Un primer enfoque para medir la calidad de FIWARE

Ignacio Villalobos, Javier Ferrer y Enrique Alba

Universidad de Málaga, Málaga, Spain
{nacho, ferrer, eat}@lcc.uma.es

Resumen FIWARE es un ecosistema tecnológico abierto que pretende convertirse en la plataforma de referencia para los servicios y aplicaciones del Internet del Futuro. Para ello, primero se necesita solventar las dudas existentes en cuanto a la calidad de FIWARE, ya que la plataforma manejará datos sensibles tanto personales como esenciales para la correcta gestión de las ciudades inteligentes. Hay muchas formas de estudiar la calidad de un *middleware* complejo como éste. En nuestro caso seguimos las pautas de un estándar ISO usando herramientas existentes en una primera fase de identificación de problemas. Tras estudiar 26 *habilitadores genéricos* de referencia de FIWARE, hemos detectado numerosos puntos de mejora. En el caso de la confiabilidad y seguridad se podrían solventar en poco tiempo, mientras que los defectos relativos a mantenibilidad requerirán del orden de meses de trabajo. Esto posiblemente es debido al carácter tan heterogéneo del equipo de desarrollado de FIWARE (miembros de diversas empresas), que afecta directamente a la mantenibilidad del código.

Palabras clave: FIWARE, Calidad, Métricas, Ciudades Inteligentes

1. Introducción

La economía de la Unión Europea (UE) es la más grande del mundo según el Fondo Monetario Internacional. Para seguir siendo competitiva, la UE ha apostado estos últimos años por el desarrollo e innovación tecnológica para el Internet del Futuro [1]. Uno de sus grandes objetivos es crear un ecosistema tecnológico gratuito para facilitar la aparición de servicios que satisfagan la demanda del mercado en múltiples sectores como la sanidad, las factorías, las telecomunicaciones, la agricultura o los servicios medioambientales: ciudades inteligentes, agricultura inteligente, industria 4.0. A mediados de 2011, con el proyecto FIWARE [2], se comenzaron a dar los primeros pasos para cristalizar este gran objetivo.

FIWARE es una plataforma abierta impulsada no solo por la UE sino también por otros socios privados, que busca el desarrollo y despliegue de aplicaciones del Internet del Futuro. Los promotores apoyan activamente el desarrollo de estándares y de aplicaciones finales o *verticales*. La adopción de las nuevas tecnologías en la vida cotidiana de los ciudadanos es ya un hecho y se pretende que,

gracias a la naturaleza abierta de FIWARE, surjan numerosos suministradores de servicios o aplicaciones innovadoras abiertas para todos los ciudadanos.

En contraposición, los ciudadanos son cada vez más conscientes del carácter invasivo que tienen ciertas aplicaciones. Esto puede generar cierta preocupación cuando las aplicaciones manejan datos sensibles como ocurre en el sector de la sanidad u otros sectores estratégicos para la ciudad como son el tráfico, la energía o el medioambiente. Asegurar la calidad de la plataforma FIWARE, que incluye aspectos como la seguridad y la confiabilidad, es esencial para ganar la confianza tanto de desarrolladores de nuevos servicios y aplicaciones, como para usuarios.

La calidad del proceso y del producto software suele medirse desde diferentes ángulos según el estándar utilizado. Un estándar para la calidad del proceso es el CMM (*Capability Maturity Model*) [3], que dio lugar en 2006 a CMMI (*Capability Maturity Model Integration*) [4], propuestos en la Universidad Carnegie Mellon. En este estándar se definen procedimientos destinados a evaluar y mejorar los procesos de desarrollo e implementación del software. También existen otros estándares que provienen del ámbito industrial como el IEEE 730-2014 [5] que es una guía de procesos para el aseguramiento de la calidad. Aunque existen multitud de estándares y modelos de calidad diferentes, los más destacados son los estándares de la Organización Internacional de Normalización (ISO) adoptadas por 163 países [6]. En concreto, para el caso del aseguramiento de la calidad de un producto software existe el ISO 25010 [7], que debido a su amplia repercusión ha tenido una gran influencia en el sector del software. En este estándar se definen diferentes formas de medir la calidad; concretamente tres: modelo de calidad interna, externa y de uso. Estos modelos de calidad del software se basan en la etapa de desarrollo del software para el estudio de la calidad.

Debido a la falta de estudios científicos y a la poca información oficial por parte de los desarrolladores de FIWARE, así como de su Fundación, es realmente necesario un estudio del estado actual de la calidad de los principales componentes de esta plataforma. Estos componentes llamados *Generic Enablers* (GE) o habilitadores genéricos, son elementos funcionales de FIWARE, formados por un conjunto de componentes desarrollados en diversos lenguajes de programación (Java, Javascript, C++, Python, PHP,..) que proveen especificaciones de funcionalidades y APIs. Nuestro objetivo final es ofrecer un estudio cuantitativo de calidad del producto en los ocho ejes del estándar ISO 25010 y dar pautas para su mejora. En este primer estudio vamos a enfocarnos en aspectos clave del modelo de calidad interna como son la confiabilidad, la seguridad y la mantenibilidad.

Las contribuciones principales de este trabajo son las siguientes:

- Análisis de la calidad interna de los 26 habilitadores genéricos de FIWARE.
- Evaluación de tres atributos clave para la calidad de FIWARE: confiabilidad, seguridad y mantenibilidad.
- Estimación del esfuerzo necesario para mejorar los 26 GEs de FIWARE.

