Marco de caracterización para la cuantificación de la calidad en componentes web

Miguel Ortega Moreno
Investigador de la Universidad Politécnica de Madrid
mortega@conwet.com

David Lizcano Casas
Profesor doctor de la Universidad a Distancia de Madrid, UDIMA
david.lizcano@udima.es

Genoveva López Gómez
Profesora titular de la Universidad Politécnica de Madrid
genoveva.lopez@upm.es

Extracto

El desarrollo web está evolucionando muy rápidamente tanto en tecnología como en los servicios que ofrece. La ingeniería de *software* que subyace, por su parte, no ha sido capaz de adaptarse a los nuevos problemas que presenta esta evolución. La encapsulación y la interoperabilidad han permitido el nacimiento de nuevos paradigmas de desarrollo donde usuarios finales sin conocimientos de programación, intermediarios y proveedores crean, explotan y reutilizan elementos conocidos como «componentes web». Sin embargo, los componentes web todavía carecen de un marco de calidad que sea aceptado universalmente, capaz de cuantificar lo bueno o malo que es un componente.

Palabras clave: componentes web; métricas de *software*; ingeniería de *software*.

Fecha de entrada: 17-05-2019 / Fecha de revisión: 26-06-2019 / Fecha de aceptación: 28-06-2019

Cómo citar: Ortega Moreno, M., Lizcano Casas, D. y López Gómez, G. (2019). Marco de caracterización para la cuantificación de la calidad en componentes web. *Tecnología, Ciencia y Educación*, *14*, 39-70.

Characterization framework for the quantification of quality in web components

Miguel Ortega Moreno

David Lizcano Casas

Genoveva López Gómez

Abstract

Web development is rapidly evolving both in technologies available and services being offered. Software engineering that lies behind has not been quite able to keep up with the new challenges that have to be addressed. However, encapsulation and interoperability have been pushed forward with a new paradigm: web components. Using package management tools, developers with different level of programming expertise reuse web components from many sources. Nevertheless, web components still lack a quality framework universally accepted. It is difficult to assess how good or bad a component is.

Keywords: web components; software metrics; software engineering.

Citation: Ortega Moreno, M., Lizcano Casas, D. y López Gómez, G. (2019). Characterization framework for the quantification of quality in web components. *Tecnología, Ciencia y Educación*, *14*, 39-70.

Sumario

Nota: este trabajo se enmarca dentro de la Convocatoria de Ayudas a Proyectos de I+D+i 2017 de la Fundación Hergar (categoría: Investigación aplicada y tecnológica en ingenierías).

1. Introducción

La web ha evolucionado hacia un medio colaborativo donde el contenido es creado tanto por desarrolladores expertos como por usuarios carentes de conocimiento en las tecnologías subyacentes. La proliferación de medios que facilitan la divulgación de nuevos elementos como la encapsulación y la interoperabilidad proporcionada por los componentes web, sumado a la necesidad de generar contenido de manera rápida y constante, ha provocado que la calidad sea un aspecto fundamental a la hora de reutilizarlos.

Gran parte de este auge está ligado al concepto de *web components* (componentes web). Estos suponen un cambio a la hora de compartir contenido, proporcionando a los desarrolladores la capacidad de crear estructuras en repositorios colaborativos que pueden ser reutilizadas por terceros para componer sus propios componentes y aplicaciones. La ocultación de los detalles de implementación es la clave para convertirse en el futuro del desarrollo web.

A pesar de esta necesidad, actualmente no existe ningún marco de calidad que sea capaz de adaptarse de manera adecuada a los nuevos requisitos impuestos por paradigmas como *end-user development* (EUD), donde el usuario final no cualificado es quien genera nuevo contenido.

Actualmente no existe ningún marco de calidad que sea capaz de adaptarse de manera adecuada a los nuevos paradigmas

Junto a la ausencia de marcos de calidad, también existe un problema derivado de la heterogeneidad del nivel técnico de los desarrolladores que publican contenido en los repositorios colaborativos. La inmensidad de componentes publicados dificulta discernir cuál de todos ellos cumple en mejor medida la funcionalidad que se está buscando. De igual forma, las aplicaciones construidas con componentes web adolecen de la ausencia de calidad en las piezas principales y de mecanismos de comunicación entre las distintas partes. Todo esto seguido de no poder determinar el nivel de compatibilidad existente entre las diferentes piezas que componen una aplicación, tanto a nivel de visualización como de funcionalidad y orquestación.

Se pretende encontrar un conjunto de métricas que sean capaces de determinar el grado de aceptación de un componente por parte de un usuario final

Con el objetivo de resolver esta carencia, se pretende encontrar un conjunto de métricas que sean capaces de determinar el grado de aceptación de un componente por parte de un usuario final. Para ello se partirá de los estándares tra-

dicionales que servirán como punto de referencia para la selección de las métricas que, a través de la validación por parte de los usuarios finales, determinarán el impacto que tiene cada una de ellas sobre la calidad que perciben.

2. Métricas de calidad asociadas a los componentes web

> Para determinar qué grado de calidad es alcanzado en un proyecto, se establecieron marcos de referencia que indican qué aspectos son fundamentales y qué se ha de considerar como parte de su definición

A lo largo de la evolución de la web se han ido creando y adaptando diferentes estándares que promueven el desarrollo de *softwares* de calidad. Para determinar qué grado de calidad es alcanzado en un proyecto, se establecieron marcos de referencia que indican qué aspectos son fundamentales y qué se ha de considerar como parte de su definición. Existen varias normas que marcan los fundamentos de las métricas básicas que se han de utilizar. Estas pueden ser entendidas como formales, ya que definen de manera genérica qué se debe comprender por cada una de las métricas, pero no llegan a establecer fórmulas que determinen los valores objetivo que permitan analizar el grado de cumplimiento en cada una de ellas. Muchos de estos marcos establecen un modelo jerárquico en función de las medidas de calidad que intentan establecer, organizándolas en diferentes niveles. En ellos se hace una alusión a dos conceptos:

- **Calidad funcional.** Nivel de cumplimiento del *software* en función de las especificaciones dadas.
- **Calidad estructural del *software*.** Relacionado con el cumplimiento de requisitos no funcionales, tales como la seguridad, la mantenibilidad o la complejidad.

Los marcos tradicionales que se han venido utilizando como referencia son ISO/IEC 9126, que posteriormente ha derivado en versiones más avanzadas: ISO/IEC 25010 y *web quality model* (WQM).

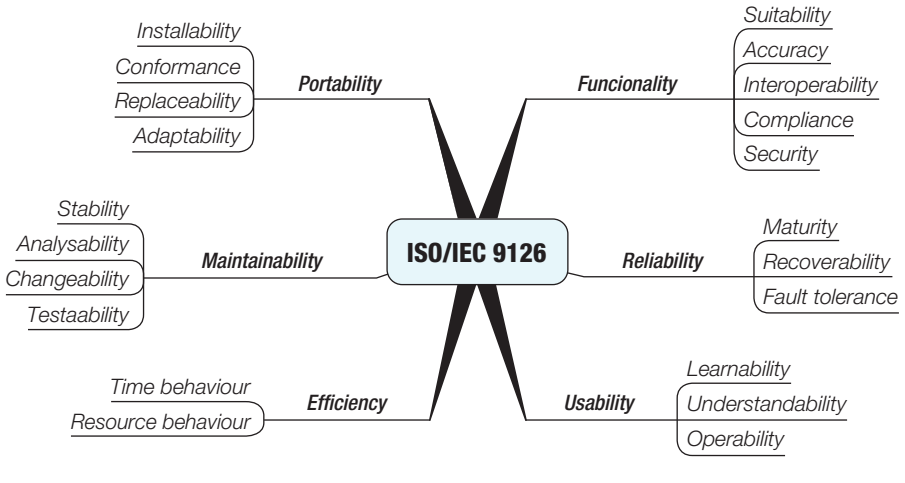
2.1. ISO/IEC 9126

Este estándar (Bevan, 1997) ha sido el marco de referencia durante muchos años para la evaluación de la calidad del *software* hasta la aparición de la norma 25010. En este estándar se establecen los aspectos que se deben valorar y que debe cumplir cualquier *software* si se quiere considerar de calidad (véase figura 1). Los aspectos que hay que tener en cuenta son una combinación de características que pueden ser evaluadas y medidas,

además de que hay que diferenciar entre tres perspectivas diferentes en función del tipo de calidad que se intente analizar (Cappiello, Daniel y Matera, 2009; Padayachee, Kotze y Van der Merwe, 2010):

