Development of a Quality Metrics Definition, Evaluation and Quantification Framework for EUD Web Components

David Lizcano
School of Computer Science
UDIMA
Madrid, Spain
david.lizcano@udima.es

Andrés Leonardo Martínez
Google Software Department
Madrid, Spain
aleonar@gmail.com

Sandra Gómez, Ana Isabel Lopera, Miguel Ortega, Luis Ruiz, Juan Francisco Salamanca, Genoveva López
Conwet Lab
UPM
Madrid, Spain
{sgomez, alopera, mortega, lruiz, jfsalamanca, glopez}@conwet.com

***Abstract*—Web components technology improves Internet applications development. Although still at the experimental stage, there is a growing interest in its quality metrics. We aim to define a reference and evaluation framework for measuring the quality of web components and mashups. This paper presents the pilot phase of a real experimentation environment for comparing reference metrics built from existing software quality metrics with curated metrics based on user-perceived quality. The preliminary results of the evaluation of the alpha version conducted by the developers who participated in platform design and development speak for the suitability of the selected approach.**

Keywords-quality metrics; web components; end-user programming.

I. INTRODUCTION

Web components are programmable HTML tags built using a compendium of open technologies. Web components are elements independent of external libraries that are built into web browsers and are able to encapsulate HTML, JavaScript and CSS in reusable functional modules. They improve web development componentization, improving quality and productivity. Although still at the experimental stage, they are being implemented using technologies like Polymer or Bosonic. Alongside technology development, there is a growing interest in quality metrics. This is not currently a hot topic, however, as highlighted by the articles related to web components [1].

We aim to define a reference and evaluation framework for measuring the quality of web components and mashups composed by interconnecting several components. This paper presents the pilot phase of a real experimentation environment for comparing reference metrics built from existing software quality metrics with mature metrics based on user-perceived quality.

The absence of a universally accepted formal framework that can be applied to determine the quality of web components has led to the adaptation of traditional standards. However, some trial standards have been launched. For example, there is the Gold Standard Checklist [2], which is modeled on the W3C checklist for Web Content Accessibility Guidelines (WCAG) [3], or the idea of establishing quality control as the main point for quality in web components [4]. Neither standard provided a sound groundwork for our approach. Therefore, we decided to devise a new framework.

This approach has resulted in the definition of a set of metrics for assessing quality based on real user experiences. For this purpose, we developed an online platform which provides a social network hub. This platform displays different versions of components for experimental groups composed of real users and gauges perceived quality for comparison against the traditional models.

The user platform operates like a testbench where end users interact with the web components under evaluation in order to gather the key events associated with the use of these elements. This provides a black-box view of the component, and the analysis focuses on the functions evaluated by an end user when he or she uses the user platform.

The main functions implemented in the experimentation platform enable users to log in with OpenID, define groups, aggregate information from different social networks and follow the posts by other members as a group. The components considered in this study consume data from several social networks, including Twitter, Facebook and LinkedIn.

From the conducted evaluation (of the illustrated in Figure 1 and Figure 2 or similar components), we found that most users had problems moving components. On this ground, this is one of the aspects to be improved.

Section 2 describes the state of the art of web components. Section 3 includes the technical details for developing the evaluation platform. Section 4 explains how the metrics elicited from users will be validated. Section 5 reports the preliminary evaluation of the platform, followed by the conclusions of the paper.

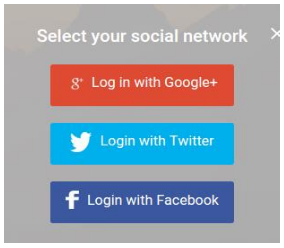
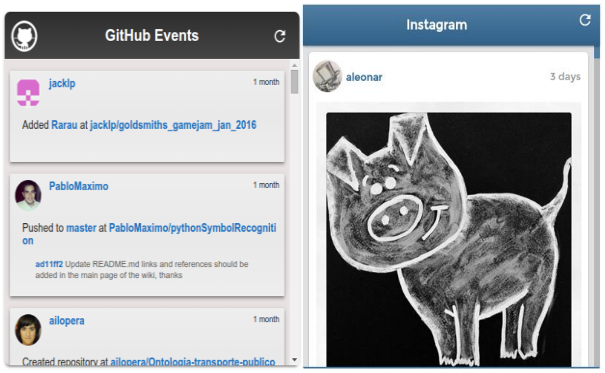


Figure 1. List of login components



jacklp 1 month

Added Rarau at jacklp/goldsmiths_gamejam_jan_2016

PabloMaximo 1 month

Pushed to master at PabloMaximo/pythonSymbolRecogniti on

ad11ff2 Update README.md links and references should be added in the main page of the wiki, thanks

ailopera 1 month

Created repository at ailopera/Ontologia-transporte-publico

Figure 2. Examples of timeline components (Github and Instagram).

II. THEORETICAL FRAMEWORK

Web components refer to technologies for creating new HTML tags or components using conventional web development languages (HTML, Javascript and CSS) [5]. These new elements are easy to reuse and can embed all the component implementation details, which renders them transparent to the document in which they are used. Web components are a way of modularizing web elements, creating more complex tags to extend the tools available for building web applications or mashups.

