Análisis de deuda técnica de UX en repositorios de GitHub

Ana Liz Lubomirsky, Juan Cruz Gardey, and Alejandra Garrido

LIFIA, Facultad de Informática, Universidad Nacional de La Plata, Argentina
ana.lubo99@gmail.com
{jcgardey,garrido}@lifia.info.unlp.edu.ar

Resumen El concepto de deuda técnica se utiliza para denotar problemas de calidad del software que a medida que transcurre el tiempo se hacen progresivamente más dificiles de reparar. Este trabajo consiste en investigar la presencia de deuda técnica relacionada a la experiencia de usuario (UX) en las incidencias (issues) de 8 proyectos de GitHub, con el fin de fundamentar la necesidad de administrar esta deuda de UX, tal como ocurre con otros tipos de deuda técnica. Mediante diferentes análisis, se caracterizaron los issues relacionados a la UX y se identificaron aquellos que pueden contribuir a la acumulación de deuda de UX, para comparar su índice de resolución frente con otros tipos de issues, y así determinar cuanta importancia se le da al tratamiento de la deuda de UX. Los resultados muestran que los issues que generan deuda de UX con frecuencia no se resuelven, o su atención se posterga para priorizar otro tipo de issues.

Keywords: deuda técnica; experiencia de usuario; refactoring

1. Introducción

La deuda técnica es una metáfora propuesta por Ward Cunningham que describe las consecuencias de las acciones de desarrollo de software que, de forma intencionada o no, priorizan el desarrollo de la funcionalidad requerida por el cliente por encima de consideraciones de diseño e implementación y sus parámetros de calidad [1]. Conceptualmente, la deuda técnica es un análogo de la deuda financiera; la acumulación de deuda genera un interés que hace que su costo de remediación sea cada vez mayor.

Originalmente la métafora de deuda técnica fue pensada para describir cuestiones de implementación a nivel de código fuente, pero luego surgieron otros tipos de deuda como por ejemplo de arquitectura, de requerimientos, de documentación, etc [8]. Dentro de estos tipos de deuda, aparece la deuda técnica de diseño de experiencia de usuario (UX debt), a la que Kuusinen define como "diseño no del todo correcto que posponemos hacerlo bien" [6]. Si bien existen trabajos que reconocen a la UX debt como un tipo de deuda técnica, ninguno de estos provee una caracterización que permita identificarla y mantenerla bajo control.

Volviendo a la metáfora original, la presencia de *code smells* es uno de los indicadores de deuda técnica [8]. Los code smells son un conjunto de malas prácticas de codificación que pueden ser resueltos a través de *refactorings*. Los refactorings son transformaciones de código que tienen por objetivo mejorar la calidad del mismo sin modificar la funcionalidad de la aplicación. Siguiendo esta misma filosofía, en trabajos previos se ha propuesto el concepto de *UX smells* como indicadores de una mala experiencia de usuario [5]. Estos UX smells no tienen que ver con aspectos funcionales del sistema, sino con cuestiones de la interacción del usuario que pueden mejorarse a través de uno o más UX refactorings. Así como los code smells generan deuda técnica, los UX smells contribuyen a la acumulación de UX debt.

Este trabajo consiste en investigar la presencia de UX debt (en términos de UX smells) en interfaces web de proyectos en GitHub, con el objetivo de recopilar evidencia acerca del nivel de atención que reciben los UX smells. Para esto se propone analizar los *issues* de un conjuntos de proyectos, que básicamente son incidencias reportadas por los mismos colaborados del proyecto o por usuarios externos. Un issue puede ser entre otras cosas un error a corregir, una petición para añadir una nueva funcionalidad o una pregunta para aclarar alguna cuestión puntual.

A partir del análisis de los issues, se propone realizar una clasificación automática de éstos en la que primero se identifican aquellos que reportan aspectos relacionados a la experiencia de usuario y luego dentro de este grupo, se identifican los que cumplen con la noción de UX smell. Es decir, que no reflejan cuestiones funcionales del sistema (como bugs o feature requests), sino aspectos de la interfaz de usuario (UI) que pueden modificarse para mejorar la interacción. Posteriormente, se analiza la resolución de los UX smells para determinar la existencia de UX debt en los diferentes proyectos.