- **Calidad interna.** Basada en las propiedades intrínsecas del *software*, entendidas estas como una caja blanca. La calidad es independiente del entorno en el que se encuentra y de la interacción con el usuario, y puede ser medida directamente del código fuente del *software* y del flujo que sigue en su ejecución.
- **Calidad externa.** Entendida como un modelo de caja negra que hace referencia al comportamiento de la interfaz de los servicios ofrecidos por el sistema y a su ejecución dentro de un entorno determinado.
- **Calidad en uso.** Asociado a la capacidad del sistema para permitir cumplir con aspectos relacionados con la efectividad, la productividad, la seguridad y su satisfacción dentro de un contexto de uso del propio sistema.

Figura 1. **Estructura de la ISO/IEC 9126**



Fuente: Bongais (2017).

2.2. ISO/IEC 25010

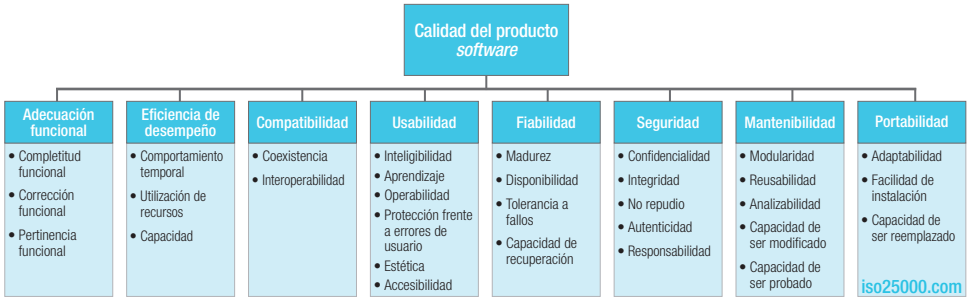
La ISO/IEC 25010 (ISO, 2019) es resultado de la evolución de la ISO/IEC 9126 (ISO, 2011). Esta norma revisa el marco propuesto, modificando y añadiendo nuevas caracte-

rísticas que deben ser analizadas para determinar la calidad del *software*; características que, al igual que en su predecesora, se dividen en subcategorías. Este modelo establece la calidad basándose en tres marcos diferentes, según el ámbito en que vaya a ser aplicado:

- **Modelo de calidad de uso.** Orientado a los *stakeholders*, a los que va dirigido el producto *software*.
- **Modelo de calidad de producto.** Centrado en los propios desarrolladores del *software*.
- **Modelo de calidad de datos.** Recogido en la ISO/IEC 25012.

Se pueden encontrar un total de ocho características diferentes, ampliando en dos las de su predecesora, donde algunas de las categorías ya definidas han sido reformuladas y adaptadas. Estas características se pueden ver en la figura 2.

Figura 2. **Estructura de la ISO/IEC 25010**



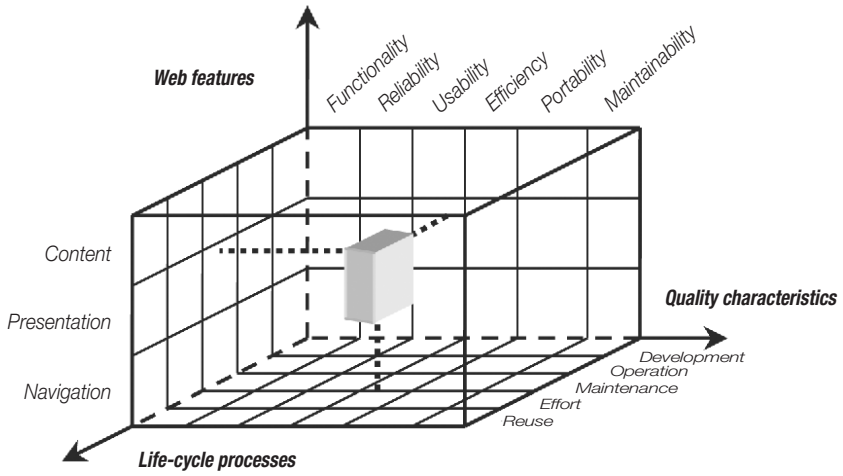
Fuente: ISO (2019).

2.3. WQM

Este modelo de calidad web (Calero, Ruiz y Piattini, 2005) establece un marco para tratar de estimar la calidad que tienen los sitios web a través de una evaluación basada en una matriz de tres dimensiones en forma de cubo (véase figura 3). La matriz está compuesta por aquellos aspectos que se consideran más relevantes para que una página web tenga calidad. Los aspectos han sido el resultado de la clasificación de las diferentes métricas web que se consideraban hasta el momento (Calero, Ruiz y Piattini, 2004). El resultado de la clasificación dio lugar a tres dimensiones diferentes:

- Características de la web.
- Ciclo de vida del proceso.
- Características de la calidad.

Figura 3. **Cubo de las dimensiones del WQM**



Fuente: Chávez (2009).

2.3.1. Características de la web

En esta dimensión se incluyen tres aspectos clásicos de la web:

- Contenido.
- Presentación.
- Navegación.

La navegación es importante en el diseño de elementos, ya que hace posible que los usuarios sean capaces de acceder a la información que necesitan y hacer que esa información sea fácil de encontrar. El contenido y la presentación son dos aspectos fundamentales a la hora de analizar una página web, ya que la combinación de ambos aspectos determina la facilidad a la hora de usar una página web. En el contenido están incluidos todos los elementos que componen una aplicación web, desde los elementos básicos de texto hasta los programas que lo componen.

2.3.2. Características de la calidad

La calidad está basada en el modelo Quint2 (Niessink, 2002), una revisión de la norma ISO/IEC 9126. Define una estructura jerárquica basada en seis características, con sus diferentes divisiones, tal y como hemos visto con anterioridad. Además de incluir todas las características y subcaracterísticas del modelo Quint2, se añade una subcategoría en la portabilidad que hace referencia a la coexistencia de dos *softwares* en un mismo entorno en el que comparten recursos. Aunque está ausente en Quint2, la ISO/IEC 25010 ya hacía referencia a este aspecto en la característica de compatibilidad.

2.3.3. Ciclo de vida del proceso

El ciclo de vida del proceso fue incluido debido a la consideración de que el equipo de desarrollo está formado por un equipo multidisciplinar con diferentes habilidades y conocimientos. Por lo tanto, en esta dimensión se analiza el ciclo de vida de la web según lo indicado en la ISO/IEC 12207-1 (Singh, 1996). Se incluyen tres procesos principales:

- Proceso del desarrollo.
- Proceso de explotación, que incluye soporte a usuarios.
- Mantenimiento del proceso, que incluye las actualizaciones y los cambios necesarios.

A pesar de la existencia de estos marcos, observamos que no son capaces de adaptarse al desarrollo web basado en componentes, ni a los *mashup* construidos con estos, debido, principalmente, a un enfoque destinado a la construcción de aplicaciones completadas como páginas web o como soluciones híbridas para dispositivos móviles.

3. Selección y aplicación de las métricas a componentes aislados

Como hemos indicado, los estándares tradicionales no se pueden adaptar a esta nueva concepción de la web, ya que hablan de la construcción de aplicaciones formadas por el conjunto de todas las características deseadas. En contraste, los componentes disponen solo de características concretas y en mayor medida pequeñas, además de centrarse principalmente en su funcionalidad, dejando a un lado la composición en sí misma.

