Plataforma de Desarrollo Rápido de Aplicaciones bajo el Proceso de Software Personal: en busqueda de agilidad, solidez y disciplina para la Ingeniería de Software

Autor: Mariano Reingart
mreingart@unimoron.edu.ar

Carrera de Grado: Licenciatura en Sistemas
Fecha de Defensa: 16 de Diciembre de 2011
Tutor: Lic. Oscar Bravo
Titular de Cátedra: Dr. Jorge S. Ierache
(Director de Investigación FICCTE)

Facultad de Informática, Ciencias de la Comunicación y Técnicas Especiales
Universidad de Morón
Cabildo 134, (B1708JPD) Morón, Buenos Aires, Argentina
+54 (011) 5627-2000

Resumen: En el campo del Desarrollo Rápido de Aplicaciones, comúnmente existen diferentes aproximaciones metodológicas, teorías, estándares y buenas prácticas para llevar a cabo el Desarrollo de Software; generalmente sin una sólida fundamentación científica o sin basarse en datos empíricos completos e imparciales, agravándose sobre todo en la búsqueda de soluciones a las falencias o riesgos que presenta el método, por lo que es difícil encontrar una herramienta que de soporte integral al proceso, cuestión fundamental para la efectividad del método.
Este trabajo busca definir una base teórica, para aumentar la productividad pero resolviendo los problemas asociados al método de desarrollo rápido; y proveer una herramienta integrada que le de soporte.
Se utilizará el Proceso de Software Personal, un enfoque simple y predecible guiado por métricas y estimaciones estadísticas, desarrollando e integrando una nueva generación de herramientas, para encauzar la metodología ágil, solucionando sus falencias pero sin perder sus beneficios.

Palabras Clave: Metodologías Ágiles; Proceso de Mejora Continua; Aseguramiento de Calidad; Herramienta Integrada de Ingeniería de Software Asistida por Computadora (I-CASE); In-Process Software Engineering Measurement and Analysis System; recolección automática de métricas; Personal Software Process (PSP); Rapid Application Development (RAD); Software Libre; Código Abierto; Python; web2py; WxPython; PostgreSQL;

Plataforma de Desarrollo Rápido de Aplicaciones bajo el Proceso de Software Personal: en búsqueda de agilidad, solidez y disciplina para la Ingeniería de Software

Autor: Mariano Reingart

Carrera de Grado: Licenciatura en Sistemas
Fecha de Defensa: 16 de Diciembre de 2011
Facultad de Informática. Universidad de Morón

mreingart@unimoron.edu.ar

Resumen: En el campo del Desarrollo Rápido de Aplicaciones, comúnmente existen diferentes aproximaciones metodológicas, teorías, estándares y buenas prácticas. Estas bases conceptuales generalmente no tienen una sólida fundamentación científica o no se basan en datos empíricos completos e imparciales. El problema se agrava dados los riesgos que presentan las metodologías ágiles, principalmente si las herramientas son incorrectas o su uso es inadecuado.

Este trabajo busca definir una base teórica y proveer una herramienta integrada que le de soporte, para aumentar la productividad, resolver los problemas asociados al método de desarrollo rápido y posibilitar alcanzar niveles de madurez al proceso.

Se utilizará el Proceso de Software Personal, un enfoque simple y predecible guiado por métricas y estimaciones estadísticas. Se desarrolla e integra una nueva generación de herramientas, para encauzar la metodología ágil analizada, buscando solucionar sus falencias pero sin perder sus beneficios.

Palabras Clave: Metodologías Ágiles; Proceso de Mejora Continua; Aseguramiento de Calidad; Herramienta Integrada de Ingeniería de Software Asistida por Computadora (I-CASE); In-Process Software Engineering Measurement and Analysis System; recolección automática de métricas; Personal Software Process (PSP); Rapid Application Development (RAD); Software Libre; Código Abierto; Python; web2py; WxPython; PostgreSQL;

1. Introducción

James Martin [1] argumentó que los desarrolladores pueden lograr grandes reducciones en tiempo y costo, entregando sistemas de información de gran calidad al usar métodos modernos que combinan el involucramiento extensivo del usuario final con metodologías modernas de desarrollo soportadas por herramientas de diseño asistido por computadora (CASE) bien integradas (a lo que denomina I-CASE).

En relación a los riesgos y viabilidad del método, en un estudio de campo publicado por la ACM [2], se destaca que si bien existe evidencia creciente que apoya al Desarrollo Rápido de Aplicaciones (RAD) como una mejora del "orden de magnitud" en la velocidad de construcción de software, emergen **implicaciones serias**. En primer lugar, es importante reconocer que RAD representa un cambio radical, por lo que se debiera realizar entrenamiento extensivo. Más importante aún, se deben evitar las capacidades RAD que deterioran las buenas prácticas en el desarrollo de sistemas, ya que solo se puede lograr alta calidad, bajo costo y desarrollo rápido si se utiliza una metodología de desarrollo (proceso de software disciplinado [13]).

Si bien existen otros acercamientos al tema para solucionar los **problemas de calidad**, como ser el Método Dinámico de Desarrollo de Sistemas (DSDM) desarrollado a partir del RAD, que también se basa en enfocarse no solo en la velocidad sino también en la calidad. Sin embargo, a diferencia del proceso PSP más tradicional y robusto, no existe evidencia empírica públicamente disponible y consensuada sobre la efectividad del DSDM [17].

El Proceso de Software Personal (PSP) fue desarrollado en 1993 por Humphrey [4], y es un marco personal y disciplinado de trabajo para llevar a cabo las tareas de software, que consiste en una serie de métodos, formularios y guiones que muestran a los ingenieros de software como planear, medir y administrar su trabajo. La meta del proceso es producir productos con cero defectos, respetando el calendario y con los costos planeados.

La efectividad de la metodología PSP fue documentada tanto en entornos académicos e industriales. El resultado de estos estudios encontró que el PSP **mejoró el rendimiento** (estimación del tamaño y esfuerzo, calidad de producto y proceso) mientras que **no afecto la productividad** [5].

Si bien estos estudios publicados demuestran las ventajas del PSP (por. ej. en Microsoft India [26]), diversos trabajos también señalan ciertos inconvenientes en la recolección manual de los datos y posterior análisis por parte de los desarrolladores, cuestionando su exactitud, recomendando que el uso de herramientas integradas debe ser necesario para un análisis de alta calidad de los datos del PSP [6].

También se sugiere que si bien los estudiantes aprenden el PSP y logran mejorar sus procesos, lo abandonan cuando ya no se les requiere utilizarlo. Este **"problema de adopción"** puede ser producido por dos causas: sobrecarga de trabajo por la recolección y análisis métricas y el cambio de contexto (entre la herramienta de desarrollo y la recolección de métricas) [7].

El objetivo de esta investigación es solucionar los problemas de calidad del las metodologías ágiles (RAD en particular), mitigando el problema de adopción de procesos disciplinados (PSP) y mejorando la calidad de métricas estadísticas como base para estimaciones y mejora continua.