El resto del artículo tiene la siguiente estructura. En la Sección 2 hablaremos de Fiware: qué es, cuál es su arquitectura, sus elementos más importantes y finalmente sus objetivos y metas a largo plazo. Después, en la Sección 3 tendremos un acercamiento a la calidad desde el punto de vista del software y profundizaremos

en el estándar ISO 25010, que es nuestra inspiración metodológica a la hora de realizar este estudio. A continuación, la Sección 4 detalla el diseño de los experimentos, donde explicamos qué elementos han sido analizados y como se han estudiado dichos elementos. En la Sección 5, mostramos los resultados del análisis de las características que consideramos más importantes. En la última sección podremos encontrar las conclusiones extraídas de los experimentos, además de esbozar las líneas de trabajo futuro que consideramos más prometedoras.

2. FIWARE: un complejo caso de estudio

Recapitulando brevemente, FIWARE es un proyecto público-privado que pretende crear un ecosistema software abierto con el objetivo de convertirse es una plataforma de referencia para el Internet del Futuro. Este ha sido promovido principalmente por la UE y Telefónica junto a otros socios del sector TIC. Entre estos socios destacan las siguientes empresas: SAP, IBM, Telecom Italia, France Telecom, Nokia, Deutsche Telekom, Ericsson, Atos Spain, Alcatel-Lucent, Siemens, Intel, etc. Este proyecto fue inicialmente financiado por el Programa Marco 7, participando en él 90 instituciones y empresas de diferentes países. Recientemente se terminó de establecer su filosofía, arquitectura y API en 2016.

El ecosistema de FIWARE está compuesto por un conjunto de elementos llamados *habilitadores genéricos* (Generic enablers o GEs) que son los principales componentes de la plataforma. Todas las implementaciones de los GEs son de código abierto, por tanto, de libre acceso y públicas. El conjunto de GEs se encuentra ordenado según las capacidades que estos proporcionan. Estas diferentes clasificaciones y los GEs que conforman cada una de ellas se conocen como el *catálogo de FIWARE*. En dicho catálogo los GEs están organizados en siete capítulos tecnológicos diferentes según su funcionalidad: *Data/Context Management*, *Internet of Thing Services Enablement*, *Advanced Web-based User Interface*, *Security*, *Interface to Networks and Devices*, *Architecture of Applications / Services Ecosystem and Delivery Framework* y *Cloud Hosting*.

En FIWARE existen conceptos cercanos a los GEs que pueden llevar a confusión, por tanto queremos aclararlos para posteriormente definir el ámbito de nuestra experimentación en este trabajo. A continuación definimos los conceptos de GE, GEi, GEri, GEi obsoleto y GEi incubado:

- Generic Enabler (GE): es una definición de componente software basada en una especificación. Por ejemplo, el componente llamado Context Broker es uno de los GEs de FIWARE.
- Generic Enabler implementation (GEi): es una implementación particular de un GE. Hay que tener en cuenta que un GE puede tener varias implementaciones dado que las especificaciones de los GE son abiertas. Por ejemplo, Orion Context Broker es un GEi de FIWARE.
- Generic Enabler reference implementation (GEri): es un GEi en concreto que se ha elegido implementación de referencia. Por ejemplo, Orion Context Broker es el GEi elegido como GEri para el GE Context Broker.

- GEi obsoleto: es un GEi que no va a ser mejorado ni revisado y presenta diversos problemas en su desarrollo. No se recomienda su uso.
- GEi incubado: es un GEi que está siendo evaluado para comprobar que cumple con las especificaciones del GE que implementa.

En nuestro estudio nos centraremos únicamente en los GEris pues queremos conocer el estado real de los GEs que están siendo utilizados realmente en las aplicaciones que usan FIWARE. Actualmente, el uso de FIWARE se está extendiendo a muchas ciudades españolas y europeas debido a la realización de diversos proyectos en ciudades inteligentes. Estas aplicaciones abarcan los seis ejes definidos para el avance de las ciudades inteligentes: Economía, Movilidad, Gobernanza, Personas, Calidad de Vida y Medioambiente. La importancia de las aplicaciones para la ciudad, junto con la cantidad de datos que manejan diariamente, hace que FIWARE se haya convertido en una pieza clave y que sea necesario evaluar la calidad de las implementaciones que se usan para construir estas aplicaciones, ya que estarán desplegadas por toda Europa.

3. Calidad ISO 25010: nuestra inspiración metodológica

Informalmente un usuario quiere un software o servicio que es "bueno". Este término puede resultar ambiguo y no es cuantificable, en especial cuando hablamos de software. En las primeras definiciones desde un punto de vista más profesional se describe la calidad de un software como su aptitud para ser usado por un cliente (1970) o la conformidad de todos los requisitos de un producto (1979). Unos años más tarde, en la ISO 8402 (1986) se introduce una definición de la calidad de un producto y servicio, para posteriormente introducir con la ISO 9126 (1991) el término *producto software* a dicha definición. Pero en la actualidad queremos tanto el desarrollo como el producto software sea de calidad. Citando a Tom DeMarco (y a los Griegos antiguos): "*You can not control what you can not measure. Measurement is the prerequisite to management control*".