> Con el modelo de desarrollo basado en componentes se promueve la reutilización de contenido creado por terceros, lo que provoca una pérdida del control en el proceso de desarrollo

Con el modelo de desarrollo basado en componentes se promueve la reutilización de contenido creado por terceros, lo que provoca una pérdida del control en el proceso de desarrollo e incluso en ciertos aspectos de adaptación e integración. Además, en cada uno de ellos se asume que estos componentes funcionan correctamente, ya que se quieren utilizar en un entorno de producción.

A pesar de que no se adaptan de manera adecuada, sí que sirven como punto de partida; por esto, las métricas que se estudian son una referencia a estos estándares, principalmente al ISO/IEC 25010 y al WQM, contrastadas con la opinión de expertos en el desarrollo basado en componentes.

Las métricas identificadas en diferentes entrevistas con expertos coinciden con cada una de las métricas descritas dentro de la ISO/IEC 25010. Las métricas utilizadas serán:

- Completitud.
- Latencia.
- Complejidad ciclomática.
- Mantenibilidad.
- Usabilidad.
- Seguridad.
- Complejidad estructural.
- Estabilidad.
- Portabilidad.

También se ha incluido la métrica de «refresco de datos», que lidia con la volatilidad de los datos y el tiempo en el que la información nueva es mostrada, ya que actualmente la web consume de servicios de terceros para complementar su funcionamiento.

3.1. Completitud

La completitud de datos se define como la cantidad de información que muestra el componente en relación a la cantidad ofrecida por un servicio a través de una API, que, en la mayoría de los casos, será un servicio de terceros.

Se centra en el caso de componentes que presentan un conjunto de datos para que sean consumidos por los usuarios del componente o incluso por otros componentes. La cantidad de información que proporciona puede ser diferente en función de diversos motivos, como la cantidad de permisos que tiene la aplicación respecto a un usuario o a la propia API.

De esta forma, se ha definido que la completitud de los datos de un componente se puede medir a través de la cantidad de información que ofrece un componente en relación a la cantidad de datos que un servicio es capaz de servir. El origen de la ausencia de información no se tiene en cuenta a la hora de determinar su error. La completitud puede ser medida mediante la siguiente fórmula matemática, en la que se expresa el porcentaje de completitud que ofrece un determinado componente (véase ecuación 1).

Ecuación 1. **Métrica de la completitud**

$$\text{Completitud} = \frac{CDC}{CDA} \times 100$$

Donde:

- *CDC* = Conjunto de datos ofrecidos por el componente.
- *CDA* = Conjunto de datos ofrecidos por la API.

Para clasificar los valores dentro de unos rangos controlados, se ha aplicado una correlación de los valores a cinco niveles diferentes, dependiendo del nivel de completitud ofrecido por el componente, siendo un valor de 5 aquel componente que ofrece un 100 % de la completitud y un valor de 1 si ninguno de los datos otorgados es correcto. La expresión que define esta clasificación es la que se puede ver en la ecuación 2.

Ecuación 2. **Fórmula para la clasificación de la completitud**

$$\text{Nivel de completitud} = 5 \times (1 - \% \text{ completitud})$$

Donde:

- Nivel de completitud = Un valor entre 1 y 5.
- Completitud = Se obtiene con la fórmula anterior (véase ecuación 1).

3.2. Latencia

Se trata de una métrica relacionada con el consumo de datos proporcionados por terceros a través de un servicio externo. Es una consecuencia directa de la evolución de la web y el modo en que los usuarios utilizan la información. Las plataformas desde las que se consume contenido han ido transformándose a favor del acceso a la web me-

diante móvil y es en estos dispositivos donde la latencia afecta más, puesto que las velocidades de conexión varían y son bastante inferiores a la velocidad disponible en los equipos de sobremesa.

De acuerdo con un estudio realizado por <gomez.com> y <akamai.com> (Patel, 2018), el 47 % de los usuarios espera que una página web esté disponible en menos de 2 segundos, el 40 % abandonará la página si el tiempo de carga es superior a 3 segundos y 1 segundo de retraso o 3 de espera provocan que la satisfacción de un cliente al usar un servicio disminuya en un 16 %.

Por estos motivos, la latencia se puede considerar una de las métricas importantes a la hora de medir la calidad de un componente web que proporciona acceso a un recurso externo. Mediante esta métrica se pretende cuantificar el tiempo que transcurre desde que se inicia la petición de los datos hasta que se recibe la información requerida. De esta forma se puede entender que la latencia está compuesta por tres factores:

- Tiempo que tarda desde que la petición sale del cliente hasta el proveedor del servicio.
- Tiempo de procesamiento de una petición por parte del servidor.
- Tiempo que tarda la petición en volver desde el servidor hasta el cliente.

El tiempo de respuesta entre cliente y servidor se ve afectado por el estado de la red y el tipo de conexión que se emplee en ese momento. Por su parte, el tiempo de respuesta depende en gran medida de la capacidad del proveedor para procesar la petición de un cliente, atendiendo a posibles problemas de tráfico masivo o saturación del servidor.

Debido a la ausencia de puntos de referencia para entender si una latencia dentro de un componente es buena o mala, se ha medido mediante la diferencia entre el tiempo medio que tarda el servicio en contestar a peticiones realizadas directamente, frente al tiempo que tarda el componente que requiere de ese servicio. Para minimizar posibles desviaciones en momentos puntuales, se han realizado pruebas en diversos momentos del día (véase ecuación 3).

Ecuación 3. **Métrica de latencia**

$$\text{Latencia} = \frac{\sum_{i=0}^{n-1} \min\left(0, T_{\text{componente}} - T_{\text{host}}\right)}{n}$$

3.3. Complejidad ciclomática

La estructura del código y su calidad de programación son aspectos que siempre han importado a los desarrolladores. La complejidad es una métrica ampliamente conocida debido a su independencia del lenguaje en el que se está programando. En el ámbito de los componentes web, los proveedores y los integradores prestan principal atención al modo en que esté programado el código, puesto que en la mayoría de los casos se tiene que entender y modificar para adaptarlo al entorno en el se quiere ejecutar.

La complejidad es una métrica ampliamente conocida debido a su independencia del lenguaje en el que se está programando

Muchos desarrolladores intentan reducir la complejidad del código en la medida de lo posible obligando a realizar un desarrollo basado en módulos para facilitar la comprensión del mismo. Para estimar el valor de la complejidad, se utilizan la cantidad de caminos que se encuentran en cada uno de los bloques de código que existen en el programa.

Una vez calculados los valores de la complejidad ciclomática, en la Carnegie Mellon University (Bray, Brune, Fisher, Foreman y Gerken, 1997) se realizó una clasificación de los valores en función de cómo se comporten los programas en relación a este valor (véase cuadro 1).

Cuadro 1. **Evaluación de los riesgos según el nivel de complejidad**

Complejidad ciclomática	Evaluación del riesgo
1-10	Programa simple (sin mucho riesgo).
11-20	Más complejidad (riesgo moderado).
21-50	Complejo (programa de alto riesgo).
> 50	Programa no analizable (riesgo muy elevado).

Fuente: elaboración propia.

Los rangos definidos por dicha universidad han servido como punto de partida para el cálculo de la métrica, sin embargo se ha realizado una variación de los diferentes rangos. En el caso del rango de valores entre 21-50 se ha dividido en dos, dando como resultado un

nivel para los valores de 21-35 y otro para los valores de 36-50. Esta división se ha realizado con la intención de crear cinco niveles diferentes, que es la referencia utilizada. Para el cálculo del valor de la complejidad, se ha utilizado la ecuación definida por McCabe (1976) (véase ecuación 4).