El artículo está organizado de la siguiente manera. La sección 2 presenta el trabajo relacionado a la deuda técnica, UX debt, y al análisis de issues en sistemas de gestión de incidencias. La sección 3 explica la clasificación de issues realizada y el posterior análisis de la resolución de los mismos. Finalmente la sección 4 describe las conclusiones y trabajos futuros.

2. Trabajo Relacionado

No existen muchos trabajos que mencionen a la UX como un tipo específico de deuda técnica. Algunos de estos utilizan una definición de "usability debt" que en general concuerda con la idea de posponer el diseño de UX, pero no proveen una definición práctica que permita identificarla [4,6]. En este trabajo se utiliza el concepto de UX smell como elementos que contribuyen a la acumulación de UX debt, así como los code smells son un componente fundamental de la deuda técnica.

Estudios anteriores han utilizado sistemas de gestión de issues para analizar la deuda técnica [7] basándose en la premisa de que los desarrolladores comúnmente documentan esta deuda a través de comentarios del código fuente, como han

observado Potdar y Shihab, en su estudio a través del análisis de más de 100K comentarios de código [9].

Además, otros estudios proponen el uso de técnicas de procesamiento de lenguaje natural para apoyar la identificación de deuda técnica admitida, que es un caso particular de deuda técnica donde los desarrolladores reconocen explícitamente sus decisiones de implementación subóptimas [2]. Maldonado et al. [3] propusieron una técnica para identificar con precisión este tipo de deuda, basada en palabras clave y frases fijas.

En este trabajo se analizan los issues con el propósito de profundizar el estudio de la presencia de UX debt, que puede ser introducida de forma intencional o no. Se realiza una clasificación diferenciando los issues genéricos de los issues de UX, y de estos se separan los issues de UX que se pueden resolver con refactorings de UX.

3. Análisis de issues en GitHub

3.1. Selección de repositorios

El primer paso del proceso de análisis fue la búsqueda de repositorios públicos que sean útiles para la búsqueda a realizar. En particular, se seleccionaron proyectos de aplicaciones web que hagan un uso intensivo de una interfaz de usuario. Adicionalmente, se establecieron tres criterios más:

- Actividad. Que sean repositorios con actividad reciente, es decir, que no sean repositorios archivados u obsoletos.
- Cantidad de issues. Se buscaron repositorios con al menos 2000 issues para que el análisis sea representativo.
- Popularidad. Se buscaron repositorios que sean populares para asegurarse que tengan cierto movimiento de issues. Como medida de popularidad se utilizó el ranking de estrellas (stars) que es la manera en que los usuarios pueden valorar los repositorios.

Para la búsqueda de los proyectos se utilizó el buscador de GitHub. Con el término "web application" se listaron todos los proyectos relacionados con aplicaciones web, y luego de ordenarlos por popularidad en forma descendente, se analizó el nombre y la descripción de cada uno para determinar si cumplía con los criterios anteriores. La tabla 1 muestra los datos de los 8 proyectos seleccionados. Se decidió limitar la cantidad de proyectos elegidos a 8 porque el análisis de cada uno requiere un tiempo considerable.

3.2. Recuperación de issues

Luego de seleccionar los repositorios a analizar, se descargaron los issues de cada uno utilizando la API de GitHub y se almacenaron en un archivo csv (valores separados por comas) para su posterior procesamiento. De todos los datos que proporciona la API, se seleccionaron los que se listan a continuación:

Tabla 1. Datos de los repositorios elegidos

Proyecto	Nombre en GitHub	#Issues	Cerrados
Jitsi Meet	jitsi/jitsi-meet	4705	66 %
Matomo	matomo-org/matomo	11823	77 %
ERPNext	frappe/erpnext	12282	75 %
KeystoneJS	keystonejs/keystone-classic	2458	88 %
Visual Studio Code	microsoft/vscode	81381	95 %
Wallabag	wallabag/wallabag	3027	82 %
Open MCT	nasa/openmct	2736	73 %
Superset	apache/superset	8059	88 %

- **Título (title):** el título del issue.
- **Descripción (body):** el contenido del issue.
- **Etiquetas (labels):** etiquetas usadas para clasificar y categorizar el issue, esto ayuda a tener una idea rápida del tipo de issue, y también poder filtrar por dichas clasificaciones.
- **Estado (state):** estado del issue, puede ser abierto o cerrado (open/closed).
- **Fecha de creación (created_at):** fecha en que se creó el issue.
- **Fecha de cierre (closed_at):** fecha en que se cerró el issue (si es que está cerrado).
- **Cantidad de comentarios (comments):** comentarios realizados por los colaboradores y usuarios del proyecto.

