Desarrollo rápido de software libre de alta calidad

Ingeniería de software asistida por computadora, enfocada en tareas, para agilizar el ciclo de vida de las aplicaciones, mediante mejora continua disciplinada a nivel personal

Mariano A. Reingart

reingart@uoc.edu
Universitat Oberta de Catalunya

Abstract. El presente artículo resume el estudio, desarrollo e implementación de un marco de trabajo teórico/práctico para creación y mantenimiento ágil de software libre, orientado al desarrollo de aplicaciones empresariales centradas en datos (principalmente transaccionales, con bases de datos relacionales, lenguaje de programación dinámico e interfaces web y visuales). Se analiza el estado del arte respecto a proyectos libres y abiertos (producción por pares, fomento, motivación, diversidad, gobernanza, calidad, etc.) y se exploran técnicas para migración de lenguajes legados discontinuados (VB clásico, usando técnicas EBNF). Se incluye el desarrollo de artefactos de software: herramientas integradas para ingeniería asistida por computadora (I-CASE), contemplando la recolección automática de datos estadísticos. Se proponen varias mejoras al estado del arte respecto a mayor precisión en la medición de tiempos (detección facial vía WebCam) y seguimiento de líneas de código por identificadores globales (UUID); presentando los resultados preliminares de una prueba piloto (experiencia personal investigación-acción), para validación cualitativa y cuantitativa inicial, relevancia, implicaciones y trabajos futuros.

Palabras clave: ingeniería de software libre / código abierto, métricas, calidad, python, postgresql, web2py, wxwidgets

1 Introducción

En el futuro cercano, los investigadores todavía tienen muchas preguntas intrigantes que explorar: ¿Aplica la Ley de Brooks al desarrollo de código abierto, bajo cuales circunstancias, o por qué no? ¿Cuales son las implicancias? ¿Es el desarrollo de código abierto una manera de producir software mejor y más eficiente, o es un enorme esfuerzo gastado invisiblemente? ¿Como podemos estimar el esfuerzo real que se consuma en un proyecto open source, y como podemos medir el impacto de la comunidad rodeando al núcleo de desarrolladores? ¿Cuales diferencias pueden ser encontradas sobre la organización, procesos y productos, y cómo se relacionan con el éxito del proyecto? Si consideramos que la ingeniería de

> *software ha estudiado el desarrollo por décadas, y que muchos proyectos aún están experimentando problemas al punto de fracasar completamente, la ingeniería de software open source será un tema interesante en los años venideros...*
>
> Traducido de Koch S. & Gonzalez-Barahona J. M. (2005). **Open Source software engineering: The state of research**. Publicado en "First Monday's Special Issue #2: Open Source" [1] (peer review journal on internet)

1.1 El contexto de la ingeniería de software libre:

Esta investigación tiene como eje las propuestas de Brooks que hizo en su ensayo "No Silver Bullets"[2] acerca de si se podría encontrar una bala de plata para poder superar la crisis del software. El autor propone:

- Evitar construir lo que puede ser adquirido
- Usar prototipado rápido
- Hacer crecer el software orgánicamente
- Desarrollar a los grandes diseñadores

Entendiendo que el software libre presenta muchas de estas características, en base a estos puntos se realizó el análisis del estado del arte para encontrar un marco metodológico y proponer una herramienta superadora.

1.2 Motivación y antecedentes

Una de los propósitos de este proyecto es aplicar los lineamientos y mejorar las deficiencias conocidas del PSP-BOK[3] (Personal Software Process – Body of Knowledge), que junto con el Team Software Process del SEI forman un conjunto de prácticas disciplinadas de la ingeniería del software. Sus elementos podrían ser aplicables a una certificación de calidad de software CMMI nivel 5 que serían de utilidad en ciertos contextos (por ej. la Ley 25.922 de Promoción de la Industria del Software en Argentina).

Se exploran investigaciones modernas como el proyecto Mylyn[4] (plugin para el IDE Eclipse), y tendencias recientes Agile ALM (Application Lifecycle Management[5]), que buscan integrar la gestión ágil del ciclo de vidas de las aplicaciones.

---

[1] http://firstmonday.org/ojs/index.php/fm/article/view/1466/1381
[2] http://es.wikipedia.org/wiki/No_hay_balas_de_plata
[3] http://www.sei.cmu.edu/tsp/tools/bok/
[4] http://www.eclipse.org/mylyn/
[5] https://en.wikipedia.org/wiki/Application_lifecycle_management

Al contemplar herramientas de desarrollo rápido de aplicaciones, se consideran migraciones de lenguajes legados como Visual Basic Clásico (5, 6) / Visual FoxPro (discontinuados por su proveedor), analizando un Conversor EBNF (Notación Extendida Backus-Naur ISO / IEC 14977).

En artículos anteriores[6] se realizó un acercamiento inicial a las las metodologías, procesos, herramientas, lenguaje de programación (Python), base de datos (PostgreSQL) y framework web (web2py) y toolkit gráfico (wxPython), contemplados en el presente trabajo. También se publicó un artículo[7] sobre el desarrollos del proyecto "PyAfipWs[8]" de software libre que analiza cuestiones relevantes ampliadas en el presente documento.

2 Estado del arte

2.1 Modelo de Producción del Software Libre

2.1.1 Fomento y motivación de los desarrolladores:

Ciertos estudios (Rossi, 2006) encuentran que el software libre sería superador "homo œconomicus", donde no sólo se buscaría una motivación material, sino también recompensas extrínsecas: reputación y reconocimiento pares, solución necesidades propias -scratching an itch-, aprendizaje/mejoramiento skills, lo que favorecería la libre revelación de innovaciones.