El estándar de referencia y más utilizado para evaluar la calidad de un producto software en la actualidad es el estándar ISO 25010. Este estándar define una taxonomía de las características principales que considerar a la hora de medir la calidad de un producto software. En la ISO 25010 las características de calidad se dividen en los siguientes ejes: Confiabilidad, Seguridad, Mantenibilidad, Funcionalidad, Eficiencia, Usabilidad, Compatibilidad y Portabilidad.

Cuando se aborda un proyecto software, debemos tener en cuenta cuales son nuestras prioridades de entre los ejes anteriormente mencionados, de acuerdo con los requisitos de nuestro proyecto. Incluso se aconseja actualizar estos requisitos de calidad a lo largo de la vida del proyecto [8]. Esto es debido a que algunas exigencias o restricciones en ciertos ejes pueden influenciar negativamente otros. Por ejemplo, puede darse el caso que al intentar buscar la máxima eficiencia tengamos que sacrificar algunos aspectos referentes a la mantenibilidad o viceversa.

En el estándar ISO 25010 se proponen diferentes formas de medir la calidad, concretamente se detallan tres modelos de calidad del software en función de la etapa de desarrollo del mismo:

- El modelo de calidad *interna* analiza el software desde su interior. Se miden aspectos de calidad que estén presentes en sus componentes analizando el código fuente. Este análisis se puede considerar de caja blanca.
- El modelo de calidad *externa* estudia la calidad del software desde el punto de vista del comportamiento que tiene éste durante su ejecución. Se puede considerar este análisis como de caja negra (o gris).
- El modelo de calidad de *uso* sólo es entendido en el entorno de operación y nos permite conocer si el usuario final es capaz de llevar a cabo el desempeño para el que fue diseñado. Puede considerarse de caja negra.

En este estudio nos centraremos en el modelo de calidad interna puesto que tenemos acceso al código fuente de los GEris de FIWARE.

4. Diseño experimental

En este trabajo analizamos la calidad interna de cada uno de los GEris seleccionados, debido a la importancia de tener un buen diseño software desde su etapa más temprana de desarrollo y así prevenir problemas posteriores. Nuestro estudio es cuantitativo, lejos de suposiciones subjetivas no demostrables que son frecuentes encontrar en el dominio del software. Para la realización de los experimentos se han seleccionado un total de 26 GEris del catálogo de FIWARE[1] (todos aquellos cuyo código fuente está accesible). De los GEi y GEris del catálogo sólo se quedan fuera de estudio los calificados como incubados (6), que están a la espera de ser evaluados para formar parte de FIWARE y los obsoletos (12), que ya no reciben soporte alguno.

Para la realización de los análisis se ha utilizado la herramienta SonarQube en su versión 6.7 y varias extensiones para analizar los diferentes GEs, pues estos se encuentran desarrollados en diferentes lenguajes de programación. Se han utilizado las extensiones estándar de SonarQube para todos los lenguajes (SonarJava, SonarJS, SonarPHP, SonarPython) a excepción de C++, donde se ha utilizado la extensión CppCheck para SonarQube. En la Tabla 1 podemos observar cada uno de los GEris ordenados alfabéticamente junto a la versión analizada de cada uno de ellos y los lenguajes en los que están desarrollados. También podemos observar el número de ficheros que componen cada uno de ellos y sus líneas de código totales (LOC), todo ello mostrando el alcance y dificultad de este trabajo.

5. Análisis de los resultados

En esta sección se analizan los resultados experimentales tras realizar los análisis a los GEris seleccionados (Tabla 1). Los experimentos se han centrado en tres de los ejes de la ISO 25010. El primero de estos ejes será la *confiabilidad*, debido a la importancia que tiene que un sistema software realice su función

[1] https://catalogue.fiware.org/

Tabla 1. Resumen GEris y las versiones analizadas.

ID	GEris	Lenguaje(s) principal(es)	Versión	#Ficheros	LOC
1	AuthzForce Server	Java	8.0.1	42	7850
2	Business API Ecosystem	Javascript y Python	6.4	11	909
3	Cepheus	Java	1.0.0	86	8584
4	Cloto	Python	2.8	31	2394
5	Cloud Portal	Javascript	5.4	184	31916
6	Cloud Rendering	Javascript	3.3.3	31	23058
7	Context Ckan	Javascript y Python	2.7.2	887	190667
8	Cosmos	Javascript	1.0.0	28	1834
9	GeoServer	Javascript y Python	5.4.3	5497	525815
10	Interface Designer	Javascript	5.4.3	13	55093
11	IOT Broker	Java	6.4	255	24597
12	IOT Discovery	Java	0.5.1	93	4745
13	Keyrock	Python	6.0	6	323
14	Kiara	Java	0.4	476	42873
15	Murano	Python	5.0.0-rc2	387	35081
16	OFNIC	Java	2.1	70	4290
17	Orion	C++	1.11.0	707	49655
18	Pegasus	Java y Python	2.4	504	38217
19	PEP Proxy Wilma	Javascript	6.2	7	621
20	POI Data Provider	PHP	3.3.3	12	1707
21	Proton	Java	2.0	3832	322407
22	Sagitta	Java y Python	3.2.0	342	22321
23	Sth Comet	Javascript	2.3.0	23	5192
24	Swift	Python	1.2.0	127	29813
25	Virtual Character	Javascript	3.3.3	64	32067
26	WireCloud	Javascript y Python	1.1.1	532	78564

sin presentar ningún fallo. En segundo lugar, analizamos el eje de *seguridad*, debido a la vital importancia de prevenir los ataques a las aplicaciones IoT: seguridad por diseño [9], ya que la seguridad activa formaría parte del modelo de calidad externo. Por último, consideramos también el eje de *mantenibilidad*, pues es completamente necesario que un sistema esté preparado para que cualquier modificación se pueda realizar en un tiempo razonable. A continuación, presentamos los tres análisis en sendas subsecciones.