Ecuación 4. **Métrica de complejidad ciclomática**

$$\text{Complejidad ciclomática} = E - N + P$$

Donde:

- E = Número de aristas.
- N = Número de vértices.
- P = Número de componentes conectados.

3.4. Mantenibilidad

Una de las métricas derivadas de la complejidad ciclomática es la matenibilidad. Esta métrica está relacionada con la capacidad de mantener el código bajo un correcto funcionamiento. Influye la dificultad que existe en el componente para ser cambiado y entendido.

La mantenibilidad es la métrica que está relacionada con la capacidad de mantener el código bajo un correcto funcionamiento

Está claramente alineado con los objetivos que debe tener un desarrollador, puesto que son ellos los que en mayor o menor medida tienen que tratar con los componentes, ya sea para construirlos o integrarlos. En ambos casos, se han de utilizar otros componentes, por lo que la sencillez de arreglar, adaptar y entender su funcionamiento es primordial a la hora de elegir qué componente es mejor para su construcción.

Al igual que en la complejidad, existe un índice de mantenibilidad que fue diseñado en 1991 (Gill y Kemerer, 1991; Coleman, Ash, Lowther y Oman, 1994). Este índice define un rango que va desde el infinito negativo (– ∞) hasta 171, partiendo de una base logarítmica. Esta métrica tiene en cuenta la cantidad de líneas de código, entendidas como líneas de código lógicas, la complejidad ciclomática, calculada con la métrica anterior, y las medidas de Halstead.

La medida de Halstead fue introducida por Maurice Howard Halstead en 1976 (McCabe, 1976). Dicha medida se basa en el número de operadores y operandos que contiene un

código fuente. Partiendo de estas dos variables, se definen diferentes medidas, como el vocabulario del programa, la longitud del programa, la longitud calculada, el volumen, la dificultad y el esfuerzo (véase ecuación 5).

Ecuación 5. **Métricas de Halstead**

Operadores y operandos	Diferentes medidas
• $n1$ = Número de operaciones distintas. • $n2$ = Número de operandos distintos. • $N1$ = Número de operadores totales. • $N2$ = Número de operandos totales.	• Vocabulario del programa (n) = $n1 + n2$. • Longitud del programa (N) = $N1 + N2$. • Longitud calculada ($\check{N}$) = $n1 \times \log_2 n1 + n1 \times \log_2 n1$. • Volumen ($V$) = $N \times \log_2 n$. • Dificultad (D) = $\frac{n1}{2} \times \frac{N2}{n2}$. • Esfuerzo ($E$) = $D \times V$. • Tiempo requerido para programar (T) = $\frac{E}{18}$. • Número de bugs (B) = $\frac{V}{3.000}$.

Finalmente, la ecuación utilizada para calcular la mantenibilidad emplea todas las variables introducidas anteriormente y establece un valor comprendido entre 0 y 100. Al igual que en las anteriores métricas se ha realizado una escala del 1 al 5, que va desde una mantenibilidad de 0 para el valor 1, a una mantenibilidad de 25 para la escala 2, hasta una mantenibilidad de 100 para el valor 5 (véase ecuación 6).

Ecuación 6. **Métrica de mantenibilidad**

$$M = Max(0, (171 - 3{,}42 \times ln(E) - 0{,}23 \times ln(Cc) - 16{,}2 \times ln(loc)) \times 100/171)$$

3.5. Usabilidad

Marca la facilidad de un usuario para interactuar con un componente que tiene un interfaz visual. A pesar de su importancia, es difícil cuantificar el grado de usabilidad que tiene una aplicación, ya que las necesidades y la experiencia de usuario son muy diversas entre los

diferentes usuarios de una aplicación. A lo largo del desarrollo *software*, se han descrito patrones de diseño que son ampliamente aceptados por los usuarios y que, en su mayoría, definen una serie de metáforas que facilitan la comprensión de lo que está ocurriendo cuando interactúan con la aplicación.

El modo de medir la usabilidad es una tarea difícil, ya que aún no se han definido métricas que la describan. Por este motivo se ha utilizado la accesibilidad, una de las subcategorías descritas dentro de la norma ISO/IEC 25010 y que permite tener una leve aproximación a lo que los usuarios entienden como usabilidad.

Dentro de ella, existen normas y recomendaciones que se pueden seguir para determinar el grado de accesibilidad de la aplicación para los usuarios con diferentes necesidades.

Se han utilizado dos normas diferentes:

- WCAG.
- A11Y.

Para determinar una fórmula que sea capaz de describir la usabilidad, se han clasificado las normas en función de categorías y se les ha asignado un peso (véase ecuación 7).

Ecuación 7. **Métrica de accesibilidad**

$$\text{Accesibilidad} = \frac{\sum_{i=0}^{n-1} max\left(0, \frac{C_i - E_i \times W_i}{C_i}\right)}{n}$$

Donde:

- C_i = Número de ítems en la categoría n.
- E_i = Número de errores en la categoría n.
- W_i = Peso de la categoría n.

3.6. Seguridad

La seguridad es una de las métricas más importante cuando se está tratando con componentes desarrollados por terceros y se consumen datos personales o de acceso restringido. Si algunos de estos componentes presenta algún fallo de seguridad, puede provocar que futuros componentes desarrollados sobre estos hereden esos problemas.

La seguridad en los componentes es un campo muy amplio y su medición es compleja, ya que en muchos casos el usuario final es un factor determinante. Por este motivo, se ha decidido llevar a cabo una aproximación por medio de los permisos que un componente requiere al usuario para acceder a sus datos en diferentes redes sociales.

Si observamos cómo se proporcionan permisos en la mayoría de las redes sociales, se utiliza un protocolo de autenticación conocido como Oauth 2.0 (Hardt, 2012). Para la implementación de este protocolo se suelen emplear diferentes grados de permisos, y serán estos los que se analizarán para determinar el nivel de seguridad del componente.

Se contemplan permisos de dos tipos, en función de los privilegios que otorguen:

- **Permisos de lectura.** Solo permiten consumir información del usuario, como sus datos en las diferentes redes sociales o sus publicaciones.
- **Permisos de escritura.** Dan acceso a una tercera entidad para publicar, eliminar o modificar contenido en las diferentes redes sociales, como puede ser crear nuevas entradas o eliminar datos de un calendario.

Se ha asumido que los permisos de escritura son potencialmente más peligros que los de lectura, puesto que autorizan a realizar acciones en nombre de un usuario, lo que puede implicar actividades con éticas contrarias a la de su usuario o sin su consentimiento. Para calcular el valor de la métrica, se utiliza la cantidad de permisos concedidos al componente y que no son necesarios para la finalidad que pretende realizar. Los sobrepermisos de escritura serán más penalizados que los sobrepermisos de lectura (véase ecuación 8).

Ecuación 8. **Métrica de seguridad**

$$\text{Seguridad} = 5 - \frac{r}{R} - k \times \frac{w}{W}$$

Donde:

- r = Cantidad de sobrepermisos de lectura pedidos.
- w = Cantidad de sobrepermisos de escritura pedidos.
- R = Cantidad de permisos de lectura disponibles en una determinada API.
- W = Cantidad de permisos de escritura disponibles en una determinada API.
- k = Penalización en los permisos de escritura.

3.7. Complejidad estructural

La forma en la que se implementan los componentes implica la reutilización de contenido mediante la importación del mismo. La complejidad estructural es la métrica que mide la cantidad de importaciones que tiene un componente tanto a nivel directo como en profundidad. Visualizando las dependencias como un grafo se pueden observar las consecuencias que tiene la utilización de un determinado componente y cuánto contenido extra se necesita para que funcione correctamente.

Es importante conocer su estructura por diferentes motivos. Uno de ellos está relacionado con su peso total. Los componentes son usados dentro de un entorno web donde cada descarga de documentos tiene un impacto directo en el rendimiento de la aplicación y en la cantidad de datos que se deben consumir para conseguir la página web completa. El tamaño total de un componente no es solo el que ocupa el mismo, sino también el que ocupan todas sus dependencias. A su vez, las dependencias necesitan pedir nuevo contenido al servidor. Este no es solicitado hasta que no se analiza el propio componente, por lo que la descarga de mucho contenido en conexiones lentas puede provocar un aumento en el tiempo de carga de una aplicación web completa.