Además se han encontrado motivaciones intrínsecas: diversión y autodeterminación, que en conjunto posibilitaría una mayor diversidad para evitar la "tragedia de los comunes[9]" (disputas entre personas bien intencionadas y con razonamientos lógicos sectorizados que destruyen un bien compartido).

También existen dificultades (Feller & Fitzgeral, 2003) por un elaborado sistema de tabúes y costumbres (normas "esotéricas", disputas de ego, etc.) lo que conlleva a una menor tolerancia a novatos y principiantes en las comunidades de software libre.

---

[6] Reingart, Mariano. **Plataforma de Desarrollo Rápido de Aplicaciones bajo el Proceso de Software Personal: en búsqueda de agilidad, solidez y disciplina para la Ingeniería de Software.** 41° JAIIO - EST 2012 - ISSN: 1850-2946 – Pg. 344 - 367 http://41jaiio.sadio.org.ar/sites/default/files/17_EST_2012.pdf

[7] Reingart, Mariano. **PyAfipWs: facilitando, extendiendo y liberando los Servicios Web de AFIP (Factura Electrónica ...)** 41° JAIIO - JSL 2012 - ISSN: 1850-2857 Pg. 164-178 http://41jaiio.sadio.org.ar/sites/default/files/15_JSL_2012.pdf

[8] https://www.github.com/reingart/pyafipws

[9] https://es.wikipedia.org/wiki/Tragedia_de_los_comunes

2.1.2 Crecimiento orgánico: modularización y gobernanza:

Siguiendo el análisis inicial (Rossi, 2006), el desarrollo de FOSS es un caso exitoso no solo por la capacidad de atracción de programadores altamente capacitados y motivados a lo largo del tiempo, sino también por la calidad del producto final, rapidez de desarrollo y bajos costos.

Si bien esta planteada la discusión sobre la caracterización "catedral vs. bazar[10]" (descentralización de decisiones ex-ante, diseño concurrente, integración de usuarios, auto-selección según habilidades), todo esto facilita un proceso desarrollo paralelo larga escala vs. el "proceso producción fabril" más tradicional de la industria propietaria. A su vez, se aprovecha la inteligencia de la comunidad, por lo que el ritmo de desarrollo es determinado por el miembro más productivo. Pero por otro lado, la mayoría de los proyectos cuentan con pocos desarrolladores y el contacto entre comunidades suele ser bajo, por lo que algunos autores lo caracterizan como una "cueva" en vez de "comunidad" (no hay una "red plana de pares interactuando").

También hay un división del trabajo entre contribuidores heterogéneos: núcleo de desarrolladores, desarrolladores ocasionales y usuarios. Esto generaría la existencia de barreras que dependen de los siguientes factores claves que deben ser superados:

- la facilidad de la programación y modificación de los módulos
- el grado de independencia entre los módulos
- la libertad de elección del lenguaje de programación
- la posibilidad de "conectar" el módulo a la arquitectura

Respecto a la gobernanza, se encuentran principalmente esquemas de "dictadura benevolente" o "comité de votación", por lo que imagen anárquica del FOSS no es adecuada: es esencial el liderazgo y autoridad (reputación, lealtad e identificación, lo que acompleja la coordinación en un contexto de colaboración voluntaria, especialmente para impedir bifurcaciones y duplicación del trabajo).
Ciertos estudios (Holck & Jørgensen, 2005) analizan la delegación de "lugartenientes" (modularización), pasando de proyectos en solitario ("solo") a proyectos jerarquizados con "dueños de módulos" o "mantenedores", con el objetivo de balancear anarquía con control: ¿cuánta estructura se puede imponer sin desencantar a los colaboradores voluntarios?
Uno de los factores motivacionales fundamentales es ver rápidamente los resultados del trabajo, por lo que la burocracia excesiva puede ser contraproducente ("integración big bang"), y por el contrario, un control relajado puede producir versiones inestables.

[10] https://es.wikipedia.org/wiki/La_catedral_y_el_bazar

Por estos motivos, se propone en esta investigación un marco de trabajo simple, enfocado a nivel de programadores individuales (PSP), contemplando mecanismos de planificación y control adecuados (apoyados por herramientas visuales mejoradas y bibliotecas livianas). Con ello se busca favorecer la viabilidad y concurrencia del desarrollo de aplicaciones modulares, aún en grupos reducidos.

2.1.3 Alternativa a la disyuntiva entre "construir" o "comprar":

Debido a la complejidad creciente del software, la mayor parte del costos surgen de las pruebas, depuración y mantenimiento (no del diseño y codificación). El software empaquetado parece haber alcanzado límites significativos (30% en inversión de software), y las empresas propietarias no han podido cumplir adecuadamente las necesidades de una porción sustancial del mercado.
Por ello, (Bessen, 2006) propone un modelo de coexistencia alternativo complementario "a medida", especialmente útil para aquellos clientes con capacidades de auto-desarrollo, como solución intermedia al dilema de comprar vs desarrollar (concibiendo medios socialmente más eficientes para obtener el software que se necesita).
Retomando el análisis anterior (Rossi, 2006), el FOSS podría describirse como un modelo de innovación privada-colectiva, expandiéndose a modelos de "producción entre pares basada en bienes comunes", en contraste con los modelos de producción tradicionales.
Por ello con este trabajo se busca posibilitar un un mercado más horizontal y diverso, con "prosumidores[11]", donde cada usuario pueda ser un potencial desarrollador / depurador.