Para tal fin, el presente artículo presenta una "Plataforma de Trabajo" y se organiza de la siguiente manera: en la Sección 2 se analiza el Estado del Arte mediante un relevamiento bibliográfico, descripción y enumeración de las herramientas actuales; en la Sección 3 se propone la solución, con su correspondiente elección del modelo a desarrollar, las herramientas a integrar, el proceso de software y metodología a utilizar, con su adecuado sustento teórico y tecnológico; en la Sección 4 se describe el desarrollo de experimentos para probar el prototipo piloto, con los resultados obtenidos; en la Sección 5 y 6 se presenta la conclusión y futuras líneas de investigación que posibilita el presente trabajo, tanto en los ámbitos profesionales, educativos y académicos. Por último, se presentan las referencias bibliográficas a libros y sitios de Internet consultados y/o citados en la presente investigación, y en los anexos se pueden encontrar las tablas de datos analizados y siglas utilizadas.

2. Antecedentes

El Process Dashboard es una herramienta de soporte al PSP, considerada de segunda generación. Fue originalmente desarrollada en 1998 por la Fuerza Aérea de Estados Unidos, y ha continuado evolucionando bajo el modelo Open Source. Es analizado en un artículo publicado en la ACM [6] como mejora a la recolección y análisis de datos del PSP.

En la Figura 1 se presenta una captura de pantalla de dicha herramienta, con los controles básicos para inicio/detención de la medición del tiempo, contador de defectos (bugs), menú de jerarquía (proyecto, producto, programa) y fase del PSP:

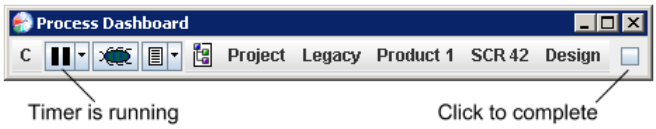

Figura 1: (PSP) Process Dashboard [8]

Como ventajas se destaca la simplificación de la recolección de datos y el soporte automático para analizarlos, lo que disminuye sensiblemente la sobrecarga de trabajo necesaria para utilizar el método PSP, obteniendo resultados de mejor calidad que al utilizar herramientas manuales de primera generación.

Como desventajas, se observa el cambio de contexto necesario entre las aplicaciones de desarrollo y la herramienta de recolección de datos, esto es, constantemente ir a la herramienta de recolección e informarle en que fase y estado se encuentra en este momento, cantidad de defectos, tiempos de inicio y finalización,

etc. Este cambio de contexto, si bien puede ser una acción simple como hacer clic en un botón, puede ser tomado por algunos usuarios como algo muy intrusivo cuando necesitan períodos ininterrumpidos enfocados en el desarrollo.

A su vez, al ser un proceso semiautomático, el usuario es aún responsable de determinar e informar manualmente en que fase, que momentos y demás datos cuantitativos como defectos y líneas de código, lo que pudiera ocasionar inconsistencias por errores u omisiones (ej. el usuario se olvidó de detener el reloj al recibir una llamada telefónica; omitió procesar un archivo fuente por lo que no se contaron correctamente las líneas de código; error de tipeo al informar la cantidad de defectos; etc.)

En Mayo de 2001 se inició el proyecto Hackystat [7], una herramienta considerada de tercera generación (análisis y medición automatizada e incorporada al proceso, AISEMA). Recolecta automáticamente las métricas conectando sensores directamente a las herramientas de desarrollo. Los datos son enviados automáticamente a un servidor, el cual los procesa y analiza, enviando alertas. Su arquitectura se resume en el esquema mostrado en la Figura 2:

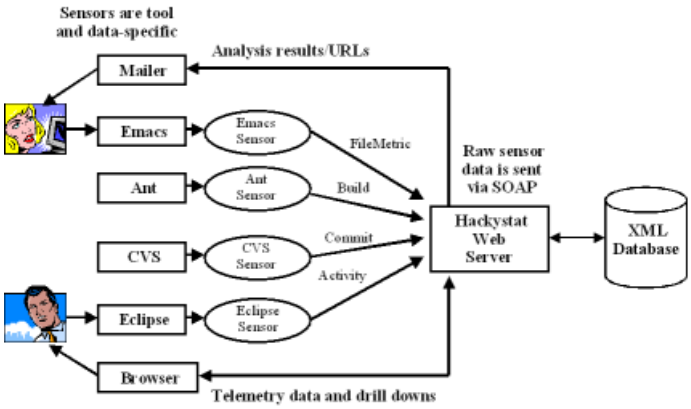

Figura 2: Esquema de Hackystat [9]

Como ventajas se destaca la total automatización de la recolección y el análisis de datos sobre métricas, lo que prácticamente anula la sobrecarga de trabajo y el cambio de contexto, proveyendo una mejora substancial sobre las herramientas de segunda generación.

Como desventajas, los mismos autores destacan el cambio en el tipo de métricas recolectadas, esto es, una especie de "gran hermano", ya que el servidor Hackystat pone a disposición un registro detallado de las actividades de los desarrolladores, lo que puede ocasionar preocupaciones sobre la privacidad de los profesionales.

Actualmente existen diversos proyectos similares a la presente investigación, pero con distintos enfoques y metas:

- Jasmine, A PSP Supporting Tool [19]: posee recolección automática de ciertas métricas y una guía electrónica de proceso (EPG), que permite una fácil navegación por los elementos del PSP, y un repositorio de experiencias (ER), que permite almacenar y compartir información relativa al proyecto. Como desventaja se aclara que no puede detectar tiempo fuera de Eclipse (modificación otros documentos, revisión de código o testeo) y no puede rastrear errores post-publicación.
- PSP EVA [20]: aparentemente superadora respecto a las anteriores con una propuesta de "agentes" (ver Figura 4), presentaría una mejor visualización del progreso, con una interfaz proactiva para que la interacción no se vuelva una sobrecarga y provee flexibilidad en el flujo de fases (para evitar la secuencialidad planificación, diseño, codificación, compilación, pruebas y post-mortem)
- PSP Assistant [18], que si bien esta enfocada a PSP y posee recolección de métricas automática (tiempos, defectos, LOC), lo cubre solo parcialmente -por ej., no soporta todas sus fases, solo codificación/compilación y tests solo de unidades. La integración En la Figura 3 se puede observar las utilidades del PSP Assistant integrado dentro de la herramienta Eclipse.