Los componentes son usados dentro de un entorno web donde cada descarga de documentos tiene un impacto directo en el rendimiento de la aplicación y en la cantidad de datos que se deben consumir para conseguir la página web completa

Además, las propias dependencias suelen ser creadas y mantenidas por terceros, lo que supone un aumento de los puntos de fallo. La mayoría de desarrolladores que mantienen un componente suelen cambiar el interfaz de manera más o menos constante, actualizándolo a los nuevos estándares. Conocer la estructura de un componente es necesario para saber si se están utilizando dependencias que han cambiado y pueden suponer un punto de fallo o, incluso, que se sabe que contienen fallos de seguridad.

Tal y como se puede ver en el experimento llevado a cabo por Gilbertson (2018), es muy importante conocer qué dependencias tiene un componente, ya que puede contener código malicioso camuflado tras una funcionalidad, *a priori*, útil. Este código malicioso puede ser una dependencia de una dependencia, y el desarrollador puede ignorar que existen estos problemas, puesto que no son agregadas directamente por él.

A través de esta métrica lo que se quiere analizar es el impacto que tiene la cantidad de dependencias que posea un componente. Su análisis se realiza en profundidad, es decir, se analizan tanto las dependencias directas como las dependencias de las dependencias. Tan solo se contarán una vez, independientemente del número de veces que sean requeridas, ya que el contenido que ya ha sido cargado no se solicita otra vez, sino que se conserva la referencia a la primera vez que se descargó.

La expresión que define la complejidad estructural es la suma de todos los nodos del grafo que generan las dependencias (véase ecuación 9).

Ecuación 9. **Métrica de complejidad estructural**

$$\text{Complejidad estructural} = \sum \text{nodos}$$

3.8. Estabilidad

Uno de los requisitos que tiene el consumo de datos de terceros es la disponibilidad de estos. Muchos servicios aseguran que sus API están disponibles durante un porcentaje de tiempo. Este compromiso es conocido como «acuerdo a nivel de servicio» (*service level agreement* [SLA]). Con el SLA, el proveedor de información se compromete a que su servicio cumple con un nivel de calidad mínimo.

De acuerdo a la evolución de la web donde el consumo de datos de terceros se realiza de manera constante, es importante conocer cuánto tiempo permanece una API disponible. Muchas empresas proporcionan métodos para conocer si un servicio no está operativo. Incluso existen páginas en internet que se dedican a comunicar incidencias en estas API. Por lo tanto, con esta métrica se intenta analizar la disponibilidad de las API que consume un componente. Es importante conocer estos valores, puesto que muchos son envolturas de estos servicios y, si algunos de ellos no se encuentran disponibles, el comportamiento del mismo no será el esperado, llegando a comportarse de manera errática.

El valor de esta métrica evoluciona a lo largo del tiempo, ya que es una de las variables que se tienen en cuenta para su medición. La expresión de esta métrica se puede expresar como se indica en la ecuación 10.

Ecuación 10. **Métrica de estabilidad**

$$\text{Estabilidad} = \frac{\sum_{i=0}^{n-1} S_i}{n}$$

Donde:

- n = Número de API que consulta un componente.
- S_i = Porcentaje de disponibilidad de un servicio.

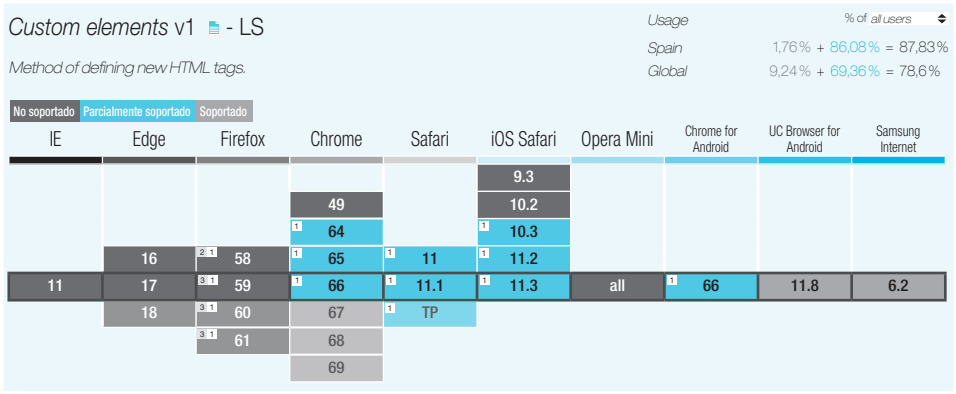
3.9. Portabilidad

La diversidad de navegadores y plataformas desde las que se accede a contenido web es muy variada. Cada plataforma tiene unas características determinadas y cada navegador tiene diferentes implementaciones de ciertos interfaces o incluso carecen de ellos. Esta heterogeneidad hace que el comportamiento de un componente varíe en función de la plataforma o del navegador donde se esté ejecutando. En algunos casos se dejará de tener el comportamiento deseado o incluso puede dejar de mostrarse.

En el ámbito de los componentes suele ocurrir con frecuencia, ya que las tecnologías que se están utilizando para su funcionamiento son muy recientes, y muchos de los navegadores solo dan soporte a estas funcionalidades en sus versiones más modernas o no se platean dar soporte a estas tecnologías a corto plazo (véase figura 4).

La diversidad de navegadores y plataformas desde las que se accede a contenido web es muy variada. Esta heterogeneidad hace que el comportamiento de un componente varíe en función de la plataforma o del navegador

Figura 4. **Soporte de los navegadores a *custom elements***



Fuente: Deveria y Schoors (2019).

Con esta métrica se pretende conocer si un determinado componente tiene algún problema en alguno de los navegadores más usados por los usuarios. Los errores proceden de las *issues* abiertas por los desarrolladores dentro de la plataforma de Github y que se encuentran sin resolver. Gracias al uso de repositorios colaborativos es sencillo realizar un seguimiento de estos problemas, ya que toda la comunidad participa.

Al igual que en la métrica de complejidad estructural, los errores que se analizan no solo son del propio componente, sino que también se analizan los errores de los componentes de los que depende. La fórmula que describe esta métrica es la que se puede ver en la ecuación 11.

Ecuación 11. **Métrica de portabilidad**

$$\text{Portabilidad} = \sum C_e$$

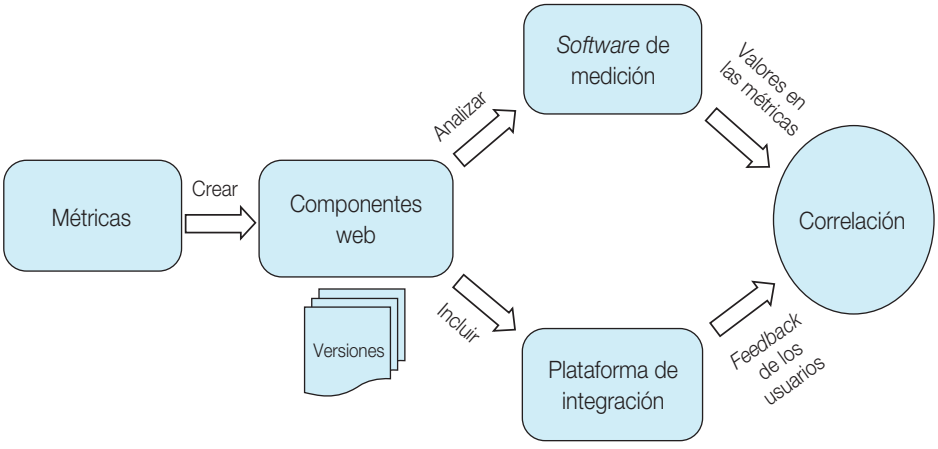
Donde:

- C_e = Cada uno de los componentes utilizados que se conocen con errores.