Además, se agregaron los siguientes atributos calculados a partir de los datos de cada issue:

- **Solucionado (verdadero/falso):** en principio se consideró resuelto a todo issue cuyo estado era cerrado. Luego analizando los issues, se identificó que en muchos casos estos se cierran asignándoles la etiqueta "wontfix", la cual se utiliza para indicar que el issue no será resuelto. Por ese motivo, se agregó este atributo cuyo valor es verdadero únicamente cuando el estado del issue es cerrado y no posee la etiqueta wontfix.
- **Tiempo de resolución:** es la diferencia en días entre la fecha de cierre del issue y la fecha de apertura.

3.3. Clasificación de issues de UX

Teniendo en cuenta que cada proyecto contiene diferentes tipos de issues, para identificar los UX smells primero fue necesario categorizar aquellos issues relacionados con la experiencia de usuario. Esta clasificación se realizó de forma automática mediante la búsqueda de ciertas palabras clave (ver apéndice[1]). Este listado de palabras clave se confeccionó inspeccionando manualmente un

[1] https://bit.ly/3SkvQqA

conjunto aleatorio de issues de cada proyecto analizado que los mismos usuarios etiquetaron como "UX".

Para cada issue, se agregó un atributo **UX** (verdadero/falso) cuyo valor es verdadero si al menos 3 de las palabras clave están contenidas en el issue. Para esta búsqueda se realizó el proceso de tokenización del issue, el cual consiste en dividir el texto en las palabras que lo conforman, considerando el título, contenido y los labels.

Es importante mencionar que este proceso de búsqueda fue iterativo. En cada iteración, se realizó la clasificación con las palabras seleccionadas hasta el momento y luego se inspeccionó un conjunto aleatorio de 20 issues de cada proyecto para refinar el listado de palabras. Esto se repitió en todos los proyectos hasta alcanzar al menos una precisión del 80 % en el conjunto de issues inspeccionado. En cuanto a la cantidad de palabras clave que un issue debía contener, se realizaron pruebas con diferentes umbrales (1, 2 y 3 valores) y la mejor precisión se obtuvo con 3.

3.4. Clasificación de UX smells

Luego de clasificar los issues de UX, el siguiente paso fue identificar dentro de este grupo, aquellos que pueden considerarse UX smells. Un UX smell se puede definir como un indicio de una experiencia de usuario deficiente [5]. Un ejemplo de UX smell es *Unformatted Input*, que ocurre cuando se utiliza un campo de texto libre en un formulario para ingresar un dato con cierto formato (como por ejemplo una fecha), lo cual es propenso a errores. Si bien ya se definió un catálogo de UX smells en [5], este no es definitivo y podría ser extendido a medida que se encuentran casos de una mala experiencia de usuario. Es por eso que el objetivo en este punto es identificar issues que cumplan con los criterios de un UX smell: que no sea un bug o un error de codificación, y que no implique agregar o quitar funcionalidad de la aplicación. A continuación se presenta un issue que se considera UX smell:

"Link preview text on hover are inconsistent. *Our link preview text on hover are inconsistent across various views, we should work to make all of them consistent."* (microsoft/vscode).

En un primer análisis de los issues se pensó que habría tres separaciones estrictamente marcadas dentro de los issues de UX: bugs, feature-requests y smells. Por este motivo, en una primera instancia únicamente se utilizaron los labels que los mismos usuarios asignan para descartar bugs ("bug") y feature-request ("feature" y "feature-request"), pero luego se decidió agregar también las palabras de la descripción, porque no siempre los issues estaban bien etiquetados.

Al igual que la clasificación anterior este también fue un proceso iterativo de dos pasos: primero se realizaba la búsqueda y después se analizaban un conjunto de 20 issues en cada proyecto para refinar el listado de palabras clave. Estos pasos se repitieron hasta alcanzar una precisión del 80 % en el conjunto de issues inspeccionado.

