.

[11] L. Prechelt and B. Unger. "An Experiment Measuring the Effects of Personal Software Process (PSP) Training". IEEE Transactions on Software Engineering. Vol. 27, Issue 5, pp. 465-472. May, 2001.

[12] M. Pomeroy-Huff, R. Cannon, T.A. Chick, J. Mullaney and W. Nichols. "The Personal Software Process (PSP) Body of Knowledge". Software Engineering Institute. 2005.

[13] M. Host and C. Wohlin. "An Experimental Study of Individual Subjective Effort Estimations and Combinations of the Estimates". Proceedings of the 1998 International Conference on Software Engineering. 1998.

[14] M. Morisio. "Applying the PSP in Industry". IEEE Computer Society. Vol. 17, Issue 6, pp. 90-95. Nov./Dec., 2000.

[15] M. Hairul, A.M. Yusolf. "Automating a Modified Personal Software Process". Malaysian Journal of Computer Science. Vol. 18, Issue 2, pp. 11-27. December, 2005.

[16] P. Ferguson, W.S. Humphrey, S. Khajenoori, S. Macke and A. Matvya. "Results of Applying the Personal Software Process". IEEE Computer Society. Vol. 30, Issue 5, pp. 24-31. May, 1997.

[17] P.M. Johnson, H. Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, S. Zhen and W.E.J. Doane. "Beyond the Personal Software Process: metrics collection and analysis for the differently disciplined". Proceedings of the 25th International Conference on Software Engineering. IEEE Computer Society, pp. 641-646. 2003.

[18] P.M. Johnson, C.A. Moore, J.A. Dane and R.S. Brewer. "Empirically-Guided Software Effort Guesstimation". IEEE Software. Vol. 17, Issue 6, pp. 51-56. 2000.

[19] R.F. Grove. "Using the Personal Software Process to Motivate Good Programming

Practices". ACM SIGCSE Bulletin. Vol. 30, Issue 3, pp. 98-101. September, 1998.

[20] R. Sison. "Personal Software Process (PSP) Assistant". APSEC `05. 12th Ais-Pacific Software Engineering Conference. 2005.

[21] R. Schoedel. "PROxy Based Estimation (PROBE) for Structured Query Language (SQL)". Microsoft Corporation. Software Engineering Institute. Technical Report. 2006.

[22] R.B. Grady. "Practical Software Metrics for Project Management and Process Improvement". Prentice-Hall. 1992.

[23] R.S. Pressman. "Ingeniería del Software: Un enfoque práctico". Sexta Edición, McGraw-Hill. 2002.

[24] R. Moser, A. Janes, B Russo, A. Sillitti and G. Succi. "PROM: taking an echography of your software process". XLIII Congresso Annuale AICA. AGILE Publications. Udine, Italy. 2005.

[25] M. Pomeroy-Huff, R. Cannon, T.A. Chick, J. Mullaney and W. Nichols. "The Personal Software Process (PSP) Body of Knowledge, Version 2.0". Software Engineer Institute. Fecha de consulta: 8 de marzo 2011. URL: http://www.sei.cmu.edu/reports/09sr018.pdf

[26] S.K. Lisack. "The Personal Software Process: Where Does it Belong in a Undergraduate Information System Curriculum?". The 15th Conference on Software Engineering Education and Training. 2002.

[27] T.B. Hilburn. "PSP Metrics in Support of Software Engineering Education". Proceedings of the 12th Conference on Software Engineering Education and Training, pp. 135-136. 1999.

[28] W.S. Humphrey. "A Discipline for Software Engineering". First Edition, Addison-Wesley. 1995.

[29] W.S. Humphrey. "Why Should You Use A Personal Software Process?". Software Engineering Notes. Vol. 20, Issue 3, pp. 33-36. 1995.

[30] W.S. Humphrey. "Introduction to the Personal Software Process". Addison-Wesley. 1996.

[31] W.S. Humphrey. "The Personal Software Process (PSP)". Technical Report. The Software Engineering Institute. 2000.

[32] W.S. Humphrey. "The Personal Software Process: Status and Trends". IEEE Software. Vol. 17, Issue 6, pp. 71-75. 2000.

[33] W. Hayes. "Using a Personal Software Process to Improve Performance". Proceedings Fifth International Software Matrics Symposium. pp. 61-71. 1998.

[34] W.I. Bullers. "Personal Software Process in the Database Course". ACE '04 Proceedings of the sixth conference on Australasian computing education. Vol. 30. pp. 25-31. 2004

[35] X. Zhong, N.H. Madhavji and K. El Emam. "Critical Factors Affecting Personal Software Process". IEEE Software. Vol. 17, Issue 6, pp. 76-83. 2000.

[36] Y. Zulueta Véliz. "Introducción de técnicas del Personal Software Process desde los primeros años en la formación del ingeniero informático". Revista Ingeniería Informática. Fecha de consulta: 8 de marzo 2011. URL: http://www.inf.udec.cl/~revista/ediciones/edicion14/zulueta.pdf

[37] Z. Car. "A Method for Teaching a Software Process Based on the Personal Software Process". The 21st IASTED International Multi-Conference on Applied Informatics. Innsbruck, Austria. February, 2003.

[38] A. Sillitti, A. Janes, G. Succi and T. Vernazza. "Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data". Proceedings 29th Euromicro Conference. September, 2003.

[39] ETSU Design Studio. "PSP Studio". Fecha de consulta: 8 de marzo 2011. URL: http://csciwww.etsu.edu/psp/dlpsps.html

[40] D. Tuma. "PSP Dashboard". Fecha de consulta: 8 de marzo 2011. URL: http://processdash.sourceforge.net/

[41] C.A. Moore. "Project Leap: Personal Process Improvement for the Differently Disciplined". Proceedings of the 21st International Conference on Software Engineering. pp. 726-727. 1999.