4. Experimentación

Para poder llevar a cabo la curación de las métricas propuestas mediante la opinión de los usuarios, se ha diseñado una prueba con usuarios finales siguiendo el método científico tradicional (véase figura 5).

Figura 5. **Esquema de metodología aplicada**



Fuente: elaboración propia.

La prueba se ha ideado para poder obtener información a través de la plataforma desarrollada. Junto a ella se han diseñado diferentes componentes web, basados en redes sociales, de los que se han creado múltiples versiones, tantas como métricas se han analizado. Cada una de ellas corresponde a un deterioro controlado en la calidad medida mediante *software* automático creado previamente, permitiendo así conocer si el usuario es capaz de detectar el empeoramiento. Así, por ejemplo, en el caso de la latencia, se ha incrementado el tiempo de respuesta en dos segundos por cada petición.

La prueba se ha ideado para poder obtener información a través de la plataforma desarrollada. Junto a ella se han diseñado diferentes componentes web, basados en redes sociales, de los que se han creado múltiples versiones, tantas como métricas se han analizado

La plataforma y el conjunto de componentes desarrollados nos permiten crear un entorno donde los usuarios pueden interactuar libremente con cada uno de ellos, evitando posibles ruidos entre el comportamiento de la plataforma y la funcionalidad de los componentes.

La plataforma y el conjunto de componentes desarrollados nos permiten crear un entorno donde los usuarios pueden interactuar libremente con cada uno de ellos

El conjunto de los componentes en sus diferentes versiones forma un escenario que se presenta a cada usuario a través de la plataforma. Este escenario es la base que se utiliza para obtener la información necesaria para entender qué aspectos de la calidad analizada tienen más impacto en la calidad percibida por los usuarios. Para cubrir todos los aspectos y distribuir correctamente las versiones se ha utilizado un sistema de distribución basado en el análisis de valores límite (*boundary value analysis* [BVA]) (véase figura 12).

Ecuación 12. **Representación de los diferentes escenarios**

$$S_i = \{C_j^0, C_j^1, C_j^2, C_j^3, C_j^4, C_j^5, C_j^6, C_j^7, C_j^8\}$$

Donde:

- S_i = Escenario determinado.
- C_j^k = Cada uno de los componentes que forman un escenario, donde *k* indica el nombre del componente y *j* la versión que se quiere implementar para cada uno. Cada *j* es independiente para cada componente.

Las versiones pueden dividirse en dos tipos:

- Uno en el que se encuentra la versión estable, que es la versión sin fallos y donde el funcionamiento del componente es correcto.
- Otro grupo es el resto de versiones que tienen un defecto en algunas de las métricas que se quieren tratar.

Realizada esta división, el BVA nos permite ofrecer los escenarios de manera que se prueben ambos extremos del sistema, en los que podremos encontrar unos escenarios donde todos los componentes son buenos y otros donde todos los componentes son malos. Una vez servidos ambos extremos, se van alternando entre un límite y otro, reduciendo la cantidad de componentes buenos, en el caso de que todos vayan bien, y los componentes malos, en el caso de que todos sean componentes defectuosos. Esta manera de operar continúa hasta que ambos se crucen por completo, donde se vuelve a iterar en el sentido opuesto.

Una vez definido el entorno y los escenarios que cada usuario utilizará, se les proporciona un guion con un pequeña introducción donde se explica cuál es el objetivo de la prueba, para qué se emplean los resultados en los que están participando, se describe la prueba que van a llevar a cabo (especificando las diferentes tareas) y se incluye información adicional sobre preguntas frecuentes y sobre los términos y las condiciones en los que se utilizarán los datos recolectados.

Algunos de los componentes desarrollados requieren acceso a diferentes redes sociales para poder obtener la información que se va a mostrar. Aunque se trata de redes sociales comunes y ampliamente extendidas en la sociedad, se puede dar el caso de que alguno de los usuarios no disponga de una cuenta en esa red social o que simplemente no quiera utilizar su cuenta personal.

En ambos casos se han facilitado un conjunto de cuentas de prueba con sus respectivas contraseñas para que el usuario pueda interactuar con todos los componentes sin ningún problema. En el guion también se les proporciona a los usuarios un pequeño apartado en el que se les enseña a borrar los permisos otorgados a la plataforma, ya que únicamente son necesarios durante la prueba y no se usarán para estudios posteriores.

Cuando los usuarios acceden a la plataforma, se les indica que vayan añadiendo componentes y jugando con ellos. Una vez que han interaccionado lo que consideren necesario, se les proporciona el cuestionario. Deben rellenar uno por cada componente. Este será utilizado para contrastar los resultados obtenidos por las herramientas de medición de la calidad y la opinión ofrecida por los usuarios.

El conjunto de preguntas está basado en una puntación dentro de una escala likert de cinco valores: «completamente en desacuerdo» o «muy malo»; «en desacuerdo» o «malo»; «ni de acuerdo ni en desacuerdo» o «en la media»; «de acuerdo» o «bueno»; y «completamente de acuerdo» o «muy bueno». Cada uno de estos valores se corresponde con un valor numérico que va desde 1, para «completamente en desacuerdo», hasta 5, para «completamente de acuerdo». Se ha elegido este tipo de escala porque es preferido por los usuarios respecto a un sistema basado en una escala ordinal (Reips y Funke, 2008). Además, se han incluido dos preguntas binarias y dos preguntas de respuesta abierta.

La prueba ha sido programada para llevarse a cabo sin la interacción del personal del equipo, por lo que han de ser los propios usuarios quienes irán descubriendo el funcionamiento de la plataforma y contestando a las diferentes preguntas sobre los componentes que hay que usar. A pesar de no ser supervisado, un miembro del proyecto estará presente para solventar los problemas que los usuarios no sean capaces de resolver tras un periodo de tiempo (véase cuadro 2).

Cuadro 2. **Encuesta realizada a los usuarios**

¿Qué nota le darías al componente?				
Muy malo	Malo	En la media	Bueno	Muy bueno
○	○	○	○	○
El componente es difícil de usar.				
Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
○	○	○	○	○
El diseño del componente me parece agradable/apropiado.				
Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
○	○	○	○	○

►

▶

La información del componente se muestra de manera concisa y clara.				
Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
○	○	○	○	○

Usaría el componente.

Sí ○ No ○

Recomendaría el componente.

Sí ○ No ○

Indique una ventaja del componente.

.....

Indique una desventaja del componente.

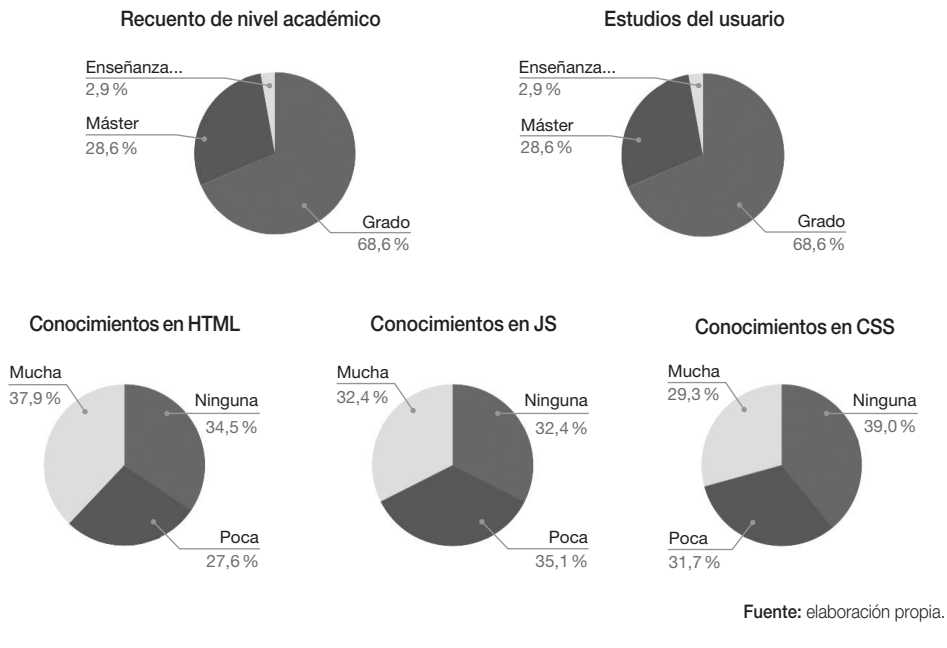
.....