En las primeras iteraciones, al leer detenidamente los issues uno por uno, se observó que había muchos UX smells que estaban etiquetados por los usuarios como "feature-requests", lo que hizo tomar la decisión de no descartar desde el principio estos issues y replantear la manera de detectar cada tipo de issue. Así, se decidió descartar solamente los bugs y se trabajó en refinar las listas de palabras para identificar los smells.

Analizando los issues se observó que era muy difícil tener un sólo listado de palabras generalizado para clasificar los smells de cualquier proyecto y alcanzar la precisión buscada. Dado que cada proyecto tiene sus particularidades y palabras específicas en los issues, para los primeros tres proyectos analizados (Jitsi Meet, Matomo y ERPNext), se realizó un refinamiento exhaustivo de las palabras clave (ver apéndice[2]). Para los siguientes repositorios a tratar, se utilizó un único listado de palabras general dado que no se identificaron grandes diferencias.

3.5. Resultados

Con el objetivo de analizar cuánta atención se presta a los UX issues se calcularon las siguientes métricas:

- **Proporción de issues resueltos**: se consideran resueltos a los issues cerrados que no poseen la etiqueta "wontfix". En particular, se calculó la proporción de smells resueltos (Smells-PR) y del resto de los UX issues que no son smells (UX-PR).
- **Tiempo de resolución promedio**: es la cantidad de días entre la fecha de apertura y la fecha de cierre del issue. Se calculó tanto para los smells (Smells-TR) como para el resto de los UX issues (UX-TR).
- **Cantidad de comentarios promedio**: también se calculó para los smells (Smells-C) y para el resto de los UX issues (UX-C), con el fin de tener una medida del grado de discusión que tienen los issues y analizar si esto tiene alguna relación con su resolución.

Los resultados aparecen en la tabla 2. Se puede observar que:

- La proporción de UX issues resueltos es siempre menor que la de issues en general (ver columna "Cerrados" en la tabla 1). En algunos casos la diferencia es pequeña pero en otros, como Jitsi Meet, es casi la mitad.
- Con respecto a la proporción de smells resueltos, ERPNext es el único proyecto en el que esta es notablemente menor que la del resto de los UX issues. En los proyectos restantes (a excepción de KeystoneJS), la diferencia es mínima. Sin embargo, en todos estos casos se observa una diferencia considerable en el tiempo de resolución de los UX issues versus UX smells. El hecho que los UX smells tarden más tiempo en resolverse que los demás issues de UX, indica que la resolución de UX smells suele postergarse en favor de otros issues.

[2] https://bit.ly/3SkvQqA

Tabla 2. Resultados de la búsqueda automática. PR: proporción de issues resueltos. TR: tiempo de resolución (en días).

Repositorio	UX-PR	Smells-PR	UX-TR	Smells-TR	UX-C	Smells-C
Jitsi Meet	35 %	34 %	112,40	228,57	5,43	3,48
Matomo	64 %	58 %	166,55	348,90	6,96	5,62
ERPNext	65 %	38 %	221,82	335,40	2,05	1,66
KeystoneJS	84 %	91 %	165,21	179,60	4,22	5,24
VS Code	92 %	90 %	12,97	15,40	4,15	4,52
Wallabag	78 %	72 %	147,68	220,50	3,61	3,48
Open MCT	69 %	64 %	139,44	168,11	3,01	3,27
Superset	86 %	85 %	145,93	210,96	3,96	4,19

- En el caso específico de VS Code, los resultados muestran que los smells reciben la misma atención que el resto de los UX issues porque tanto la proporción de resolución como el tiempo de resolución muestran valores muy similares, y además son significativamente mayor y menor que los demás proyectos.
- Con respecto a la cantidad de comentarios promedio no se encontró una relación entre esta cantidad y la resolución de los issues.

3.6. Búsqueda manual

Los resultados anteriores fueron obtenidos a partir de la clasificación automática de UX issues basada en la búsqueda de palabras clave. Observando la tabla 2, resulta llamativo que haya un gran número de UX smells resueltos; en el caso de KeystoneJS y VS Code poseen una proporción del 90 %. Analizando el conjunto de UX smells resultantes de la clasificación, se encontraron muchos falsos positivos, es decir, issues que fueron clasificados como UX smells cuando en realidad no lo eran. Esto se debe a que la búsqueda a través de la palabras clave tienen sus limitaciones, ya que una misma palabra puede tener diferentes significados en distintos contextos. Además, también puede ocurrir que los usuarios utilicen alguna de las palabras usadas para identificar el smell para reportar otro tipo de issues.