2.1.4 Posibilidades de prototipado rápido de calidad (no descartables):

El ciclo de vida del software libre de código abierto es diferente al tradicional (Feller & Fitzgeral, 2003). La codificación (primera fase) es la actividad sine qua non, muchos colaboradores temen enviar su código a revisión, pero también es un incentivo real para mejorar la calidad.
Si bien la revisión es un aspecto fundamental, se presenta una paradoja: cuanto más simple es el código, mas revisiones tendrá, y generalmente será difícil obtener revisiones del diseño. Según el postulado de la Ley de Linus: "Dado un número suficientemente elevado de ojos, todos los errores se convierten en obvios", pero recientes sucesos como goto fail de Apple y Heartbleed presentan cuestiones a este tema.

[11] Acrónimo de Productor + Consumidor, https://es.wikipedia.org/wiki/Prosumidor

2.2 Proceso de Software Personal (PSP)

Continuando con la revisión del estado, presentamos el Proceso de Software Personal: un conjunto de prácticas disciplinadas para la mejora continua a nivel de los desarrolladores, que podría aportar soluciones a la calidad (defectos tendientes a cero) y mejoras en la estimación (tiempos y recursos), entre otras métricas que podrían ser útiles para la toma de decisiones en proyectos de software libre.

El inconveniente que presenta es el problema de adopción: los desarrolladores que lo aprenden, no lo usan en la práctica, principalmente por dos dificultades:

1. Sobrecarga de trabajo: es complicado recolectar todas las métricas que solicita (tiempos, defectos, líneas de código)
2. Cambio de contexto: con otras herramientas externas al entorno de desarrollo, por ej. llevando planillas de cálculo.

2.2.1. Recolección automática de métricas:

Para solucionar el problema de la sobrecarga, siguiendo investigaciones que buscan una herramienta superadora e integrada (Choi, Sang-Hun., Syed, Ahmad, Hyun-Il & Young-Kyu, 2013), en esta investigación se propone una enfoque completamente integrado, incluyendo el uso de un sensor de detección facial, para detectar automáticamente si el desarrollador está o no frente a la computadora. Con esta técnica además se mejoraría la calidad de las métricas que se recolectan (pudiendo detectar interrupciones y otros tiempos no relacionados al desarrollo).

2.2.2. Dificultades en la fase de Diseño:

Si bien hay otras investigaciones que también buscan atacar problemas más profundos del PSP, como las complejidades de su fase de diseño (Chaiyo, 2013), para este proyecto se trató de simplificarlo aún más: directamente hacer el diseño en pseudo-código, contemplando los análisis anteriores, que identifican la codificación como una de las actividades principales del software libre.

2.2.3. Reconciliación procesos disciplinados y metodologías ágiles

Por último, la idea también es reconciliar el PSP con metodologías ágiles (más ligadas al desarrollo de software libre). En este sentido, hay investigaciones como SCRUM-PSP (Rong, Shao & Zhang, 2010) que han demostrado que ese camino en viable. Si bien para este proyecto en vez de SCRUM se utiliza RAD, desarrollo rápido de aplicaciones, los conceptos e implicancias serían similares.

2.3 Desarrollo Rápido de Aplicaciones (RAD)

Justamente el área del desarrollo rápido de aplicaciones, si bien no hay muchas novedades modernas, si hay gran cantidad de investigaciones sobre las herramientas para el desarrollador, IDE, en especial Eclipse con un plug-in como Mylyn (Goth, 2009) que tienen enfoques para permitir realizar minería de datos, sobre defectos, tiempos, etc.; tendientes a analizar y luego mejorar el comportamiento de los programadores.
En este sentido se puede citar:

- Estudio desarrolladores Java que utilizan la IDE Eclipse → Mylyn (Murphy, Kersten, Findlater 2006)
- Enfoques sociales a la SWE, detectando "patrones de tareas" (Schmidt & Reinhardt, 2009)
- Incorporación de herramientas de asistencia al desarrollador -StackOverflow- (Bacchelli & Lanza, 2012)
- Estudios empíricos sobre patrones de edición (Zhang, Khomh, Zou, Hassan 2012)

2.4 Mantenimiento y evolución del Software

Relacionado con estos temas y para finalizar la sección del estado del arte, se relevó cual es la tendencia en mantenimiento y evolución del software, contemplando también la migraciones de aplicaciones legadas, que se resumen a continuación.

2.4.1. Migración de Aplicaciones Legadas (VB) TXL, EBNF

Desde el punto de vista comparativo, se puede analizar el experimento en conversión automática de Java a C# (ElRamly, Eltayeb & Alla 2006), ya que utiliza un proyecto reconocido (TXL 82) tanto en el ambiente académico como en la industria. Muchos de los criterios y consideraciones son aplicables, y de hecho las similitudes y diferencias entre Visual Basic/Python son parecidas a las de los lenguajes Java/C# (aunque un poco más acentuadas). Dado que TXL es un proyecto no libre y su lenguaje es altamente específico, más complejo por ser funcional, en la presente investigación se plantea utilizar Python (lenguaje imperativo de propósito general) para escribir las reglas de transformación, compartiendo el enfoque la transformación basada en EBNF (ver proyecto vb2py[12]).