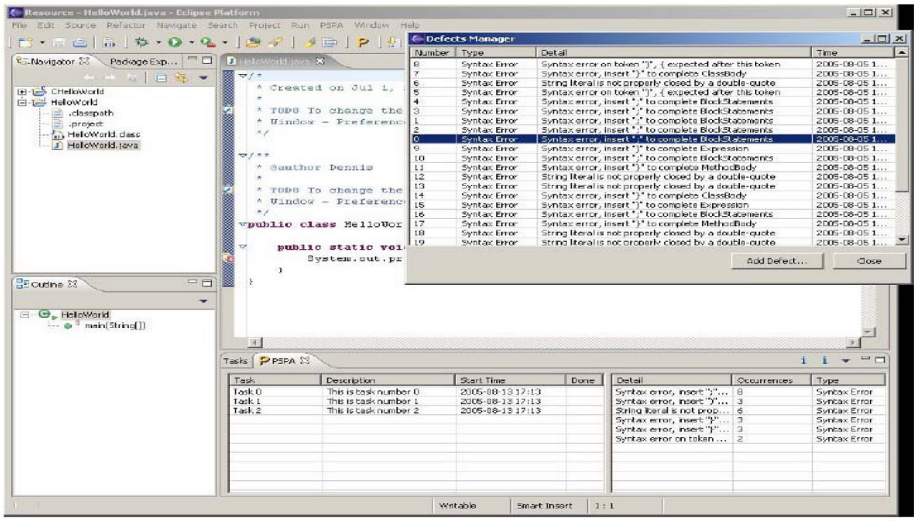

Figura 3: Eclipse IDE Workspace con ventanas del PSP Assistant [18]

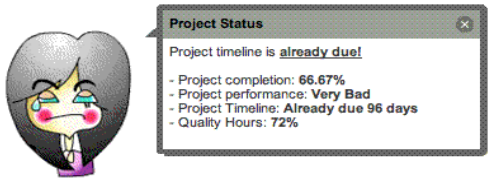

Figura 4: Ejemplo de agente con rendimiento del proyecto [20]

Si bien estos proyectos buscan integrar y ayudar a los ingenieros a entender y usar el PSP de manera efectiva, surgen las siguientes implicaciones:

- no son herramientas de desarrollo en si, sino que utilizan sensores agregados, por lo que pueden tener complicaciones para implementar funcionalidades no contempladas y recolectar correctamente las métricas de forma automatizada (por ej, por no poder medir tiempos de planeamiento, diseño o revisión, por estar dichas características fuera de las IDE tradicionales, o no poder contemplar todos los casos para detección de defectos)
- no son enfoques personales sino que están diseñados para grandes organizaciones y equipos, lo que puede dificultar su instalación y mantenimiento,
- no contemplan las metodologías de desarrollo rápido de aplicaciones per se, y no están orientadas hacia un framework particular (por ej python+web2py), lo que se entiende necesario para esta investigación ya que la adopción PSP es una parte de la problemática junto con RAD (la dispersión conceptual puede afectar la uniformidad de las métricas)
- son herramientas propietarias (no están disponibles públicamente y su código es cerrado), dada la escasa información publicada que se pudo recolectar, lo que dificulta su análisis y mejora, presentando cuestiones legales incompatibles con el software libre.
- están realizadas en PHP, .NET o JAVA, lo que dificulta su modificación y/o integración con sistemas desarrollados en Python, y en ciertos casos, su operación multiplataforma es inviable o dependen de paquetes complejos como Eclipse.

Por lo tanto, se entiende que es necesario solucionar la problemática actual quedando en evidencia la necesidad de una herramienta de "cuarta generación" para Python, realizada en software libre de código abierto, que integre completamente el proceso de desarrollo de software para simplificar y sustentar una efectiva disciplina personal, totalmente automatizada, sin necesidad de programas o tareas adicionales.

3. Solución Propuesta

3.1. Elección del modelo

Como solución, para el presente trabajo, vistos los problemas descriptos en la introducción y antecedentes, y luego de efectuar una investigación sobre el tema, se plantea:

1. Integrar la cadena de herramientas de desarrollo para lograr una herramienta I-CASE que de soporte completo al RAD, estructurándolas sobre el Proceso de Software Personal, buscando con esto solucionar las deficiencias comúnmente asociadas al método de desarrollo (poco uso de herramientas de modelado, subversión potencial de la metodología de desarrollo, falta de planeamiento y control, etc.).

2. Integrar directamente la recolección y análisis de métricas dentro de las herramientas de desarrollo I-CASE, orientado a eliminar los problemas de datos observados al PSP (cambio de contexto, sobrecarga de trabajo, privacidad, etc.), obteniendo estadísticas confiables para la evaluación de la calidad y el proceso de mejora continua.

El esquema general de las herramientas desarrolladas se detalla en la Figura 3.1:

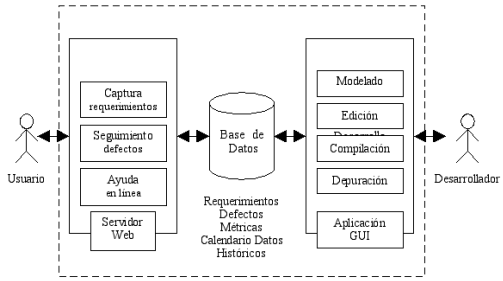

Figura 5: Esquema básico de la Herramienta I-CASE desarrollada

Al integrar la cadena de herramientas de desarrollo RAD, enfocado sobre el proceso PSP, se piensa obtener los siguientes beneficios:

- Desarrollar e Integrar las diferentes herramientas de cada etapa del ciclo de vida (análisis, diseño, codificación, implementación), poniendo énfasis no solo en la construcción sino también en el análisis del dominio;
- A través de las herramientas, hacer cumplir un proceso bien definido (PSP), para evitar que se desvirtúe o directamente se abandone la estructura de los proyectos, mejorando la capacidad de ser controlados y por ende la calidad final de sus productos;

A su vez, al integrar las métricas PSP a la herramienta de desarrollo, se piensa que mejorará el proceso ya que:

- No hay cambio de contexto ni sobrecarga de trabajo ya que la recolección de datos para las métricas será automática en la misma herramienta I-CASE,
- Por lo tanto no se utilizan sensores que pueden ser accedidos remotamente, obteniendo el análisis directamente de la misma herramienta, dándole al desarrollador total control de la información personal recolectada.
- Mayor retroalimentación con respecto al análisis de las métricas, ya que se puede realizar un seguimiento prácticamente instantáneo mejorando de esta forma el proceso de mejora continua.

3.2 Lenguaje de Programación, Framework Web y Toolkit GUI

Para lograr desarrollar esta herramienta de ingeniería de software integrada (I-CASE) se propone utilizar el entorno de programación Python [10], ya que se cree que se ajusta al método RAD al cumplir los siguientes puntos [11], [12]:

- Prototipado Rápido: al ser un lenguaje dinámico, con tipos de datos nativos de alto nivel, de sintaxis clara y fácil lectura acelera la productividad, sin comprometer la mantenibilidad;
- Reuso: provee modularidad completa, jerarquía de paquetes, librería estándar extensiva, facilidad de extensión vía C, C++, Java, .NET;

Adicionalmente, para el presente trabajo, web2py [22] es la herramientas más adecuada de introducir para obtener una mejor calidad desde el marco conceptual planteado, debido a que cumple todas las características de desarrollo rápido de aplicaciones (prototipado, iterativo, incremental, priorizando la funcionalidad y posee testeo integrado), es centrado en bases de datos relacionales y posee herramientas CASE totalmente integradas (IDE de desarrollo, pruebas, depuración, despliegue), útil incluso para aplicaciones científicas [23].

Un punto importante fue definir el tipo de entorno (aplicación web o aplicación "visual" GUI). Inicialmente web2py es una aplicación web totalmente administrable por Internet (por ej. con editor de código por el navegador). Esto es muy útil para ambientes distribuidos, pero es limitado para desarrollar un prototipo simple en tiempo y forma, por lo que se desarrolló un prototipo "visual" con interfaz gráfica de escritorio basado en wxPython, un conjunto de herramientas GUI para Python [28].