5. Resultados

El estudio se ha realizado en momentos diferentes y en los que han participado un total de 35 personas. Cada uno ha tenido que valorar un total de 9 componentes de diferente índole y de los que se disponen 8 versiones distintas.

Han participado tanto estudiantes como desarrolladores de componentes con un conocimiento mínimo en informática. Al tratarse de un estudio no supervisado, algunos de los usuarios que han realizado la prueba no llegaron a completar todas las encuestas con los componentes. Esto no afecta a los resultados obtenidos, ya que el objetivo es disponer de tantas opiniones como sea posible y que al menos estas cubran todas las versiones que se disponen, independientemente de si se dispone o no de todas las versiones de todos los componentes.

Figura 6. **Descripción de la muestra de usuarios**



Fuente: elaboración propia.

Partiendo de los datos obtenidos a través de las encuestas se ha realizado un procesamiento de los datos, analizando los valores y estructurándolos de manera adecuada para poder conseguir los mejores resultados posibles.

De las encuestas con los usuarios finales se han obtenido 300 opiniones sobre los diferentes componentes. Cada opinión consta de 8 preguntas. Teniendo en cuenta la naturaleza de las preguntas, solo se han empleado aquellas que determinan la satisfacción de uso de un componente en una escala de 1 a 5. El resto de preguntas se han descartado porque, o bien son respuestas binarias de «sí» o «no», o bien son preguntas que requieren de un procesamiento del lenguaje natural y quedan fuera del objetivo de este estudio.

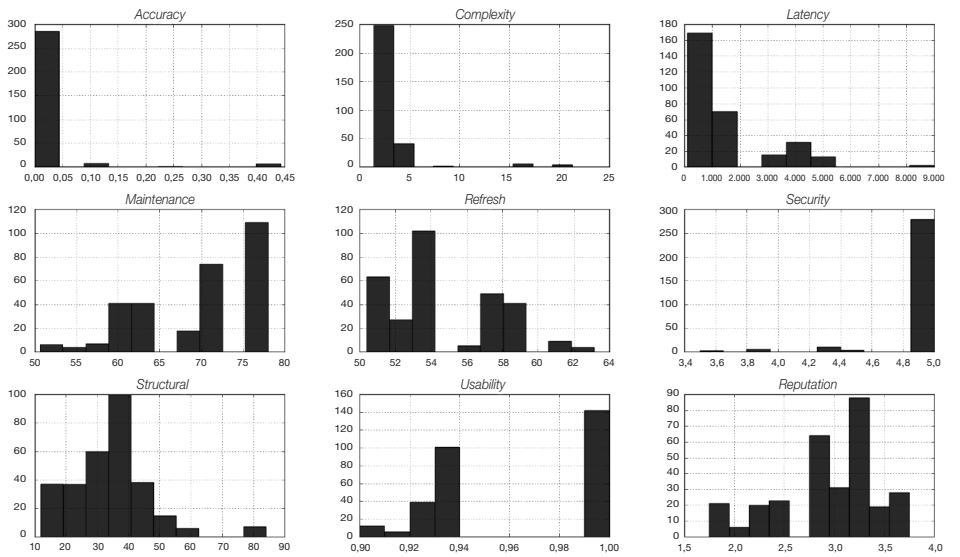
Estas preguntas han servido para construir un atributo derivado al que hemos llamado «*reputation*» y que consiste en la media de todos los valores. Una de las preguntas se ha realizado de manera negativa, es decir, a mayor nota, peor, por lo que se ha utilizado su valor inverso, donde una valoración de 1 (totalmente en desacuerdo) equivale a un 5, un 2 (en desacuerdo) equivale a un 4, y así sucesivamente.

Cada una de estas opiniones está asociada a un componente y a una versión determinada que corresponde al componente usado por el usuario en la plataforma. El componente

en una versión tiene asociado unos valores de las métricas desarrolladas. Estos valores son los utilizados y están asociados al valor de la calidad que han otorgado a ese componente en esa versión, y que componen los datos finales usados.

En total han utilizado 9 atributos diferentes, donde 8 de ellos son los empleados como descriptores del componente correspondiente a las métricas y el otro corresponde a la variable objetivo que el sistema intenta calcular, correspondiente a la calidad otorgada (véase figura 7).

Figura 7. **Histograma de los atributos de los datos**



Fuente: elaboración propia.

Cuadro 3. **Extracto de los resultados obtenidos**

Componente	Valoración versión estable (sobre 5)	Versión modificada	Valoración modificada (sobre 5)
Open-Weather	3,05	Latencia	1,44
Reddit-Timeline	3,43	Usabilidad	2,5

Componente	Valoración versión estable (sobre 5)	Versión modificada	Valoración modificada (sobre 5)
Traffic-Incidents	2,92	Seguridad	2,00
Google Plus-Timeline	3,22	Mantenibilidad	3,58
Spotify-Component	3,53	Complejidad	2,92
Finance-Search	3,05	Completitud	2,17
Facebook-Wall	3,41	Complejidad estructural	3,58

Fuente: elaboración propia.

En el cuadro 3 se ven algunos de los resultados obtenidos gracias a la opinión de los usuarios. A través de estos datos podemos observar que el usuario es capaz de detectar cambios en un componente, lo que afecta directamente en la calidad que perciben. En el caso del componente Open-Weather y su comparación entre la versión sin fallos y la versión con defectos en la latencia, vemos que el usuario penaliza de manera clara la puntuación que le otorga, pasando de un 3,05 sobre 5 a un 1,44. De estos se puede deducir que la latencia tiene un gran impacto en la calidad y que es uno de los factores de éxito.

> La latencia tiene un gran impacto en la calidad y es uno de los factores de éxito

De igual forma, la usabilidad, la seguridad o la complejidad son penalizados por los usuarios cuando se presenta un empeoramiento en estos aspectos. Atendiendo a los estándares tradicionales observamos que los usuarios coinciden, y ambos determinan estos factores como aspectos de calidad.

Frente a estos factores, vemos que el empeoramiento de la complejidad estructural o la mantenibilidad no es percibida por los usuarios o no se considera que afecte de manera negativa a la calidad. En ambos la valoración otorgada por los usuarios ha sido ligeramente superior. Esta peculiaridad se debe a la naturaleza de estas métricas, ya que son relativas a cualidades intrínsecas relacionadas con el proceso de desarrollo, de mejora y de mantenimiento de los componentes. Aunque no se percibe en el uso de los componentes, sí

> El empeoramiento de la complejidad estructural o la mantenibilidad no es percibida por los usuarios

se debe considerar uno de los factores de éxito de los componentes, puesto que el uso de los componentes en entornos de desarrollo creados por terceros requiere un alto nivel de mantenibilidad y una baja complejidad estructural.

6. Conclusiones

En la búsqueda de una web que sea completamente desarrollada utilizando componentes web, se adolece de la ausencia de un modelo que sea capaz de establecer las pautas que determinen la calidad del contenido creado. Los modelos tradicionales no se adaptan adecuadamente. Si bien pueden servir como punto de partida, en ningún momento tienen en cuenta la opinión de los usuarios finales, quienes son el último y más importante eslabón dentro de la cadena de la web.