2.4.2. Seguimiento de líneas de código fuente (LHDiff)

Rastrear el origen y variaciones de las líneas de código de un programa es fundamental para resolver ciertos problemas como: localizar la introducción de bugs,

[12] http://vb2py.sourceforge.net/

detectar clones o defectos en distintas versiones y análisis de evolución de software, actualmente realizadas con algoritmos de comparación de texto (Asaduzzaman, Roy, Schneider 2013). Estas técnicas tienen sus ventajas y desventajas (rendimiento variable -especialmente en grandes bases de código-, dificultades para detectar ciertos tipos de cambios, etc.)
Si bien para esta investigación se propone un enfoque más directo (integrando la detección de la ubicación de las líneas dentro de la propia herramienta de desarrollo), el artículo mencionado es útil como punto de partida, para analizar el estado del arte y tener un marco de comparación para evaluar este tipo de técnicas (especialmente para detectar posibilidades de mejoras).
Cabe aclarar que tener un mapeo correcto de las líneas de código, incluyendo en qué fase fueron introducidas y/o modificadas, es fundamental para la recolección automática de métricas confiables, tanto de tamaño como de defectos, para el Proceso de Software Personal.

3 Marco Teórico de Trabajo

En base al estudio bibliográfico, se propone para este trabajo un marco teórico superador, contemplando tanto el fomento de las ventajas analizadas, como la mitigación de las dificultades que se trataron en la sección anterior.

Básicamente, la solución consta de integrar la herramienta de desarrollo, simple, poderosa y de fácil uso, para facilitar su utilización a los usuarios principiantes y contemplar también las necesidades de los usuarios más avanzados.
Esto incluye bibliotecas livianas y un marco de trabajo concreto y conciso, para promover el desarrollo rápido de aplicaciones modulares de software libre, posibilitando la delegación de tareas y su posterior integración. El enfoque es a nivel de programadores individuales, basados en los procedimientos y métodos del PSP. A la vez, se incluyen mecanismos de control y planificación, para poder recolectar datos empíricos y estadísticos.

Se busca favorecer la concurrencia y viabilidad, aún en grupos reducidos y/o de poca experiencia inicial; propiciando un mercado laboral mas horizontal y diverso; contemplando que el trabajo se visualice rápidamente, con aplicaciones y herramientas visuales, facilitando la tarea y motivando al desarrollador; sin perder de vista el seguimiento y las prácticas disciplinadas como PSP para mejorar la calidad.

Por consiguiente, se intenta facilitar el mantenimiento y la toma de decisiones de arquitectura, para evitar los conflictos (especialmente en proyectos de software libre), con datos más empíricos y estadísticos relevantes.

Se espera que esto mejore las oportunidades comerciales y minimice las incertidumbres en relación a la planificación de desarrollos (tiempos y recursos).

La idea principal es cubrir determinadas necesidades insatisfechas del mercado, según las investigaciones relevadas, especialmente en software libre, donde aún hay nichos por avanzar.

4 Desarrollo

4.1 Metodología

Se utilizó para esta investigación la metodología Action Research[13] (Investigación-Acción), con cuatro ciclos y ajustes en cada iteración. En principio se participó en el programa de becas para estudiantes "Google Summer of Code[14]" (específicamente en la propuesta "Improve wxQt port and wxPython for Android proof of concept" disponible online[15]). El mismo consistió en trabajar en un proyecto de software libre real, con un mentor, cuestión que fue muy provechosa, pudiendo comprobar en la práctica muchos aspectos analizados en la teoría relevada.
En particular, se utilizó la variante Living Theory[16] de Action Research, que se enfoca en como mejorar la propia práctica.

4.2 Experimentos

Para los experimentos, en un primer momento se pensó construir un aplicativo de Facturación Electrónica, justamente usando las herramientas desarrolladas. Al avanzar, por diversas razones explicadas en el informe completo, se decidió modificar el plan inicial (en sintonía con la metodología investigación-acción), desarrollando módulos e interfaces para el organismo de recaudación de impuestos de Argentina, dado que justamente el objetivo era aplicar Action Reseach en un tema concreto y real.

[13] http://cadres.pepperdine.edu/ccar/define.html
[14] https://developers.google.com/open-source/gsoc/
[15] http://www.google-melange.com/gsoc/proposal/public/google/gsoc2014/reingart/5629499534213120
[16] http://www.jeanmcniff.com/items.asp?id=19

5 Resultados

5.1 Resultados (software)

A continuación se describe el prototipo desarrollado (Figura 1) denominado rad2py[17], que si bien ya había sido iniciado con anterioridad en otra investigación, fue continuado y mejorado con nuevas características que se describen a continuación:

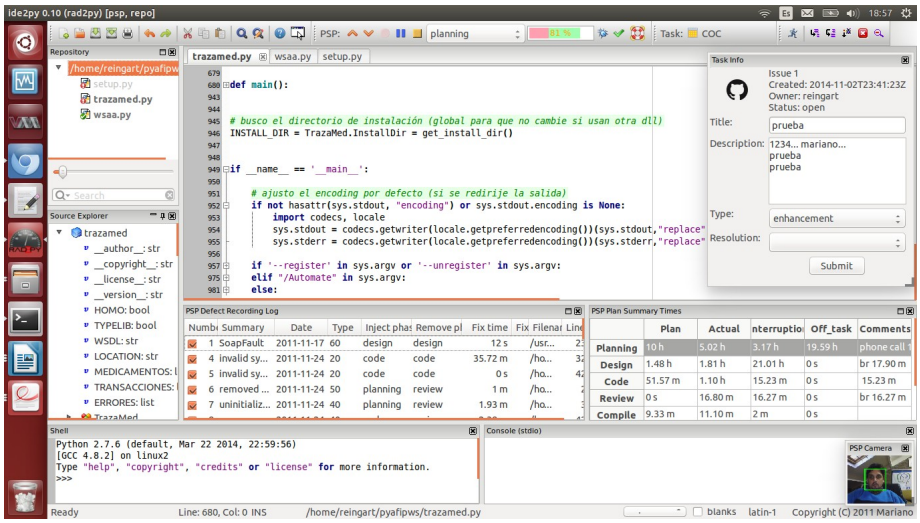

Figura 1. Prototipo IDE con las principales características desarrolladas

Todos los errores que son detectados automáticamente por la herramienta (o ingresados manualmente), se registran en la bitácora de defectos (panel "PSP Defect recording Log"), donde al seleccionar un elemento indica en que línea sucedió (además de la fase y otros datos relevantes).

En el panel de repositorio (Figura 2), es posible ver como se resaltan los archivos de interés, incluso ocultando los archivos no utilizados para la tarea, dependiendo del umbral de relevancia (calculado en base al tiempo dedicado a cada archivo).

[17] https://github.com/reingart/rad2py

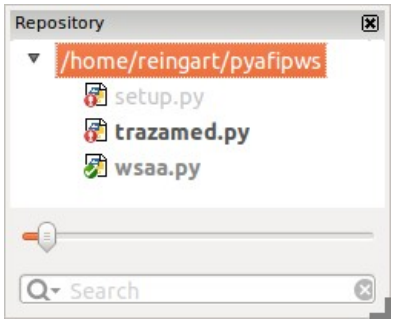

Figura 2: Repositorio con destacado automático según la actividad

El conector de GitHub (Figura 3), permite ir haciendo los cambios en el ticket y enviándolos al repositorio, controlando el titulo, descripción y otras características básicas, sin necesidad de que el usuario cambie de herramienta hacia un navegador.

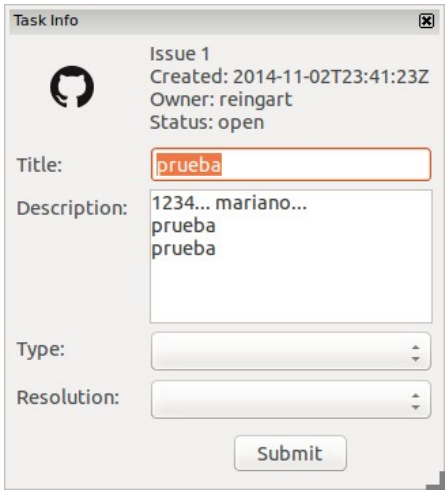

Figura 3. Conector con GitHub

Al utilizar UUID (identificadores globales, ver metadatos en la Figura 4), estos no cambiarán si se agrega o borra una linea; ya que automáticamente se actualiza la estructura interna que vincula el identificador con el número de linea físico.
Este código de identificación global va a permitir en el futuro saber donde se ubicó cada linea históricamente sin necesidad de hacer comparaciones (diff). Ello permite recordar cual fue el código relevante de cada tarea, para luego mostrarlo o no al desarrollador (aún cuando se produzcan mezclas de código en el repositorio, e incluso si hay cambios de varios programadores sobre el mismo archivo).

Adicionalmente, se puede identificar más fácilmente la fase en que se introdujeron o removieron los defectos, incluso frente a cambios sustanciales en el código fuente, que confundirían a herramientas tradicionales (basadas en el cálculo de diferencias textuales).

Gracias a estas características, el código fuente también se oculta automáticamente en el editor (folding), recordando que fragmentos eran visibles, puntos de interrupción y temas similares, para mostrar solamente las porciones relevantes a la tarea actual abierta, en la cual esta trabajando el programador.

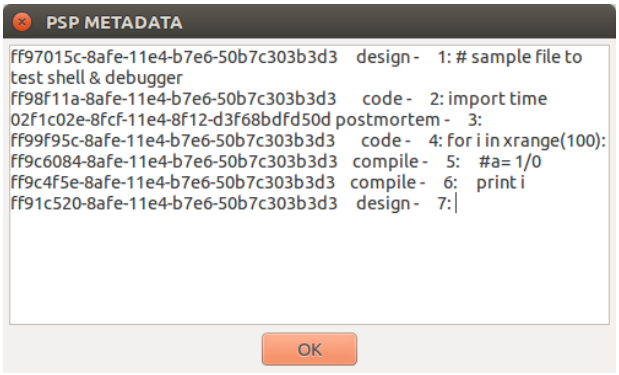

Figura 4. Metadatos PSP (UUID, fase y linea de código)

Respecto al control de los tiempos, se indica en la barra de progreso del PSP (Figura 5), y el sensor detecta el rostro del programador para diferenciar automáticamente los tiempos relevantes (verde) de las interrupciones (en rojo).

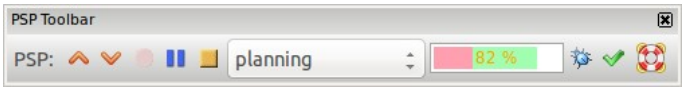

Figura 5. Barra integrada para el PSP

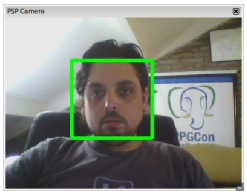

Figura 6. Sensor de detección facial

Si se mira a la pantalla, y por ende a la cámara (Figura 6), se irá incrementando el tiempo real dedicado a esta tarea (al detectar el rostro del programador). De lo contrario, se computa como tiempo de interrupción. El sensor es riguroso, en el sentido de que si el programador no está viendo a la pantalla directamente, por ej. de costado o atendiendo el teléfono, no detecta el rostro. A su vez, el fraccionamiento es pequeño (pocos segundos), por lo que se tendría una métrica más precisa.