5.1. Eje de Confiabilidad

En esta subsección se evalúa cada GEri desde el punto de vista de la confiabilidad. Para ello, nos centraremos en buscar la cantidad de *bugs* y los clasificaremos de acuerdo a su gravedad. Entendemos *bug* como un defecto en el código que, aunque aún no haya fallado, eventualmente sucederá dicho *bug*. Algunos ejemplos pueden ser: bucles sin condición de salida, descriptores de ficheros sin cerrar, valor indeterminado de variables o excepciones lanzadas, pero no capturadas. Finalmente daremos una calificación de la confiabilidad en cada GEri, o sea un ranking. Según dicha calificación, cuanta peor valoración se obtenga,es más probable que se produzca un error en algún componente de dicho GEri.

En la Tabla 2 mostramos el identificador de cada GEri seguido de su nombre. En la columna siguiente podemos ver la calificación otorgada a cada uno de los GEris. Ésta se comprende entre los valores A-E y se asigna de la siguiente manera: se recibe calificación A si no se detecta ningún *bug*, B si al menos se encuentra un *bug* leve, C si al menos tiene un bug grave, D si al menos se encuentra un *bug* crítico y finalmente E, si contiene al menos un *bug* bloqueante.

También mostramos el esfuerzo [2] en minutos estimado para arreglar todos los *bugs* detectados y finalmente reportamos cuatro columnas donde se desglosan cada uno de los *bugs* según su gravedad. Nótese que los resultados se ordenan según la calificación obtenida (de mejor a peor) y en caso de igualdad, el criterio es el esfuerzo requerido para solventarlos.

Tabla 2. Resultados en el eje de confiabilidad para cada GEri

ID	GEris	Rating	Esfuerzo(min)	Bugs			
				Bloqueante	Críticos	Graves	Leves
2	Business API Ecosystem	A	0	0	0	0	0
4	Cloto	A	0	0	0	0	0
13	Keyrock	A	0	0	0	0	0
20	POI Data Provider	B	200	0	0	0	40
19	PEP Proxy Wilma	C	15	0	0	1	0
24	Swift	C	57	0	0	12	0
23	Sth Comet	C	75	0	0	6	0
1	AuthzForce Server	C	105	0	0	5	4
17	Orion	C	155	0	0	3	28
12	IOT Discovery	C	380	0	0	25	0
6	Cloud Rendering	C	660	0	0	54	0
25	Virtual Character	C	2100	0	0	146	4
15	Murano	D	106	0	1	18	0
3	Cepheus	D	165	0	1	9	0
26	WireCloud	D	298	0	1	29	0
10	Interface Designer	D	1080	0	2	80	0
5	Cloud Portal	D	2220	0	4	150	1
16	OFNIC	E	40	2	0	2	0
8	Cosmos	E	160	5	2	7	0
22	Sagitta	E	540	6	0	35	2
11	IOT Broker	E	840	25	0	53	3
18	Pegasus	E	1260	2	0	88	2
14	Kiara	E	2880	8	1	175	37
7	Context Ckan	E	4800	1	14	363	0
21	Proton	E	8640	12	46	605	37
9	GeoServer	E	14400	50	27	493	82

Como se puede apreciar en la Tabla 2, los GEris que tienen mejor calificación no contienen ningún error. Después podemos encontrar un único GEri con una calificación B, ya que encontramos sólo *bugs* leves. A pesar de que el esfuerzo y el número de *bugs* está altamente correlacionado con un coeficiente de Spearman de 0,977, en el grupo de los GEri con calificación C, se observa que el esfuerzo en minutos de remediar los *bugs* no está directamente relacionado con el número de *bugs* que cada uno de estos tiene, incluso cuando todos los *bugs* tienen el mismo nivel de gravedad. Esto es debido a que el esfuerzo depende del tamaño del código donde se encuentra dicho bug. En el caso de los GEris con calificación D, su esfuerzo se incrementa debido a la gran cantidad de bugs encontrados más que por la gravedad de estos. Por último, tenemos el grupo de GEris con peor calificación en donde el esfuerzo aumenta considerablemente con el número de *bugs* de cualquier gravedad. Un descubrimiento interesante es que únicamente hay tres GEs calificados como A y uno como B, en total cuatro de los veintiséis estudiados. Esto parece una constatación del estado embrionario de la plataforma

[2] Definición de esfuerzo (*Remediation effort*) usada en el análisis: https://docs.sonarqube.org/display/SONAR/Metric+Definitions

en cuanto a cuidar su confiabilidad, ya que hay muchos componentes de calidad media-baja según este criterio.