Por otro lado, la única forma de determinar automáticamente si un issue está resuelto o no es a partir de su estado y sus etiquetas. Sin embargo, no siempre un issue cerrado está efectivamente resuelto. Por ejemplo, en muchos proyectos ocurre que los issues se cierran automáticamente luego de un cierto período de inactividad, o los mismos desarrolladores deciden no resolver el issue, pero no lo etiquetan como "wontfix".

Por los motivos detallados anteriormente, se realizó un análisis manual de los issues clasificados como UX smells para comprobar que estuvieran correctamente clasificados y que efectivamente estuvieran resueltos aquellos que estaban cerrados. En este análisis se incluyeron los proyectos: Jitsi Meet, Matomo, ERPNext, KeystoneJS y Wallabag.

3.6.1. Análisis de los Pull Request

Para determinar el impacto de la resolución de los UX smells en el código fuente, se recurrió al análisis de los **pull requests** (PRs) asociados a los mismos. Un pull request es una petición que hace un usuario para incorporar modificaciones al código fuente, que consisten en una colección de **commits**; cada uno de estos es una captura del estado del proyecto en un momento concreto.

Como la API de GitHub no proporciona detalles acerca de los PRs, se inspeccionaron en forma individual los smells clasificados en la web de GitHub, con los siguientes objetivos:

- Obtener el tiempo que transcurre desde la creación del issue hasta que se menciona en un PR como una medida de cuánto se pospone la resolución de un issue reportado.
- Calcular el tiempo de resolución del issue contabilizando la cantidad de días desde que se creó hasta que se cerró el PR. Esto se hizo porque se pensó que este tiempo sería más preciso que contar los días que el issue estuvo abierto.

Sin embargo, analizando los issues se observó que la proporción de smells resueltos con PRs asociados era muy baja como para calcular las métricas descriptas anteriormente. Por este motivo es que se decidió descartar el análisis de los PR, y se desarrolló una clasificación manual de los UX smells para analizar su resolución.

3.6.2. Resolución de UX smells

Inspeccionando los issues individualmente y sus discusiones generadas entre los usuarios, luego de eliminar los falsos positivos que arrojó el análisis automático, se llevó a cabo la siguiente clasificación:

- **Descartado (D)**. Un UX smell es descartado cuando los mismos usuarios luego de una discusión, determinan que la incidencia planteada no constituye un problema de experiencia de usuario. En general estos issues se cierran sin tener ningún impacto en el proyecto.
- **Ignorado (I)**. El issue no presenta ningún tipo de actividad: no tiene comentarios, ni tiene PRs asociados.
- **Resuelto (R)**. Esto ocurre cuando tiene algún PR asociado, o también cuando en la discusión generada alguno de los usuarios indica que fue resuelto en una nueva versión de la aplicación.
- **No resuelto (NR)**. A diferencia del ignorado tiene actividad, pero no hay evidencia de que haya sido resuelto. En ciertos casos también ocurre que los colaboradores deciden que resolver el smell no entra en las prioridades del proyecto y lo dejan de lado.

A cada issue se le asignó solo una de las categorías anteriores. Los resultados de la clasificación se muestran en la tabla 3. Se puede observar que:

- En casi todos los casos la cantidad de smells obtenidos en esta búsqueda manual (Smell-BM) es considerablemente más baja que la cantidad de smells

Tabla 3. Resultados de la clasificación manual de smells. Smells-BA: cantidad de smells detectados en la búsqueda automática. Smells-BM: cantidad de smells sin los falsos positivos.

Proyecto	Smells-BA	Smells-BM	D	NR	I	R	P-R
Jitsi Meet	37	29	8	1	13	7	33 %
Matomo	183	115	6	17	21	71	65 %
ERPNext	87	56	4	17	19	16	31 %
KeystoneJS	74	15	1	4	0	10	71 %
Wallabag	94	32	3	5	7	17	59 %

detectados automáticamente. Esto muestra que la búsqueda automática contiene un gran número de falsos positivos.