5.2 Resultados (prueba)

Tabla 1. Reporte de resumen de proyectos PSP

Project	Plan Time	Actual Time	Int. Time	CPI	Plan LOC	Actual LOC	Defects	Fix Time
Issue #1	8.58 h	1.93 h	8.58 h	4.44			16	15.30 h
Issue #2	4 h	4.50 h	4 h	0.89			6	43.23 m

Tabla 2. Reporte de tipos estándar de defectos PSP (Defect Type Standard Report)

N°	Nombre	Descripción	Cant.	Freq. %	Tiempos		
					Fix.	%	Prom.
10	Documentación	Errors in docstrings and comments	0	0.00 %	0 s	0.00 %	
20	Sintaxis	SyntaxError and IndentationError ...	10	45.45%	39.50 m	4.11 %	3.95 m
30	Estandar de codificación	PEP8 format warnings and errors (long lines, missing spaces, etc.)	0	0.00 %	0 s	0.00 %	
40	Asignación	NameError (undefined), IndexError/KeyError (range/limits LookupError) and UnboundLocalError (scope)	7	31.82 %	15.60 m	1.62 %	2.22 m
50	Interfaz	TypeError, AttributeError: wrong parameters and methods	1	4.55 %	15 h	93.66 %	15 h
60	Chequeo	AssertionError (failed assert) and doctests	0	0.00 %	0 s	0.00 %	
70	Datos	ValueError (wrong data) and ArithmeticError (overflow, zero-division, floating-point)	0	0.00 %	0 s	0.00 %	
80	Función	RuntimeError and logic errors	4	18.18 %	5.85 m	0.61 %	1.45 m
90	Sistema	SystemError and Libraries or package unexpected errors	0	0.00 %	0 s	0.00 %	
100	Entorno	EnvironmentError: Operating system and unexpected errors	0	0.00 %	0 s	0.00 %	

Para obtener los resultados de la prueba de concepto, se utilizó el prototipo experimental comentado anteriormente. Se desarrollaron dos tickets[18]: interfaces para los webservice de AFIP. Se ingresaron los tiempos de planificados y luego, la herramienta fue recolectando los tiempos reales y de interrupción, calculando el porcentaje. En la Tabla 1, puede observarse que se sobrestimó demasiado en la primer tarea, y en la segunda tarea, al tener mínimamente datos históricos, se pudo ser más precisos con las estimaciones y disminuir los defectos, para poder mejorar la calidad del software en cuestión.

Hay algunas incidencias relacionadas a la consistencia de los datos (pueden observarse en la Tabla 2), que están explicados detalladamente en el informe completo, pudiendo deberse a bugs que habrá que revisar y corregir de ser necesario.

La Figura 7 muestra el agregado de los defectos ordenado por frecuencia, indicando los detectados (generalmente automáticamente) antes de la fase de revisión. En la Figura 8 se observan los tiempos de corrección promedios por fase. Si bien los datos recolectados son acotados (principalmente para realizar una prueba inicial del prototipo), y en ambos casos los errores en los tiempos fueron ajustados, se observa que guardan cierta relación y consistencia con la bibliografía del PSP[19].

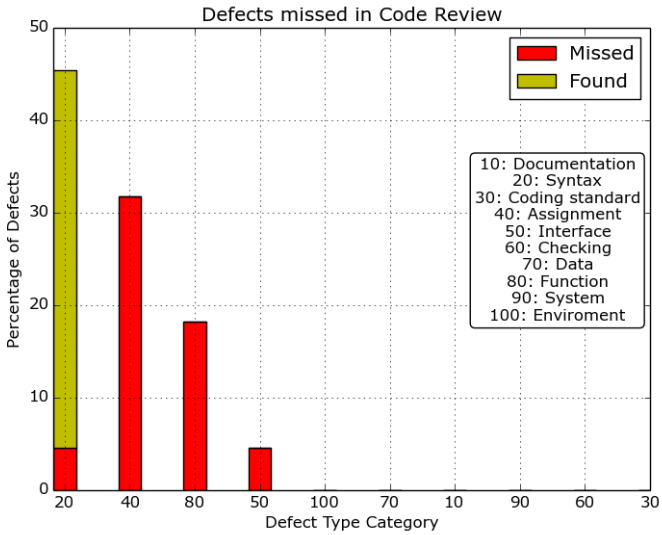

Figura 7. Distribución de pareto (defectos)

[18] https://github.com/reingart/pyafipws/issues
[19] http://www.sei.cmu.edu/reports/09sr018.pdf