Finalmente, podemos comentar ciertas particularidades de dicha tabla que ocurren en los saltos entre una calificación y otra. Por ejemplo, el GEri con identificador 22 y el GEri con identificador 17 tienen la calificación D y E respectivamente, aunque se observa claramente que el estado del 22 se podría considerar peor pues requiere un esfuerzo significativamente superior (6060 minutos frente a 40). El GEri con menor tiempo de esfuerzo necesario tiene una peor calificación debido a que, aunque tenga un menor número de *bugs*, dos de ellos son bloqueantes, haciendo que se encuentre directamente con la peor calificación posible. Naturalmente la conclusión es que necesitamos varias métricas y no únicamente este eje de confiabilidad para valorar el estado de la calidad global y precisar las mejoras necesarias. En la Figura 1 mostramos una gráfica de barras con el número total de bugs por cada GEri (ver ID en el eje de abscisas), ordenados por su valoración, siguiendo el mismo criterio que en la Tabla 2. También mostramos la mediana del número de *bugs* (30,5), puesto que su distribución en el conjunto de GEris no es normal. Además, mostramos un diagrama de sectores donde podemos observar cómo están distribuidos los GEris atendiendo a su calificación en el eje de confiabilidad.

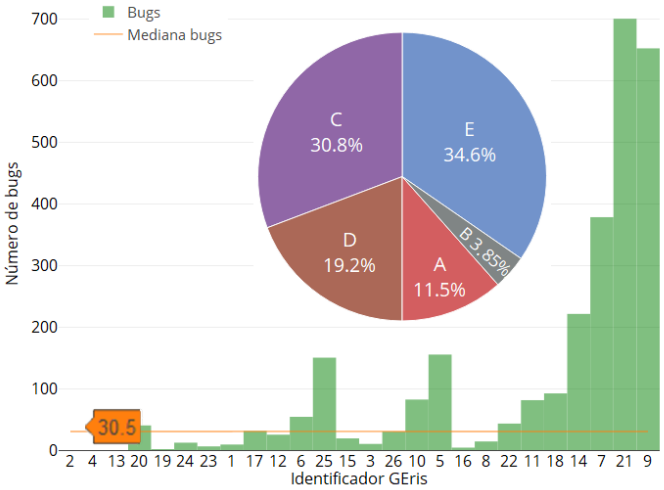

Figura 1. Bugs agregados y distribución de calificación respecto a eje de confiabilidad por cada GEri.

La Figura 1 muestra claramente que solo un 11 % de los GEris obtienen la máxima calificación, siendo un conjunto muy pequeño respecto al total. En el otro extremo, podemos observar que más de un tercio de los GEris (35 %) se evalúan con la peor calificación posible, siendo este el mayor grupo de todos. Podemos pues concluir que se requiere un esfuerzo considerable por parte de la

comunidad de FIWARE para mejorar la confianza en el correcto desempeño de sus componentes. Además, como el eje de abscisas del diagrama de barras está ordenado por calificación y esfuerzo, podemos observar que en líneas generales a medida que avanzamos en el eje se aprecia como la cantidad de *bugs* se incrementa. Aunque ya hemos comentado que existen excepciones, debido a que la localización de al menos un *bugs* de una severidad más alta, se traduce en obtener una peor calificación. Por tanto, parece claro que se necesita invertir tiempo en eliminar los *bugs* con severidad más alta para que las calificaciones de los GEris mejoren, mientras que la existencia de bus leves no parece que puedan interferir demasiado en el desempeño del software.

5.2. Eje de seguridad

Pasamos ahora a analizar los GEris desde el punto de vista de la seguridad, buscando las posibles vulnerabilidades que puedan estar presentes en los GEris seleccionados. Algunos ejemplos de vulnerabilidades son: contraseñas escritas directamente en el código, argumentos de funciones sin validar, etc. Por lo tanto, a mayor gravedad y cantidad de vulnerabilidades que se encuentren, mayor probabilidad de que se materialice una amenaza sobre dicho sistema software.

En la Tabla 3 mostramos los resultados de los análisis realizados respecto a la seguridad. En dicha tabla encontraremos los diferentes GEris ordenados acorde a la calificación obtenida. Dicha calificación está de nuevo comprendida entre los valores A y E. La calificación se establece de la siguiente forma: A si no se detecta ninguna vulnerabilidad, B si se detecta al menos una vulnerabilidad leve, C si al menos contiene una vulnerabilidad grave, D si al menos tiene una vulnerabilidad crítica y, finalmente, E si se detecta al menos una vulnerabilidad bloqueante. En el caso de que dos GEri obtengan la misma calificación, se tiene en cuenta el esfuerzo estimado necesario para solventar dichas vulnerabilidades. Además, mostramos el número de vulnerabilidades según su gravedad.

En la Tabla 3 tenemos un total de cinco GEris con la máxima calificación, es decir se puede confiar en la seguridad de su diseño. Por otro lado, en el conjunto de GEris con calificación B, se observa que el esfuerzo de remediar dichas vulnerabilidades no se relaciona directamente con la cantidad de vulnerabilidades. Por ejemplo, el GEri con identificador 25 tiene seis vulnerabilidades leves y requieren un esfuerzo total estimado de una hora para solventarlas. En cambio, el GEri con id 3, teniendo la mitad de vulnerabilidades, requiere el doble de esfuerzo. Esto se debe a que dichas vulnerabilidades afectan a una mayor cantidad de lineas de código, haciendo que el tiempo estimado de solventarlas sea mayor. También apreciamos ciertas particularidades, como en los casos de los GEris: Cloto, Cloud Rendering y Cosmos. Estos GEris tienen baja calificación pues, a pesar de que el número de vulnerabilidades es pequeño, la gravedad de algunas los sitúa en esa calificación. En el caso de Cosmos, por ejemplo, es conocida su dificultad de instalación y uso. Hay también sugerencias efectivas, como que el esfuerzo para enmendar estas vulnerabilidades es relativamente pequeño en comparación con otros GEris y deberían ser objeto de mejora cuanto antes.