Los resultados obtenidos indican que existe la posibilidad de crear mecanismos automatizados para la medición de la calidad e implementar modelos que, a través de los resultados obtenidos, sean capaces de prever el impacto que tendrá un componente en un usuario final antes de incluirlo en los catálogos de los repositorios colaborativos.

Los resultados indican que existe la posibilidad de crear mecanismos automatizados para la medición de la calidad e implementar modelos que sean capaces de prever el impacto que tendrá un componente en un usuario final antes de incluirlo en los catálogos de los repositorios colaborativos

Además, si se es capaz de entender qué métricas son más importantes para los usuarios, también existe la posibilidad de conocer qué estrategias y técnicas de desarrollo permiten incrementar la calidad percibida, ayudando a los desarrolladores a entender qué parte de su componente no causa una buena impresión a los usuarios a los que va destinado.

También existe la posibilidad de conocer qué estrategias y técnicas de desarrollo permiten incrementar la calidad percibida

La implementación de estas métricas, junto a modelos predictivos, facilita la reducción de la distancia técnica que existe entre el usuario final y el desarrollador de los componentes. A su vez, estos componentes facilitan la abstracción de esa complejidad, pero no son capaces de establecer diferenciación entre la calidad de los componentes destinados al mismo propósito. Mediante la cuantificación de la calidad de los componentes, utilizando un modelo predictivo de las métricas curadas por los usuarios finales, se puede reducir esta problemática, ayudando a reducir esa distancia entre desarrolladores y usuarios finales.

A pesar de las labores realizadas para reducirlo, aún existe otro problema sin abordar. Si se quiere facilitar que sean los propios usuarios los que desarrollen sus páginas web, se ne-

cesitan plataformas que permitan la integración de todas las partes necesarias para construirlas. Aunque existen soluciones más o menos buenas, no son capaces de abordar los problemas que existen a la hora de realizar la comunicación entre componentes.

Si se quiere facilitar que sean los propios usuarios los que desarrollen sus páginas web, se necesitan plataformas que permitan la integración de todas las partes necesarias para construirlas

Hasta el momento, el intercambio de información entre dos componentes ha sido realizado explícitamente mediante programas que se encarguen de esa labor. Por ese motivo sigue existiendo un impedimento sobre el hecho de que sean los usuarios los que lleguen a construir sus propias aplicaciones, ya que en todos los *mashup* debe existir una comunicación entre todas las partes que la componen y, actualmente, no existe una abstracción de alto nivel que elimine esos detalles de implementación.

7. Discusión

El experimento presenta varias limitaciones debido a la definición del mismo. En primer lugar, la muestra es relativamente pequeña, ya que solo se dispone de 35 usuarios. Aunque sí son suficientes para valorar las versiones de los componentes al menos una vez, ampliando significativamente la muestra se puede obtener un mejor consenso en la valoración de cada uno de los componentes en cada una de las versiones. De esta manera se podrían obtener unos valores más fieles a la opinión de múltiples usuarios.

Otra de las limitaciones de la experimentación nace de la propia carencia de mecanismos que reduzcan en mayor medida la cantidad de conocimientos necesarios para desarrollar entornos basados en componentes. Toda la experimentación se ha centrado en usuarios con una base mínima en informática, por lo que se les presupone un conocimiento mínimo en el desarrollo del *software*. Esta característica se debe a que se quiere asegurar que los usuarios disponen del criterio suficiente para discernir problemas en el desarrollo.

La experimentación utilizando usuarios no técnicos supone un riesgo en la calidad que pueden percibir, ya que no tienen por qué conocer los criterios para determinar si presentan problemas que no son usuales. Para experimentaciones futuras con muestreos de usuarios más grandes, se platea el cambio de la experimentación basada en el uso continuado de la plataforma, donde a los usuarios se les ofertan los componentes variando la versión a medida que van entrando a lo largo de los días. El objetivo de este tipo de experimentación es la obtención de resultados basándonos en un test A/B, realizando una aproximación a la calidad percibida en función de la comparativa de todas las versiones.

Referencias bibliográficas

Bevan, N. (1997). ISO 9126. *EAGLES Evaluation Group Workshop. Evaluation in Natural Language Engineering: Standards and Sharing*, November 26th and 27th. Brussels.

Bray, M., Brune, K., Fisher, D. A., Foreman, J. y Gerken, M. (1997). *C4 Software Technology Reference Guide-A Prototype.* Software Engineering Institute.

Bongais, J (2017). *ISO 9126*. Recuperado de <https://medium.com/@dinjheyn/iso-9126-77b35a02b646> (consultado el 1 de mayo de 2018).

Calero, C., Ruiz, J. y Piattini, M. (July 2004). A web metrics survey using WQM. *Web Engineering*, *3.140*, 147-160. doi: 10.1007/978-3-540-27834-4_19.

Calero, C., Ruiz, J. y Piattini, M. (2005). Classifying web metrics using the web quality model. *Online Information Review*, *29*(3), 227-248.doi:10.1108/14684520510607560.

Cappiello, C., Daniel, F. y Matera, M. (2009). A quality model for mashup components. *Web Engineering*, *5.648*, 236-250. doi: 10.1007/978-3-642-02818-2_19.

Chávez Rojas, A (2009). *Cubo de las dimensiones del WQM*. Recuperado de <https://www.researchgate.net/figure/Figura-11-Modelo-WQM-Web-Quality-Model-PQM-Tiene-como-objetivo-definir-un-modelo-de_fig1_265964918> (consultado el 15 de abril de 2019).

Coleman, D., Ash, D., Lowther, B. y Oman, P. (1994). Using metrics to evaluate software system maintainability. *Computer*, *27*(8), 44-49. doi: 10.1109/2.303623.

Deveria, A. y Schoors, L. (2019). Custom elements (v1). *Can I use…?* Recuperado de <https://caniuse.com/#search=custom 20ele 20ments 20v1> (consultado el 3 de mayo de 2018).

Gilbertson, D. (2018). I'm harvesting credit card numbers and passwords from your site. *Hackernoon*. Recuperado de <https://hackernoon.com/im-harvesting-credit-card-numbers-and-passwords-from-your-site-here-s-how-9a8cb347c5b5> (consultado el 3 de mayo de 2018).

Gill, G. K. y Kemerer, C. F. (1991). *Cyclomatic complexity density and software maintenance productivity*. IEEE.

Patel, N. (2018). *How loading time affect your bottom line*. Recuperado de <https://blog.kissmetrics.com/loading-time/?wide=1> (consultado el 26 de abril de 2018).

Hardt, D. (2012). The OAuth 2.0 authorization framework. *Internet Engineering Task Force*, *1-76*.

ISO (2019). *ISO/IEC* 25010. Recuperado de <http://iso25000.com/index.php/en/iso-25000-standards/iso-25010> (consultado el 5 de abril de 2018).

ISO (2011). 25010:2011. *Software engineering-Software product quality requirements and evaluation (SQuaRE). System and Software Quality Models*.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, *SE2*(4), 308-320. doi: 10.1109/TSE.1976.233837.

Niessink, F. (2002). *Software requirements: functional and non-functional software requirements*.

Padayachee, I., Kotze, P. y Merwe, A. van der. (2010). ISO 9126 external systems quality characteristics, sub-characteristics and domain specific criteria for evaluating e-Learning systems. *The Southern African Computer Lecturers' Association 2010.*

Reips, U. D. y Funke, F. (2008). Interval-level measurement with visual analogue scales in internet-based research: VAS generator. *Behavior Research Methods*, *40*(3), 699-704. doi: 10.3758/BRM.40.3.699.

Singh, R. (1996). International standard ISO/IEC 12207 software life cycle processes. *Software Process: Improvement and Practice*, *2*(1), 35-50. doi: 10.1002/(SICI)1099-1670(199603)2:1<35::AID-SPIP29>3.0.CO;2- 3.