3.3. Implementación de la Solución

Este trabajo se enfoca, en primer lugar, en desarrollar una IDE (entorno de desarrollo integrado), que de soporte al PSP (tanto a nivel fases como recolección de métricas), denominado IDE2PY (Figura 6):

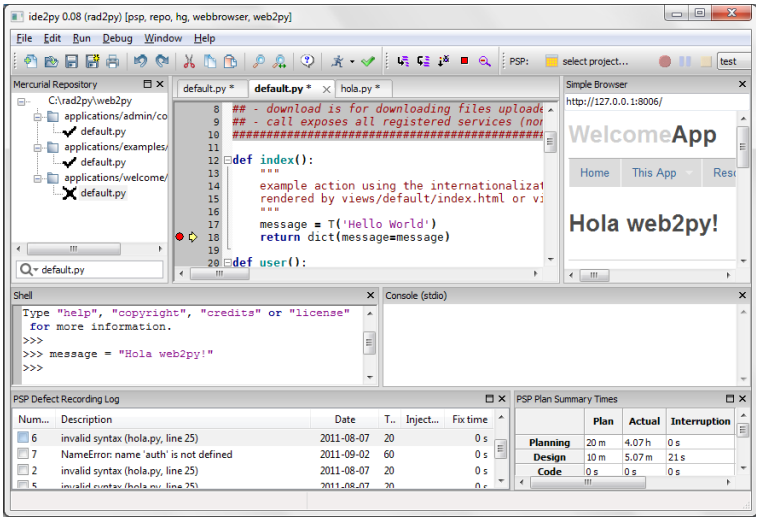

Figura 6: Captura de la pantalla principal de la IDE desarrollada (ide2py)

Internamente, el diseño de la IDE2PY es muy simple y modular, con un módulo principal (main.py), conteniendo la aplicación en sí y la administración de la Interfaz Avanzada de Usuario (AUI) con la ventana madre, los menúes, barra de herramientas y distintos paneles. A su vez, la aplicación se encarga de la configuración utilizando archivos de texto estándar .INI.

El editor (editor.py) es un control basado en Styled Text Control de wxPython, con resaltado de sintaxis, autocompletado y calltips, y por cada archivo es creada una ventana separada que es manejada automáticamente con distintas solapas la Interfaz de Documentos Múltiples (AUI MDI). La introspección es básica (con un espacio de nombres estático), y se basa en evaluar/importar los módulos necesarios, por lo que en un futuro se buscarán métodos menos invasivos.

El interprete interactivo (shell.py) se basa en los controles proporcionados por wxPython, con adaptaciones para integrarlo con la consola (console.py) y depurador (debugger.py). Este último esta basado en el propio depurador de Python (bdb), con ajustes e integración con el editor.

El manejo de repositorios esta abstraído en dos capas, la primera de alto nivel (repo.py) con las operaciones básicas comunes a todos los repositorios, y la segunda específica de cada sistema de versionado (repo_hg.py), en este caso con soporte para Mercurial[1], pudiendo agregarse otros sistemas. Para comparar las diferencias se mejoró wxpydiff.py y se adaptaron los módulos estándar de python con un comparador de secuencias -FancySequenceMatcher- para detectar correctamente los cambios en el código fuente (diffutil.py).

Para el chequeo estático se integró PyChecker[2] y PyFlakes[3] (checker.py) que analizan el código fuente y reportan los defectos encontrados, complementado con pruebas de documentación (tester.py) que ejecutan las funciones y comparan los resultados esperados según la documentación de las mismas.

En segundo lugar, se desarrolla una herramienta de soporte y seguimiento, denominada PSP2PY (Figura 7), con interfase web, para la adquisición de requerimientos, seguimiento de errores, generación de documentación on-line; con las siguientes funcionalidades:

1 Mercurial: sistema de control de versiones http://mercurial.selenic.com/
2 PyChecker: herramienta para chequeo http://pychecker.sourceforge.net/
3 PyFlakes: herramienta para chequeo https://launchpad.net/pyflakes/

- alta y edición de de proyectos
- consulta de proyectos y métricas
- estimación básica de tamaño y tiempo (método PROBE de PSP)
- reportes de productividad, defectos y proyectos
- gráficos de distribución y regresión de tamaños, tiempos y defectos

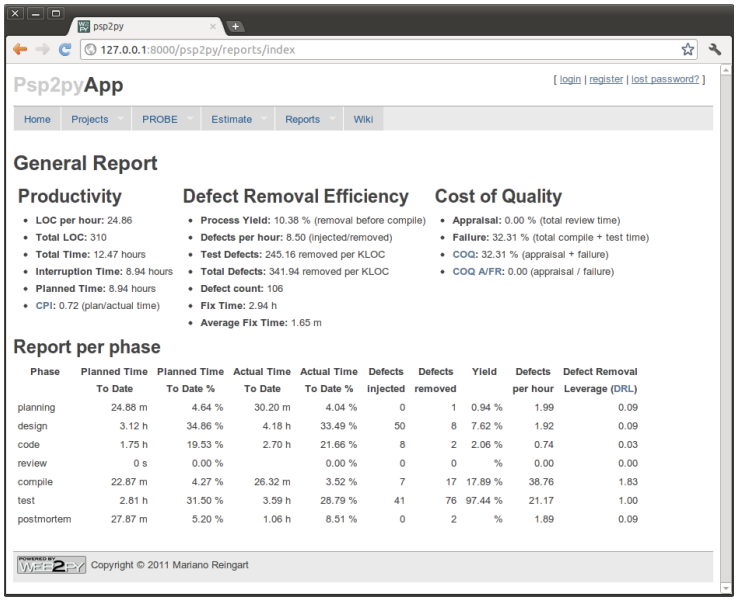

Figura 7: Captura de la página general de aplicación web (psp2py)

Se toma como referencia para la captura y análisis de métricas PSP, el sistema *Process DashBoard*. La integración de datos se realiza a través de una base de datos relacional.

El soporte al Proceso de Software Personal consiste en un listado de defectos y un grilla de tiempos en cada fase (módulo psp.py). Ambos son controlados por una barra de herramientas que permite iniciar, pausar y detener los cronómetros, elegir el proyecto y fase e ingresar defectos manualmente. Estos datos son almacenados en archivos persistentes de python y luego son sincronizados con un webserver independiente, que contiene la aplicación web de soporte PSP2PY, utilizando procedimientos remotos con notación javascript. De este modo se preserva la privacidad del desarrollador, eligiendo que datos y en que momento se envían, totalizándolos y pudiendo corregirlos.

Los errores de compilación y errores en tiempo de ejecución son automáticamente recolectados como defectos siguiendo las directivas del PSP. Estos errores son listados con un casilla de verificación y pueden ser editados (clic

izquierdo) mostrando un dialogo para tal fin. Al seleccionar un defecto (doble clic) este comienza a acumular el tiempo de corrección (fix time), hasta que el defecto es marcado como corregido (tildando la casilla correspondiente).