Tabla 3. Resultados en el eje de seguridad para cada GEri

ID	GEris	Rating	Esfuerzo(min)	Vulnerabilidades			
				Bloqueante	Críticos	Graves	Leves
2	Business API Ecosystem	A	0	0	0	0	0
13	Keyrock	A	0	0	0	0	0
19	PEP Proxy Wilma	A	0	0	0	0	0
20	POI Data Provider	A	0	0	0	0	0
23	Sth Comet	A	0	0	0	0	0
17	Orion	B	30	0	0	0	6
25	Virtual Character	B	30	0	0	0	6
3	Cepheus	B	60	0	0	0	3
1	AuthzForce Server	B	90	0	0	0	5
12	IOT Discovery	B	310	0	0	0	25
11	IOT Broker	B	1500	0	0	0	134
14	Kiara	B	5280	0	0	0	481
4	Cloto	C	120	0	0	4	0
15	Murano	C	240	0	0	8	0
18	Pegasus	C	2940	0	0	11	116
24	Swift	C	5280	0	0	188	0
6	Cloud Rendering	D	30	0	4	0	1
10	Interface Designer	D	70	0	4	0	0
5	Cloud Portal	D	105	0	4	0	1
26	WireCloud	D	840	0	1	28	0
7	Context Ckan	D	1500	0	16	39	8
8	Cosmos	E	60	2	0	0	0
16	OFNIC	E	400	1	0	0	25
22	Sagitta	E	3300	2	0	20	141
21	Proton	E	3420	2	77	0	138
9	GeoServer	E	21600	48	20	0	837

En la Figura 2 mostramos un diagrama de barras con un agregado de las vulnerabilidades por cada GEri. El eje de abscisas muestra el ID de cada uno de los GEris ordenados por el mismo criterio que en la Tabla 3, es decir por calificación y, como segundo criterio, por esfuerzo. Así mismo, tendremos el número de vulnerabilidades total de cada GEri en el eje de ordenadas. También incluimos un diagrama de sectores donde se puede apreciar la distribución de los diferentes GEris en los diferentes niveles de seguridad.

En general, en la Figura 2 se aprecia que el total de vulnerabilidades no es muy grande en cada uno de los GEris. De hecho, la mediana en número de vulnerabilidades del conjunto de GEris es 6, aunque tenemos algunas claras excepciones de GEris con una gran cantidad de estas. En este aspecto destacan los tres peores GEris por calificación y por número de vulnerabilidades (los últimos en el eje de abscisas) y también destaca el GEri con ID 11 con una gran cantidad. Además, podemos apreciar que los diferentes GEris se han distribuido de forma bastante uniforme en los grupos de calificación, de hecho, hay 3 grupos formados por 5 GEris (19,2 %), y los otros niveles contienen 4 y 6 GEris. También observamos que los GEris con más vulnerabilidades corresponden con los peores dentro de cada grupo de calificación. Es decir, el peor de entre los GEris con la misma calificación es además el que mayor número de vulnerabilidades contiene.

5.3. Eje mantenibilidad

Esta subsección se centra en analizar cada uno de los GEris desde el punto de vista de la mantenibilidad. La mantenibilidad de un código será buena si

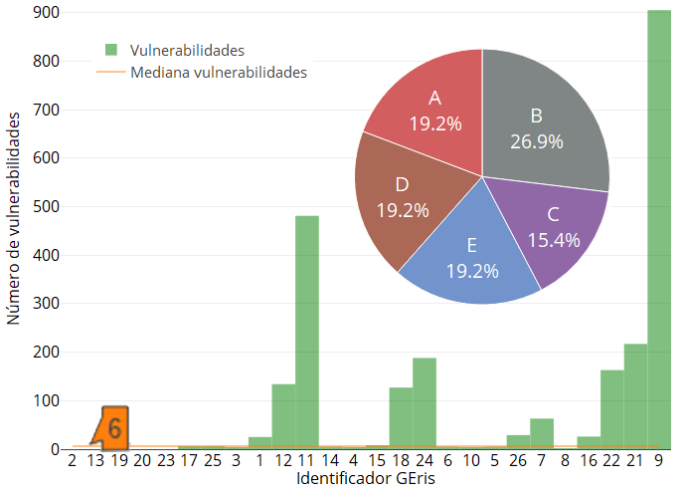

Figura 2. Seguridad en cada GEri ordenados por calificación, más su mediana.

se necesita poco esfuerzo para introducir un cambio en el código, ya sea para añadir una nueva funcionalidad o para arreglar algún tipo de error existente. En nuestro análisis vamos a detectar estructuras de código que indican violaciones fundamentales de diseño y que van a afectar negativamente a la calidad, como son los llamados *code smells*, que son además una fuente de errores. Algunos ejemplos de *code smells* son: definir varias veces la misma variable en vez de utilizar constantes, tener métodos demasiado complejos que pueden ser refactorizados, utilizar corchetes en sentencias anidadas, etc.