- La proporción de smells resueltos (P-R), que no considera los issues descartados, es significativamente más baja en los casos de ERPNext, KeystoneJS y Wallabag en comparación a los resultados de la tabla 2. Esto se debe a que muchos de los issues en estos proyectos están cerrados, por lo cual en un principio se consideraron resueltos, pero luego en el análisis de sus comentarios no se encontró evidencia de que esto haya sido así.
- A excepción de Matomo y KeystoneJS, el resto de los proyectos tiene por lo menos el 40 % de los smells sin resolver, lo que evidencia que los UX smells son dejados de lado. En el caso puntual de Jitsi Meet, la gran mayoría de los issues no resueltos fueron ignorados, lo cual es una muestra que el equipo de desarrollo desestimó los smells sin analizar si convenía solucionarlos o no. Los otros dos proyectos, ERPNext y Wallabag, contienen smells ignorados en una proporción similar a la de los no resueltos: issues que fueron discutidos pero finalmente no se resolvieron.

4. Conclusiones y trabajo futuro

Este trabajo presenta evidencia de la presencia de deuda técnica de UX en un conjunto de proyectos de GitHub. Esta deuda se caracteriza a partir de la identificación de UX smells: problemas con la experiencia del usuario de una aplicación que pueden resolverse sin alterar la funcionalidad de la misma (a través de UX refactorings).

Primero se desarrolló una clasificación automática de issues para identificar aquellos que cumplen con el concepto de UX smell. Esta clasificación se realizó a través de la búsqueda de palabras clave siguiendo un proceso iterativo. Los resultados muestran que en muchos casos los UX smells o no se resuelven (la mitad de los proyectos tienen al menos 35 % de issues sin solucionar), o son postergados en favor de otros issues (esto se refleja en el tiempo de resolución).

En los resultados de la clasificación anterior, se observó que muchos de los issues clasificados como UX smells en realidad no lo eran, y que a su vez en diferentes ocasiones estos se cerraban sin ser resueltos. Por este motivo, se decidió

realizar un análisis sistemático de los UX smells para eliminar los falsos positivos y determinar cuáles de ellos habían sido efectivamente resueltos. Como resultado, la mayoría de los proyectos analizados muestran una proporción menor de UX smells resueltos que la clasificación automática, lo cual reafirma la idea que los smells se postergan y esto es lo que contribuye a la acumulación de UX debt.

Como trabajo futuro, se propone trabajar en la mejora de la clasificación automática de UX smells, utilizando técnicas de minería de texto. Por otro lado, también queda pendiente estudiar otras métricas que permitan estimar con mayor precisión el costo de resolución de los issues, así poder tener una medida de cuánto trabajo implica remediar la deuda acumulada.

Referencias

1. Behutiye, W.N., Rodríguez, P., Oivo, M., Tosun, A.: Analyzing the concept of technical debt in the context of agile software development: A systematic literature review. Information and Software Technology 82 (2017)
2. Da Silva Maldonado, S.: Detecting and quantifying different types of self-admitted technical debt. In 7th International Workshop on Managing Technical Debt (MTD). 9–15. (2015)
3. Da Silva Maldonado, Shihab, T.: Using natural language processing to automatically detect self-admitted technical debt. IEEE Transactions on Software Engineering 43, 11 (2017), 1044–1062 (2017)
4. da Fonseca Lage, L., Kalinowski, M., Trevisan, D., Spinola, R.: Usability technical debt in software projects: A multi-case study. In: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE (2019)
5. Grigera, J., Garrido, A., Rivero, J., Rossi, G.: Automatic detection of usability smells in web applications. International Journal of Human-Computer Studies 97 (2017)
6. Kuusinen, K.: Bob: a framework for organizing within-iteration ux work in agile development. In: Integrating User-Centred Design in Agile Development. Springer (2016)
7. Lage, Kalinowski, T.S.: Usability technical debt in software projects: A multi-case study. ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2019, pp. 1-6, doi: 10.1109/ESEM.2019.8870180. (2019)
8. Li, Z., Avgeriou, P., Liang, P.: A systematic mapping study on technical debt and its management. Journal of Systems and Software 101, 193–220 (2015)
9. Potdar, S.: An exploratory study on self-admitted technical debt. In 30th International Conference on Software Maintenance and Evolution (ICSME). 91–100. (2014)