Para facilitar la detección de la fase de inyección de defectos y mejorar la calidad de las métricas recolectadas, el módulo PSP incorpora el manejo de metadata (información sobre el código fuente en sí). Estos datos constan de la última fase donde se introdujo o modificó cada línea de código, y se actualiza al completar cada fase del PSP. Luego, al detectarse un error, la herramienta busca en la metadata la linea de código que ha fallado determinando automáticamente en que fase fue escrita y consignandolo en el campo correspondiente ("inject_phase" en defectos) para un correcto seguimiento de las métricas del PSP, sin necesidad de que el usuario necesite recordar en que fase fue introducido el defecto. De ser un problema complejo originado en otra parte del código fuente, el defecto puede editarse para reflejar los datos necesarios.

Respecto a la ayuda en línea, el módulo PSP contempla la visualización de páginas wiki (formato similar al usado en enciclopedias colaborativas) para facilitar la documentación y navegación del proceso de software. Contiene la funcionalidad de una Guia Electrónica del Proceso (Electronic Process Guide o EPG) provistas por proyectos Similares (PSP Dashboard y/o Jasmine PSP Support Tool [19]).

4. Resultados Iniciales

La IDE y herramientas de soporte fueron usadas para desarrollar las tareas de programación para entrenamiento sugeridas por el libro sobre PSP [21], para reunir métricas y hacer una validación empírica básica de la hipótesis de esta investigación (verificando el modelo teórico y prototipo piloto, ver Tabla 1 y 2 en el Anexo A). Además, se validó la solución propuesta a través de las diferentes pruebas generales y de unidad. Dentro de estas, se utilizan los métodos de pruebas de caja blanca y pruebas de caja negra.

Si bien esta investigación no hace juicio de valor sobre la efectividad del PSP o RAD, cuestión que se entiende demostrada por los antecedentes y referencias, es menester analizar los datos e impresiones generales obtenidas luego de probar la herramienta experimental piloto.

Como medidas de productividad, la herramienta de soporte de PSP arrojó los siguientes resultados:

- **Total LOC:** 310 **Total Time:** 12.33 hours **LOC per hour:** 25.14
- **Interruption Time:** 8.94 hours **Planned Time:** 8.94 hours
- **CPI:** 0.72 (plan/actual time)

Si bien el tamaño y tiempo es subjetivo a cada individuo (dependiendo en gran medida del lenguaje de programación, en este caso Python) y resultó sensiblemente menor al encontrado en la bibliografía (por ej. en [21] se describe resultados de

estudiantes entre 2 y cuatro horas para producir 100 líneas de código en C++ o Pascal), su relación y tendencia es compatible (por ej. [21] cita 15 LOC/hora). Más adelante se analiza la correlación y significancia de dichos datos.

El valor 0.72 del índice de costo-productividad (CPI) indica un buen grado entre la relación del tiempo planeado y el tiempo consumido ([21], ideal acercándose a 1, demasiado conservador acercándose a 0)

Respecto a la Eficiencia para Remover Defectos, la herramienta calculó las siguientes mediciones:

- **Process Yield:** 10.38 % (removal before compile)
- **Defects per hour:** 8.60 (injected/removed)
- **Test Defects:** 245.16 removed per KLOC
- **Total Defects:** 341.94 removed per KLOC
- **Defect count:** 106 - **Fix Time:** 2.94 h - **Average Fix Time:** 1.65 m

El valor del rendimiento del proceso de eliminación de defectos (Yield) indica que se detectaron y removieron aproximadamente un 10% de los defectos antes de la fase de compilación. Dado que en esta prueba piloto inicial no se utilizo una fase de revisión formal y se realizó un chequeo informal antes de cada compilación, el valor obtenido es alto y comparable a un valor promedio (tomando como ejemplo un estudio de 12 alumnos gráficado en [21]). Se entiende que esto es una ventaja de la integración lograda y por usar pseudocodigo python, lo que posibilita detectar errores triviales de sintaxis en fases previas.

Los 8.6 defectos inyectados y eliminados por hora también están dentro de los parámetros normales (entre 1.31 y 9.43 dependiendo de la fase según una investigación en 36 programas de Pascal y 25 de C++ tabulada en [21])

El total de defectos por cada mil líneas de código puede aparentar ser un poco elevado, pero se encuentra en el orden de magnitud estándar (según se representa en un diagrama de programas de PSP en [21]). Números similares también pueden observarse en sistemas automatizados modernos para JAVA (por ej. en PSP Assistant [18] se detallan 314 defectos para el programa 6). Se entiende que estos números elevados corresponden a la buena calidad de las métricas ya que se recolectan automáticamente absolutamente todos los defectos (teniendo en cuenta de no volver a reportar duplicados) y también al relativo menor tamaño del código por usar el lenguaje python de alto nivel (lo que inevitablemente subirá la densidad de defectos comparado con lenguajes más tradicionales).

Cabe aclarar que, no se toman en cuenta las advertencias de formato como defectos, por ser violaciones menores al estándar de codificación que no influyen en la correctitud del código (tampoco se hace énfasis sobre ellas en la bibliografía).

Respecto al costo de la calidad, al no realizarse en esta prueba piloto una fase de revisión formal (ya que para la misma se necesita contar con los análisis realizados en

la presente investigación para determinar los defectos más comunes y las técnicas para prevenirlos), la medición de costo de apreciación (appraisal) arroja 0 por no tener datos de tiempos de revisión, por lo que solo se consigna el costo de fallas (failure, para fases de compilación y pruebas), siendo este el costo de calidad (COQ) para estos ejercicios, como se observa en los datos calculados por la aplicación:

- **Appraisal:** 0.00 % (review time) - **Failure:** 32.67 % (compile + test time)
- **COQ:** 32.67 % (appraisal + failure) **COQ A/FR:** 0.00 (appraisal / failure)

Igualmente, los valores están dentro de parámetros normales (tomando como ejemplo el estudio de 12 alumnos graficado en [21])

Dada las métricas recolectadas (SLOC vs Hs, Anexo A Tabla 1), la herramienta calculó el coeficiente de correlación $r^2 = 0.828075845371$. Según el autor ([21]) dicho índice se considera altamente predictivo dado la escala publicada. La Figura 8 grafica la regresión lineal analizada.

Para la prueba de significancia, la herramienta calculó el valor s = 0.00354622262343. Del mismo se encuentra que la significancia es alta, por lo que es poco probable que la relación de correlación sea una coincidencia (probabilidad inferior al 0.05 %) por los que los datos pueden ser considerados buenos dada la escala planteada ([21]).

Para la estimación de tamaño se utiliza la técnica de objetos próximos (PROBE), que consiste en aproximar el esfuerzo de desarrollo basado en la biblioteca de reuso (desarrollos previos). Si bien para la presente investigación se utilizaron funciones en vez de objetos, esto esta contemplado en la bibliografía ([21]), pero en este caso no es necesario normalizar los objetos por la cantidad de métodos ([21]).

Para juzgar los tamaños relativos de los objetos o funciones se usa la desviación estándar. Dado que los datos de tamaño no se distribuyen de manera normal (principalmente porque no varía de menos a más infinito, ya que los valores no están generalmente muy por sobre 0 como las alturas de una población), por lo que para superar dicho inconveniente es posible usar los logaritmos naturales de los datos, calcular la desviación estándar y media, y luego convertirlos a líneas de código con antilogaritmos ([21]).