En la Tabla 4, podemos observar los resultados del análisis enfocados en el eje de mantenibilidad por cada GEri. Mostramos el ID, nombre de los GEris y la calificación otorgada a cada uno de ellos. Dicha calificación está comprendida entre los valores A y E. También tenemos en las siguientes columnas el número de *code smells*, el esfuerzo estimado en minutos para resolverlos y en la última columna aparece el ratio de deuda. Este parámetro es la proporción entre el coste en tiempo de resolver todos los *code smells* y el esfuerzo total del desarrollo del proyecto, que se describe en la Ecuación 1. Para esa estimación se utiliza como tiempo de desarrollo por líneas de código un valor constante de 28,8 minutos establecido por la herramienta SonarQube, y que se corresponde con las estimaciones clásicas sobre esfuerzo de 2 líneas de código a la hora [10].

$$ProporcionEsfuerzo = \frac{EsfuerzoFix}{EsfuerzoPorLinea * NumeroLineasCodigo} \quad (1)$$

Según el valor obtenido por cada uno de los GEris para el ratio del esfuerzo se le otorgará una calificación. En el caso de que el valor sea menor que un 5 % obtendremos una calificación A, si obtenemos un valor entre 6 % y 10 % la calificación será una B, en el caso de obtener un valor entre 11 % y 20 % la calificación será de C, si se obtiene entre 21 % y 50 % será una D y, finalmente, si es mayor que 50 % tendremos la calificación peor, E.

Tabla 4. Resultados en el eje de mantenibilidad para cada GEri

ID	Geris	Rating	Esfuerzo(min)	Code smells	P.Esfuerzo
23	Sth Comet	A	2	2	0.1 %
19	PEP Proxy Wilma	A	17	4	0.1 %
2	Business API Ecosystem	A	18	2	0.1 %
13	Keyrock	A	51	12	0.5 %
17	Orion	A	385	77	0.1 %
8	Cosmos	A	660	67	1.2 %
6	Cloud Rendering	A	660	77	0.1 %
4	Cloto	A	900	213	1.3 %
20	POI Data Provider	A	1.140	171	2.3 %
5	Cloud Portal	A	1320	216	0.1 %
1	AuthzForce Server	A	1680	193	0.7 %
16	OFNIC	A	2.520	478	2.0 %
15	Murano	A	3000	469	0.3 %
26	WireCloud	A	3420	414	0.1 %
10	Interface Designer	A	4440	277	0.3 %
12	IOT Discovery	A	4440	714	3.1 %
24	Swift	A	4800	711	0.6 %
25	Virtual Character	A	6720	499	0.7 %
3	Cepheus	A	7200	545	2.8 %
11	IOT Broker	A	12000	1775	1.7 %
7	Context Ckan	A	14880	2180	0.3 %
22	Sagitta	A	16320	1860	2.5 %
14	Kiara	A	23040	2868	1.8 %
21	Proton	A	25440	3208	0.3 %
18	Pegasus	A	25920	3350	2.3 %
9	GeoServer	A	284160	29172	1.8 %

Como se aprecia en la Tabla 4 todos los GEris obtienen la mejor calificación posible. Aunque hay claras diferencias en el número de *code smells* entre los primeros y los últimos, obtienen la misma calificación debido a que el ratio en ningún caso supera el 5 %. Este ratio se mantiene por debajo de dicho umbral ya que los *code smells* sólo aumentan como consecuencia de un desarrollo que va creciendo. Este umbral puede no ser muy útil en la práctica y de ahí que no nos sirva para distinguir diferentes grupos de mantenibilidad a pesar de que claramente habría que distinguirlos. Seguramente usar el esfuerzo o el número de *code smells* directamente fuese más útil.

El tamaño de cada GEri se puede consultar en la Tabla 1. Así, los GEris con los peores ratios son: los de ID 12 y 3, con 3.1 % y 2.8 % respectivamente. Sin embargo, estos no son los peores GEris respecto a la mantenibilidad ya que el GEri GeoServer es el que más esfuerzo estimado necesita debido a su tamaño, más de medio millón de lineas de código. De hecho, los tres GEris más grandes, con más de 100.000 lineas de código (Proton, GeoServer y Ckan), ocupan las últimas posiciones de la tabla. Por tanto, teniendo en cuenta el porcentaje de esfuerzo y el tiempo total deducimos que la mantenibilidad de GeoServer es preocupante, ya que necesita 592 jornadas laborales para solventar sus defectos.

6. Conclusiones y trabajo futuro

Tras haber realizado los análisis a los veintiséis GEris seleccionados tenemos una fuente única de datos objetivos para conocer el estado real de cada uno de ellos desde el punto de vista de la calidad interna, concretamente en los ejes de

confiabilidad, seguridad y mantenibilidad, que son parte del estándar ISO 25010. Respecto a la confiabilidad, tenemos una gran mayoría de GEris con las peores calificaciones posibles (D y E). Esto es debido a la gran cantidad y gravedad de los *bugs* encontrados. Aunque también es cierto que algunos GEris como: *Cloto*, *Cosmos*, *Murano* y *OFNIC* podrían obtener una mejor calificación con un esfuerzo inferior a una jornada laboral. Son dianas claras para un incremento rápido de confiabilidad a bajo esfuerzo.