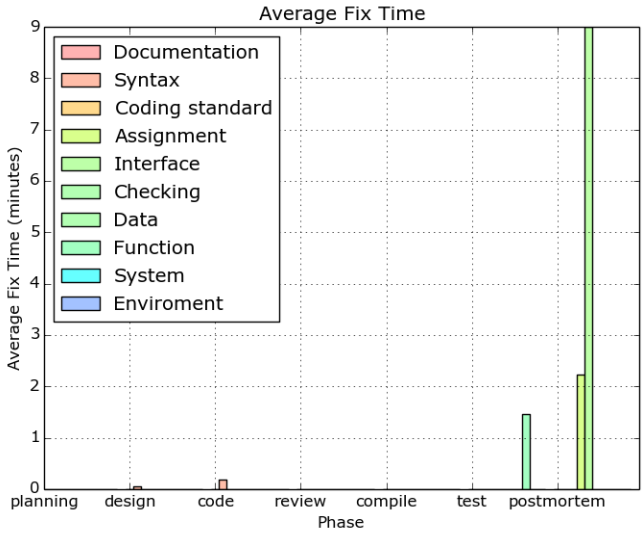

Figura 8. Tiempos promedios de corrección

6 Discusión

En principio, la investigación buscó mejorar las técnicas y herramientas de desarrollo rápido de aplicaciones de software libre. Para ello se realizo un análisis biográfico, proponiendo un marco teórico para sustentar una base más sólida sobre las cual avanzar. La idea fue superar las deficiencias conocidas del RAD (desarrollo rápido de aplicaciones), los problemas de adopción del PSP (Proceso de Software Personal) y también los desafíos del software libre de código abierto, como se explicó en las secciones precedentes.

A partir del disparador "No Silver Bullets", se estructura un relevamiento del estado del arte, justificado el marco teórico desarrollado desde la perspectiva de software libre. Se consideraron investigaciones recientes relativas a las metodologías ágiles (SCRUM), así como las últimas tendencias en herramientas de desarrollo, migración, seguimiento y evolución del software.

Se hace un énfasis especial en los conceptos del Proceso Personal de Software (PSP) -requisito previo del Team Software Process[20] (TSP), basado originalmente en el CMM nivel 5-, el cual es desarrollado y utilizado por importantes centros de

[20] http://www.sei.cmu.edu/tsp/

investigación como el Software Engineering Institute (SEI de Carnegie Mellon), entre otras universidades, tanto en ambientes académicos como laborales. De hecho varias iniciativas gubernamentales y empresariales alrededor del mundo lo contemplan para mejorar la calidad del software e incrementar la productividad del equipo, reduciendo los costos y tiempos de desarrollo.

Si bien los datos recolectados son acotados por el tipo de investigación y su alcance, se utilizan procesos y metodologías ampliamente analizadas por otras investigaciones científicas, donde existen juegos de datos y varias estadísticas que los avalan.

Por ende, la contribución principal de este trabajo no es innovar con un nuevo proceso de software, sino poder entender y aplicar las mejores prácticas de la industria al desarrollo de software libre, que presenta sus propias dificultades e implicaciones.

6.1 Relevancia

Los aspectos relevantes de la investigación son, por un lado, intenta superar la obsolescencia de lenguajes legados (por ej. VB) y otras herramientas discontinuadas incluso de software libre (PythonCard[21], WinPDB[22] o Boa Constructor[23]), que no están siendo mantenidos activamente, o de las cuales no hay mucho interés debido a su complejidad y obsolescencia, por lo que se exploran investigaciones modernas en estos sentidos.

También se hacen aportes originales para intentar resolver algunas cuestiones del estado del arte, como ser: identificadores únicos para lineas de código (UUID) para facilitar y simplificar su seguimiento (útil para registrar la relevancia de las mismas por cada tarea); y el sensor de cámara web para recolectar los tiempos de manera más precisa (utilizando reconocimiento facial).

6.2 Limitaciones

La reducida cantidad de datos empíricos que se pudieron recolectar no permiten generalizar las conclusiones, debido a los tiempos y tipo de investigación, por lo que sería necesario obtener más datos cuantitativos en futuras investigaciones.

[21] http://pythoncard.sourceforge.net/
[22] http://winpdb.org/ y https://code.google.com/p/winpdb/
[23] http://boa-constructor.sourceforge.net/

Otro desafío será analizar como se comportará la participación de las comunidades de software libre, si existirá interés en este tipo de herramientas, si habrá colaboraciones y cuestiones relacionadas.

Los aportes presentados, identificadores globales y detección facial, también abren ciertos interrogantes respecto a temas de privacidad y viabilidad técnica, que deberá ser estudiados más en profundidad.

7 Conclusión

Se implementó un marco de trabajo teórico-práctico para la creación y mantenimiento de software libre de manera ágil y de alta calidad, especialmente orientado al desarrollo de aplicaciones empresariales centradas en datos (para programadores individuales, en esta etapa). La solución está compuesta por el entorno integrado de desarrollo (incorporando nuevas características modernas) y el análisis bibliográfico de la propuesta, aportando posibles nuevas alternativas al estado del arte.

Se buscó aplicar las buenas prácticas de ingeniería del software, tendientes a una mejora continua a nivel personal, logrando avanzar en la integración de una herramienta de desarrollo rápido con interfaz enfocada en tareas. Surgieron varios aportes tentativos al estado del arte, respecto a la recolección automatizada de métricas (medición más precisa de tiempos vía detección facial y seguimiento de lineas de código por identificadores globales), para superar las dificultades conocidas de los procesos analizados. Por diversas cuestiones, no se pudo continuar sobre las temáticas de migración de código legado, de las cuales se realizó una exploración inicial.

La metodología de Investigación-Acción fue muy útil para comprobar e incorporar muchos conceptos teóricos que pueden ser aplicados a la docencia o incluso mismo a empresas de desarrollo de software, para mejorar las practicas actuales.

Dado el carácter experimental del prototipo, serán necesario ajustes técnicos, correcciones y mejoras a las herramientas para poder ser aplicable en otros contextos o a un número mayor de desarrolladores..