Realizando los cálculos necesarios (efectuados automáticamente por la aplicación de soporte desarrollada) , se obtienen los siguientes parámetros:

1. Desviación estándar: 0.747612112404
2. Media (promedio): 2.30734135395

Los valores son similares a los presentados en el libro para 12 objetos de Pascal, 0.6346 y 2.802 respectivamente ([21]). A su vez, para comprobar que el logarítmo de los tamaños se distribuyen de manera normal y la categorización de tamaños basada

en los desvíos estándar sea útil para la estimación, la frecuencia de los rangos deberían observar la siguiente escala según la distribución normal ([21]). Al observar los datos calculados por la herramienta (Figura 9 según Anexo A Tabla 4, basada en datos recolectados en la Tabla 3) se constata que se distribuyen aproximadamente de forma normal (con un error mínimo debido a la limitada cantidad de muestras que afecta a los porcentajes), por lo que los datos podrían ser usados para estimación PROBE.

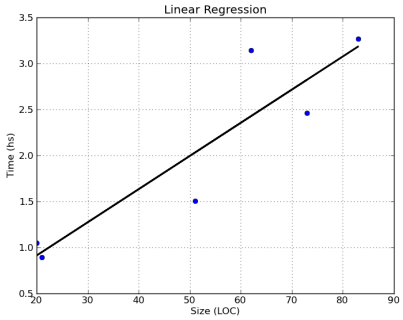

Figura 8: Regresión Lineal del tamaño (SLOC) y tiempos (en horas)

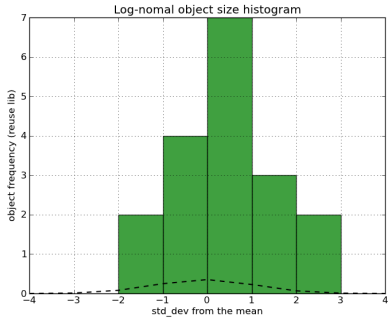

Figura 9: Histograma del tamaño de objetos (escala normal logarítmica)

Para el análisis de defectos, se comienza con el Reporte de Estandard Tipos de Defectos (Anexo A Tabla 5) generado automáticamente por la herramienta de soporte en base a las métricas recolectadas.

Dicho documento se basa en el PSP Defect Type Standard ([21]). El estándar ha sido adaptado para seguir las excepciones incorporadas del lenguaje Python, usando una convención tomada de Python Language Reference, y del chequeador automático, pep8.py**.** El tipo 30 (Build/Package) se movió dentro del 100 (Environment), para usar el tipo 30 como clasificación automática de los defectos de formatos PEP8 (tipo 22 en el estándar detallado [21])

Del análisis de los datos de defectos y experiencia de utilizar la herramienta de desarrollo se destaca que analizando la distribución de pareto (Figura 10), no se encuentran diferencias significativas con datos presentes en la bibliografía (por ej. respecto a defectos encontrados en fase de compilación de programas en C++, [21]). Lo mismo se observa de los tiempos promedios de correción (Figura 11), que son compatibles al datos de programas en Pascal o C++ ([21])

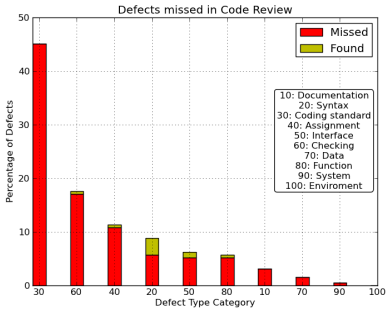

Figura 10: Distribución de Pareto de los defectos (fase compilación)

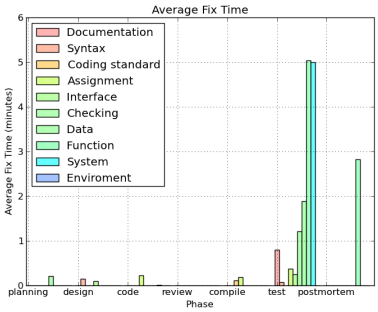

Figura 11: Tiempos promedio de corrección de errores

El Software Engineering Institute (SEI), a cargo de la dirección de gestión y fomento de la Iniciativas de PSP, ha puesto a disposición las directrices principales de una herramienta de soporte automatizado para PSP. De la Tabla 6, se puede resumir que la herramienta desarrollada cumple con todas las directrices y requisitos.

5. Conclusiones

Este trabajo busca resolver los riesgos comúnmente asociados al método RAD, relacionados con la baja calidad debido a la falta o abandono del seguimiento de un proceso de desarrollo por parte de los analistas y programadores.

Por lo tanto, se plantea utilizar el Proceso Personal de Software, PSP, para controlar el desarrollo y obtener de esta forma las métricas necesarias para evaluar la calidad y efectuar mejoras continuas.

Esto a su vez introduce nuevos desafíos respecto a la captura y análisis de las métricas y datos estadísticos, por lo cual se vuelve necesario desarrollar una marco teórico y herramienta que de soporte tanto al método RAD como al proceso PSP, automatizando los procesos manuales, facilitando y integrando las fases del desarrollo, mejorando de esta forma la calidad de las mediciones y por consiguiente de las métricas obtenidas, lo que posibilitará aumentar la productividad y disminuir la tasa de defectos.

Por ello, dado los beneficios del método RAD y del proceso PSP, se piensa que ambos son complementarios y solucionan recíprocamente las falencias de cada uno, posibilitando un desarrollo de software que minimice los tiempos y costos pero logrando alta calidad mensurable.

6. Futuras Líneas de Investigación

Luego de introducir los ajustes necesarios al prototipo inicial piloto sería posible llevar a cabo los experimentos útiles para recolectar los datos empíricos para

confirmar la hipótesis planteada a mayor escala en una futura investigación, que deberían traducirse en una mejora tanto en la productividad como en la calidad del trabajo de los desarrolladores de software.

6.1. Línea de investigación profesional

La selección de muestras podría ser relacionada al desarrollo de módulos de tres aplicaciones previas (Sistema de Facturación Electrónica 1KLOC; Sistema de Gestión de Emergencias 911 10KLOC; Sistema de Gestión Comercial 750KLOC).

Al ser proyectos de Software libre de Código Abierto, en un futuro es posible extender la investigación inicial sobre los mismos parámetros, con la colaboración de otros desarrolladores, para poder confirmar la hipótesis en una muestra mucho mayor.

6.2. Línea de investigación para equipos (TSP)

Si bien en esta investigación se enfocó en el PSP a nivel personal (desarrollador independiente), sería posible extender las herramientas para soportar el trabajo en equipo según TSP (Team Software Process), luego es aplicable para lograr acelerar la implementación de CMMI [27]. Esta tendencia la siguen otros proyectos similares, aunque hay que destacar que dicha implementación puede ser mucho más compleja que el presente trabajo y presentar nuevas implicancias y desafíos.

6.3. Línea de investigación educativa

Los ejercicios iniciales del PSP incluidos en el curso para ingenieros están desactualizados y no se ajustan a los lenguajes de programación actuales (Python en nuestro caso), comparado con los lenguajes tradicionales como ADA, C++ o Pascal usados en el libro, por lo tanto, otro curso de acción podría ser proponer un nuevo conjunto de ejercicios, especialmente diseñados para el aprendizaje de Python y PSP, que, de aplicarse en una población uniforme y controlada.