Centrándonos en los resultados extraídos en el eje de seguridad, vemos que en esta ocasión no hay muchas vulnerabilidades en la mayoría de los casos; de hecho, en un 19 % de los GEris no detectamos ninguna (lo que no significa que no la haya) y consiguen la mayor calificación. Además, existen ocho de ellos que obtienen una calificación inferior debido a la gravedad de la vulnerabilidad encontrada y siendo solo necesario un esfuerzo inferior a una jornada laboral para solventarlos. Esto contrasta especialmente con el GEri Geoserver que necesitaría, al menos, 45 jornadas laborales para enmendar las vulnerabilidades detectadas.

En el eje de mantenibilidad algunos GEris necesitarían poco esfuerzo para conseguir eliminar los defectos detectados, por ejemplo 14 de ellos necesitaría menos de una semana de trabajo para eliminar todos. Aún así, el eje de mantenibilidad es el que requiere mayor esfuerzo total. Poniendo los datos en contexto, la media de esfuerzo para confiabilidad es la menor con 1584 minutos (3,3 jornadas laborales), seguida de seguridad que requeriría 1816 minutos (3,8 jornadas laborales) y mantenibilidad 17120 minutos (35,6 jornadas laborales). Esto posiblemente es debido al carácter heterogéneo del equipo de desarrolladores de FIWARE, que afecta directamente a la mantenibilidad del código.

Con respecto a la valoración global de los GEris en los diferentes ejes analizados, aunque la mayoría requieren un esfuerzo para resolver todos sus problemas en conjunto, hay hasta cuatro GEris que se mantienen con un esfuerzo menor a una jornada laboral. Estos que son:*Business API Ecosystem*, *PEP Proxy Wilma*, *Keyrock* y *Sth Comet*, merecen resolverse cuanto antes. Otro punto que resaltar es el caso de Orion, un *ontext broker* para datos usado en prácticamente la totalidad de aplicaciones FIWARE y que aparece con una baja calidad en cuanto a confiabilidad y seguridad. Es también una diana de mejora urgente porque afectará positivamente a toda la comunidad.

Si nos centramos en los peores GEris, en la Tabla 5 podemos observar el número de defectos totales hallados y el esfuerzo requerido para solventar estos problemas de los 5 peores GEris. Destaca especialmente GeoServer como peor GEri con un esfuerzo necesario de 667 jornadas laborales, aunque los demás necesitan también al menos dos meses de trabajo. A la vista de todos los datos podemos concluir que se debería prestar más atención a las métricas de calidad de estos elementos que van a ser la base de los servicios y aplicaciones desarrolladas para las ciudades inteligentes, que implicarán a millones de ciudadanos europeos.

El trabajo futuro que consideramos más prometedor en esta línea se puede dividir en tres fases. Primero, planeamos estudiar los ejes del estándar ISO 25010 que han quedado fuera de este trabajo como son: Funcionalidad, Eficiencia, Usabilidad, Compatibilidad y Portabilidad. Segundo, tras este análisis de los

Tabla 5. Resumen de esfuerzo para los cinco peores GEris.

			Esfuerzo(min)			
ID	GEris	Defectos	Confiabilidad	Seguridad	Mantenibilidad	Total
7	Context Ckan	2621	4800	1500	14880	21180
18	Pegasus	3569	1260	2940	25920	30120
14	Kiara	3570	2880	5280	23040	31200
21	Proton	4125	8640	3420	25440	37500
9	GeoServer	31909	14400	21600	284160	320160

ejes queremos modificar los GEris para eliminar el mayor número de defectos posible. Finalmente, llevaríamos a cabo un estudio de la calidad externa de aplicaciones usando las versiones con y sin defectos de algunos de los GEris. De esta forma podremos poner de manifiesto si estas recomendaciones en cuanto a calidad impactan positivamente cuando las aplicaciones se encuentran en uso.

Acknowledgements

Esta investigación ha sido parcialmente financiada por CELTIC C2017/2-2 en colaboración con las empresas EMERGYA y SECMOTIC en los contratos #8.06/5.47.4997 y #8.06/5.47.4996. También agradecemos el apoyo del Ministerio de Economía y Competitividad y de los fondos FEDER, proyectos: TIN2014-57341-R (http://moveon.lcc.uma.es), TIN2016-81766-REDT (http://cirti.es) y TIN2017-88213-R (http://6city.lcc.uma.es)

Referencias

1. Tan, L., Wang, N.: Future internet: The internet of things. In: 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE). Volume 5. (Aug 2010) V5–376–V5–380
2. Di Cola, S., Tran, C., Lau, K.K., Celesti, A., Fazio, M.: A heterogeneous approach for developing applications with fiware ges. In Dustdar, S., Leymann, F., Villari, M., eds.: Service Oriented and Cloud Computing, Cham, Springer International Publishing (2015) 65–79
3. Paulk, M.C., Curtis, B., Chrissis, M.B., Weber, C.V.: Capability maturity model, version 1.1. IEEE Softw. **10**(4) (July 1993) 18–27
4. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI for Development: Guidelines for Process Integration and Product Improvement. 3rd edn. Addison-Wesley (2011)
5. IEEE: Standard for software quality assurance processes. IEEE Std 730-2014 (Revision of IEEE Std 730-2002) (June 2014) 1–138
6. ISO annual report: : Navigating a world in transition. Technical report, ISO (2016)
7. ISO/IEC: ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Technical report (2010)
8. Wagner, S.: Software product quality control. (2013)
9. Siriwardena, P.: Security by Design. In: Advanced API Security. Volume 58. (2014)
10. Boehm, B.W.: Improving Software Productivity. Computer **20**(9) (1987) 43–57