Por último, en futuras lineas de investigación se podría continuar mejorando el marco de trabajo y estudiando sus implicaciones. Ya con una base más sólida, se podría también profundizar en la migración de lenguajes legados, principalmente de Visual Basic, pudiendo ser aplicable incluso a otros lenguajes y áreas del mercado aún no

cubiertas por el software libre. Esto posibilitaría también ampliar la investigación y recolectar mayor cantidad de datos empíricos, en vista de una generalización más útil de las conclusiones, con resultados más completos y extensivos.

Agradecimientos

El presente es un resumen de la tesis de maestría correspondiente a mi Trabajo Final[24] del Máster en Software Libre (Universidad Abierta de Cataluña), y agradezco la directora del proyecto, Dra. Verónica Xhardez, y al co-director Dr. Ricardo Medel, por su acompañamiento a lo largo de las diferentes etapas del mismo, al consultor docente, Alexandre Viejo Galicia, y todos los que colaboraron en la revisión y evaluación de esta investigación.

Referencias

1. Rossi, M (2006). **Decoding the Free/OpenSource Software Puzzle: A Survey of Theoretical and Empirical Contributuions**. In J. Bitzer and P.J.H. Schröder (Eds.) **The economics of Open Source Software Development** (pp.15-55). Amsterdam. Elsevier. ISBN 9780444527691[25]
2. Feller J. & Fitzgerald B. (2002) **Understanding Open Source software development.** Addison-Wesley Professional. UK. ISBN 978-0201734966
3. Bessen J (2006). **Open Source Software: Free Provision of Complex Public Goods**. In J. Bitzer and P.J.H. Schröder (Eds.) **The economics of Open Source Software Development** (pp.15-55). Amsterdam. Elsevier. ISBN 9780444527691[26]
4. Choi H.J., Sang-Hun L., Syed A., Ahmad I., Hyun-Il S. and Young-Kyu P. (2013) **Towards an Integrated Personal Software Process and Team Software Process Supporting Tool.** Software Process Improvement and Management: Approaches and Tools for Practical Development. IGI Global, 2012. 205-223. Web. 5 Dec. 2013. doi: 10.4018/978-1-61350-141-2.ch010
5. Chaiyo Y., Ramingwong S. (2013) **The Development of a Design Tool for Personal Software Process (PSP)**. 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), IEEE. Krabi, 15-17 May 2013. ISBN: 978-1-4799-0546-1. pp. 1-4. doi: 10.1109/ECTICon.2013.6559562
6. El-Ramly M., Eltayeb R., & Alla H.A. (2006) **An Experiment in Automatic Conversion of Legacy Java Programs to C#**. IEEE International Conference on Computer Systems and Applications. March 8, 2006. (pp. 1037-1045). doi: 10.1109/AICCSA.2006.205215

[24] Reingart, M. Memoria del proyecto de investigación. Trabajo Final de Máster de Software Libre. Universidad Oberta de Cataluya http://hdl.handle.net/10609/40394 [PDF]

[25] http://www.econ-pol.unisi.it/quaderni/424.pdf

[26] http://www.researchoninnovation.org/opensrc.pdf

7. Asaduzzaman M., Roy C. K. , Schneider K. A. (2013) **LHDiff: A Language-Independent Hybrid Approach for Tracking Source Code Lines.** Software Maintenance (ICSM), 2013 29th IEEE International Conference on; vol., no., pp.230,239, 22-28 Sept. 2013 doi: 10.1109/ICSM.2013.34 [27]
8. Schmidt, B., Reinhardt, W. (2009) **Task Patterns to support task-centric Social Software Engineering.** 3rd International Workshop on Social Information Retrieval for Technology-Enhanced Learning &&**495. Aachen, Germany, August 21, 2009.[28]
9. Bacchelli A., Ponzanelli L., Lanza M. (2012) **Harnessing Stack Overflow for the IDE.** REVEAL @ Faculty of Informatics - University of Lugano, Switzerland. Third International Workshop on Recommendation Systems for Software Engineering (RSSE). Zurich, 4-4 June 2012. [29]
10. Zhang F., Khomh, F., Zou Y., Hassan A.E. (2012) **An Empirical Study of the Effect of File Editing Patterns on Software Quality**. 19th Working Conference on Reverse Engineering (WCRE), IEEE. Kingston, ON 15-18 Oct. 2012. ISSN:1095-1350. pp. 456 - 465. doi: 10.1109/WCRE.2012.55
11. Goth G. (2009). **The Task-Based Interface: Not Your Father's Desktop**. IEEE Software, Volume: 26 Issue 6), 88-91. doi:10.1109/MS.2009.191[30]
12. Murphy G.C., Kersten M., Findlater L. (2006) **How are Java software developers are using the Eclipse IDE?** IEEE Software article, (c) IEEE (2006) Volume: 23, Issue: 4 pp 76- 83. doi: 10.1109/MS.2006.105[31]

Nota: Los sitios web fueron consultados entre Febrero de 2014 y Junio de 2015.

Aclaración: los enlaces a los artículos de Wikipedia son meramente a título informativo e introductorio. Para mayor información revisar las referencias bibliográficas y la memoria completa del trabajo de investigación.

[27] http://www.cs.usask.ca/~croy/papers/2013/LHDiffFullPaper-preprint.pdf
[28] http://ceur-ws.org/Vol-535/Schmidt_Reinhardt_SIRTEL09.pdf
[29] http://sback.it/publications/rsse2012.pdf
[30] http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05287017_
[31] http://www.tasktop.com/pdfs/docs/publications/2006-ieee-eclipse-usage.pdf