Esto podría ser útil para la recolección de datos empíricos y facilitar la adopción de sólidos conceptos de ingeniería (sobre todo entre profesores y alumnos), enseñando con una herramienta más amigable no solo técnicas de desarrollo, sino también un proceso de mejora continua a nivel personal.

Referencias

1. James Martin, (1991) Rapid Application Development; Macmillan Publishing Co., Inc.

2. Agarwal, R., et al (2000). Risks of rapid application development. Communications of the ACM 43, 11es (Nov. 2000)
3. Baltus, B et.al. , Software Quality: State of the Art in Management, Testing, and Tools; Springer
4. Watts S. Humphrey (2000) The Personal Software Process; Software Engineering Institute, Carnegie Mellon University
5. Will Hayes, James W. Over (1997) The Personal Software Process: An Empirical Study of Impact of PSP on Individual Engineers, Technical Report; Software Engineering Institute, Carnegie Mellon University
6. Disney, A. M. and Johnson, P. M. (1998) Investigating data quality problems in the PSP. In Proceedings of the 6th ACM SIGSOFT international Symposium on Foundations of Software Engineering (Lake Buena Vista, Florida, United States, November 01 - 05, 1998). SIGSOFT '98/FSE-6. ACM Press, New York, NY, 143-152
7. Johnson, P. M. et al (2003) Beyond the Personal Software Process: metrics collection and analysis for the differently disciplined. In Proceedings of the 25th international Conference on Software Engineering (Portland, Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 641-646.
8. D. Tuma. (2000) PSP dashboard. http://processdash.sourceforge.net/
9. Hackystat Development Site, http://www.hackystat.org
10. Python Software Fundation, The Python Programming Language, http://www.python.org
11. Greg Lindstrom (2005) Programming with Python; IT Professional, vol. 07, no. 5, pp. 10-16, Sept/Oct, 2005.
12. Paul F. Dubois, (2005) A Nest of Pythons; Computing in Science and Engineering, vol. 07, no. 6, pp. 81-84, Nov/Dec, 2005.
13. Stephen E. Cross (1998) Toward Disciplined Rapid Application Development, Department of Defense Software TechNews; Volume 2 Number 1 - Rapid Application Development (RAD) issue http://www.dacs.dtic.mil/awareness/newsletters/technews2-1/toc.html
14. Rico, David F. (2002) How to Estimate ROI for Inspections, PSPsm, TSPsm, SW-CMM®, ISO 9000, and CMMIsm, Department of Defense Software TechNews; Volume 5 Number 4 - Return-On-Investment from Software Process Improvement; http://www.dacs.dtic.mil/awareness/newsletters/stn5-4/
15. Rico, David F. (2004) ROI of Software Process Improvement: Metrics for Project Managers and Software Engineers; J. Ross Publishing http://davidfrico.com/
16. Rico, David F., What is the Return on Investment (ROI) of PSPSM (página web) http://davidfrico.com/roi-psppdf.htm
17. Abrahamsson, P., Warsta, J., Siponen, M. T., and Ronkainen, J. (2003). New directions on agile methods: a comparative analysis. In Proceedings of the 25th international Conference on Software Engineering (Portland,

Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 244-254.
18. Raymund Sison, David Diaz, Eliska Lam, Dennis Navarro, Jessica Navarro. Personal Software Process (PSP) Assistant. In Proceedings of APSEC'2005. pp.687~696; http://www.csie.ntut.edu.tw/labsdtl/95-summer/0823-1.pdf
19. Shin, Hyunil; Choi, Ho-Jin & Baik, Jongmoon. Jasmine: A PSP Supporting Tool, 73-83. Software Process Dynamics and Agility, International Conference on Software Process, ICSP 2007, Proceedings (Lecture Notes in Computer Science Vol.4470), Minneapolis, MN (May 2007). http://www.springerlink.com/content/1607282107g27767/
20. Hazrina Hassan, Mohd. Hairul Nizam Md. Nasir, and Shukor Sanim Mohd. Fauzi. 2009. Incorporating software agents in automated personal software process (PSP) tools. In Proceedings of the 9th international conference on Communications and information technologies (ISCIT'09). IEEE Press, Piscataway, NJ, USA, 976-981.
21. Humphrey, Watts S. A Discipline for Software Engineering. Reading, MA: Addison-Wesley, 1995.
22. Massimo Di Pierro, School of Computing, DePaul University. Web2py Enterprise Web Framework, 3rd Edition. Lulu.com. October 2010. ISBN 978-0557604142
23. Massimo Di Pierro. web2py for Scientific Applications. Computing in Science and Engineering, vol. 13, no. 2, pp. 64-69, Mar./Apr. 2011, doi:10.1109/MCSE.2010.97
24. Noopur Davis, Barbara Spencer. Experiences Using the Team Software Process at Adobe Systems. Presentations 2009 - Team Software Process Symposium. September 22nd, 2009
25. Alan Padula. TSP*-Agile Blend: The Gun Smoke Clears. Presentations 2009 TSP Symposium. September 21-24, 2009
26. Mukesh Jain. Delivering Successful Projects with TSP and Six Sigma: A Practical Guide to Implementing Team Software Process. Auerbach Publications, November 2008. ISBN 978-1420061437
27. 2010 TSP Symposium Proceedings Document. Carnegie Mellon University. Software Engineering Institute; http://www.sei.cmu.edu/tspsymposium/past-proceedings/2010/2010_TSP_Proceedings.pdf
28. Rappin, Noel; Dunn, Robin (2006) wxPython in Action, Manning

***Nota**: las páginas web fueron visitadas en el período de Agosto a Noviembre de 2006 y Diciembre de 2011*

Anexo A: Tablas

Project	Description	Plan Time	Actual Time	Int. Time	CPI	Plan. LOC	Actual LOC	Defects	Fix Time
Program 1A	Standard Deviation	1 h	53.65 m	1 h	1.1		21	16	23.15 m
Program 2A	LOC count	2.50 h	2.47 h	2.50 h	1.0		73	31	21.15 m
Program 3A	LOC / object / method count	1.25 h	3.14 h	1.25 h	0.4		62	46	59.20 m
Program 4A	Linear Regression Parameters	1.02 h	1.05 h	1.02 h	0.9		20	29	5.13 m
Program 5A	Numerical integration	2.03 h	3.27 h	2.03 h	0.6	48	83	41	59.08 m
Program 6A	Prediction Interval	1.14 h	1.51 h	1.14 h	0.7	23	51	30	18.72 m

Tabla 1: Resumen de métricas por proyecto (salida de psp2py)

Phase	Plann. Time	Plan. Time	Act. Time	Act. Time %	Defect injectd	Defects remo'd	Yield	Defect per hour	DRL
planning	24.88 m	4.64 %	30.20 m	4.08 %	0	1	0.94 %	1.99	0.09
design	3.12 h	34.86 %	4.18 h	33.87 %	50	8	7.62 %	1.92	0.09
code	1.75 h	19.53 %	2.42 h	19.65 %	8	2	2.06 %	0.83	0.04
compile	22.87 m	4.27 %	26.32 m	3.56 %	7	17	17.89 %	38.76	1.83
test	2.81 h	31.50 %	3.59 h	29.11 %	41	76	97.44 %	21.17	1.00
postmortem	27.87 m	5.20 %	1.20 h	9.73 %	0	2	%	1.67	0.08

Tabla2: Resumen de métricas por fase (salida de psp2py)

Object (function) Name	Size Range	Object LOC	Object ln(LOC)
test_prediction_interval	very small	3	1.09861228867
logical_to_physical_count	very large	41	3.7135720667
compute_integral	medium	12	2.48490664979
find_functions_and_classes	medium	12	2.48490664979
linear_regression	medium	10	2.30258509299
prediction_interval	large	15	2.7080502011
count_logical_lines	medium	12	2.48490664979
simpson_rule_tests	large	24	3.17805383035
simpson_rule_integrate	large	27	3.295836866
get_object	medium	9	2.19722457734
double_sided_student_t_probability	small	6	1.79175946923
count_logical_lines_per_object	very large	31	3.43398720449
double_sided_student_t_value	medium	12	2.48490664979
stddev	small	5	1.60943791243
factorial	small	5	1.60943791243
variance	small	5	1.60943791243
gamma	medium	7	1.94591014906
mean	very small	3	1.09861228867

Tabla 3: Datos de tamaños de objetos/funciones y rango (salida de psp2py)

Size Range	Range Midpoint LOC	Range Midpoint ln(LOC)	Quantity	Percentage
very small	2.25267213956	0.812117129138	2	11.11 %
small	4.75753292846	1.55972924154	4	22.22 %
medium	10.0476758992	2.30734135395	7	38.89 %
large	21.2201980507	3.05495346635	3	16.67 %
very large	44.8160161444	3.80256557876	2	11.11 %

Tabla 4: Datos de distribución de tamaños (salida de psp2py)

No	Name & Description	Qty.	Freq. %	Fix Time	Time %	Avg.
10	**Documentation**: Errors in docstrings and comments	6	3.11 %	4.78 m	2.57 %	47 s
20	**Syntax**: *SyntaxError* (spelling, punctuation, format) and IndentationError (block delimitation)	17	8.81 %	4.05 m	2.17 %	14 s
30	**Coding standard:** PEP8 format warnings and errors (long lines, missing spaces, etc.)	87	45.08 %	10.10 m	5.42 %	6 s
40	**Assignment**: *NameError* (undefined), unused variables, *IndexError*/*KeyError* (range/limits LookupError) and *UnboundLocalError* (scope)	22	11.40 %	17.45 m	9.36 %	47 s
50	**Interface**: *TypeError*, *AttributeError*: wrong parameters and methods	12	6.22 %	4.22 m	2.26 %	21 s
60	**Checking**: *AssertionError* (failed assert) and doctests	34	17.62 %	48.60 m	26.07 %	1.42 m
70	**Data**: *ValueError* (wrong data) and *ArithmeticError* (overflow, zero-division, floating-point)	3	1.55 %	5.65 m	3.03 %	1.88 m
80	**Function**: *RuntimeError* and logic errors	11	5.70 %	1.44 h	46.44 %	7.87 m
90	**System**: *SystemError* and Libraries or package unexpected errors	1	0.52 %	5 m	2.68 %	5 m
100	**Enviroment**: *EnvironmentError*: Operating system and build/third party unexpected errors	0	0.00 %	0 s	0.00 %	

Tabla 5: Reporte de Estándard de Tipos de Defectos (salida de psp2py)

Directiva	Recomendación del SEI	Este trabajo
Simplificar la recolección de datos	Datos automáticamente copiados entre formularios, sin necesidad de transcribirlos	La motiviación de este proyecto fue minimizar la sobrecarga al automatizar la recolección y análisis de datos.
Personalización del Usuario	Personalización (por ej. renombrar las fases).	Este proyecto es de código abierto, por lo que el usuario puede modificarlo.
Proceso estandar versus adaptado	Entendimiento completo del concepto y practicidad del PSP y su marco de trabajo antes de hacerle cambios para soportar una definición de proceso personalizado.	Este proyecto hace cumplir y enfatiza explícitamente las fases y elementos del proceso PSP estándar para familiarizar y facilitar los conceptos al usuario, a diferencia de otras herramientas "no intrusivas" por sensores.
Flexibilidad para trabajar en cualquier fase	Capacidad de operar en cualquier fase (a elección del usuario), independiente del ciclo de vida en cascada.	El usuario cambiar la fase actual en cualquier momento o seguir un ciclo de vida tradicional al ir validando cada etapa.
Ayuda en linea	El SEI recomienda que la herramienta automatizada tenga disponible ayuda en línea.	Este proyecto dispone de ayuda en linea (python) y posee funcionalidad para una Guia Electrónica de Proyecto (wiki).
Privacidad de Datos	Las metricas personales deben ser privadas, con la posibilidad para proveer acceso a los formularios completados (manteniendo seguro el perfil del usuario)	Este proyecto almacena las métricas actuales ende manera local (máquina del desarrollador), y las envía sólo cuando el usuario lo dispone (al contrario de otros proyectos por sensores que los envían continuamente sin ningún control).
Resumen de Plan de Proyecto Publicado	Project Plan Summary puede estar publicado basado en el marco de trabajo actual y la estructura de los formularios en sus diverentes niveles.	Este proyecto provee la posibilidad de publicar el resumen de proyectos y demás reportes especificados en la bibliografía del PSP (formato HTML estándar)
Tareas de Estimación	El PSP Requiere datos históricos disponibles (predecir defectos y tamaño), promoviendo el método PROBE.	Este proyecto contempla la funcionalidad necesaria para realizar estimaciones PROBE (incluso con intervalos de predicción)

Tabla 6: Análisis Comparativo de los "SEI Principle Guidelines"

Anexo B: Siglas y Abreviaturas

- **AISEMA**: Automated in-process software engineering measurement and analysis, análisis y medición automatizada dentro del proceso de ingeniería de software
- **CASE**: Computer Aided Software Engineering, Ingeniería de Software Asistida por computadora
- **CMM**: Capability Maturity Model, modelo de capacidad y madurez
- **CMMI**: Capability Maturity Model Integration
- **CMU**: Carnegie Mellon University
- **DSDM**: Dynamic Systems Development Method, método dinámico de desarrollo de sistemas
- **GUI**: Graphics User Interfase, interfaz gráfica del usuario
- **HTML**: HyperText Markup Language, lenguaje de marcado de hipertexto
- **I-CASE**: Integrated Computer Aided Software Engineering, Ingeniería de Software Asistida por computadora integrada
- **IDE**: Integrated Development Environment, entorno de desarrollo integrado
- **LOC**: Line Of Code, líneas de código
- **PROBE**: PROxy Based Estimation, estimación basada en aproximación (PSP)
- **PSP**: Personal Software Process, proceso de software personal
- **RAD**: Rapid Application Development, desarrollo rápido de aplicaciones
- **SEI**: Software Engineering Instituto, Instituto de Ingeniería de Software (CMU)
- **SLOC**: Source Lines Of Code, lineas de código fuente
- **TSP**: Team Software Process, proceso en equipos de software