There are four key enabling technologies, based on new standards defined by W3C [6]:

- Shadow DOM: enables the definition of a new subtree of document object model (DOM) elements separate from document rendering.
- Template: enables inert elements, which can be later activated, to be inserted into the document.
- Custom elements: establishes how the user can create new tags and new interfaces.
- HTML imports: defines how to insert templates and custom elements into the document.

There are no codes of good practice or standards related to web component development. As a result, end users may have to deal with a very large unstructured catalogue of components, including many elements designed to serve the same purpose. In other cases, there may be components that constitute a potential source of security vulnerabilities or data loss. This is a problem for end users because they do not know a priori how to tell which component is the best for the job that they are doing and are unable to detect vulnerabilities. On this ground, there is a need for a system capable of establishing and quantifying the quality of a component.

Formal metrics are used to establish software quality. These metrics define which aspects measure the quality perceived by the users that consume the software. There are several rules defining software quality, which, however, all have two clearly distinct concepts in common: software structure quality and software functionality quality. Functional quality stresses software conformance with a design based on defined software specifications. On the other hand, structural quality addresses the analysis of the internal structure and non-functional requirements of the software, such as security and maintainability.

There are several quality assessment models. Most are based on the ISO 9126 quality standard. For many years, this was the international software quality assessment standard. ISO 9126 defines software quality as the combination of a number of characteristics that represent attributes whose quality can be measured and evaluated. Some of these attributes are functional adequacy (satisfaction of stated or implied needs), performance efficiency (level of performance of the software and the amount of resources used under stated conditions), compatibility (capability of two or more components to perform their functions when they share the same hardware or software environment), usability (component understandability, learnability, ease of use and attractiveness for users), reliability (capability of software to maintain its level of performance under stated conditions for a stated period of time), security (capability of data protection so that unauthorized people or systems cannot read or modify data), maintainability (capability of the component to be effectively and efficiently modified) and portability (capability of a component to be effectively and efficiently transferred from one hardware, software, operating or application environment to another) [7]. This standard has been replaced by ISO 25010 [8], including a reworked software product quality model.

This new ISO standard covers two more aspects than its predecessor: security and compatibility. Additionally, some subcharacteristics have been renamed or added. The ultimate aim of ISO 25010 is to highlight the importance of software quality of use for users.

Although these formal models describe software quality, they do not cover all the facets of web component quality, as they neglect the user. The usability quality attribute does indicate that user needs to understand the component, but makes no mention of the fact that the target user’s opinion is equally important, because component quality depends on whether or not it is used. There are not many web component and mashup quality model proposals that focus on web usability research. These models attribute the quality of the mashups and their respective components to their functional characteristics and their usability, such as service quality [9]. Although other models define metrics for establishing quality like SOA-based quality assessment [10], they address code aspects or in-development assessments, but do not take into account the user. There is also a mashup-specific model [11]. In no case, however, do they take into account external component attributes such as the availability of documentation about operation, social impact or data quality.

On this ground, the ConWet Laboratory DEUS work group, based at the School of Computer Engineering, Technical University of Madrid, has set up a portal in which the users can interact freely with social network web components. We have created different component versions, each with different characteristics. Users will interact with these versions at random to assess and rate the quality of the components. We will also collect user interaction data in order to discover how users interact with components and thus adapt the components to their way of thinking.

However, the focus of the approach is not entirely new, as there have been solutions that have focused on user-driven component interconnection. Two such approaches are Yahoo! Pipes [12] and Wirecloud [13]. Yahoo! Pipes is a solution for filtering the content of one or more queries in order to translate their content or answer the question. The deployed interface is rather complex for end users (which are the target audience of this platform). On this ground, we believe that ours is a better approach. On the other hand, WireCloud resembles our approach more closely, insofar as this solution also operates on individual elements that can be connected with each other. It has an easier to use interface than the Yahoo! Pipes. Even so, it has several buttons that do not clearly specify the functionality that they represent. However, element interconnection is highly automatic, as this platform does not work with web components like the ones used in this solution.

Additionally, there were other solutions aimed at creating web pages by interconnecting components (no web pages were created in the above examples). These solutions were based on end-user web site development. Some of these solutions were: Marmite [14], QED Wiki [15], PopFly [16] and JackBe Presto [17]. Marmite is a tool operating on the Firefox browser enabling users to gather information by searching the Internet. To do this, users had several operators (sources, filters, processors and sink) that they could use to gather the above information. On the other hand, QED Wiki is a mashup builder developed by IBM that was based on the Wiki concept. This solution simplified much of the technology side, like editing, commenting or publishing. Apart from web pages, users were able to quickly develop prototypes using this tool. On the other hand, PopFly was a solution developed by Microsoft whose main goal was to enable users to create their own web portal by adding content (like images or text), as well as adding a title and page profile. Content could be added to other spaces, like Facebook, or blogs, like WordPress. Finally, the Jackbe Presto tool provides users with a choice of filters and connections in order to visualize the collected data and build custom applications. Note that some of these solutions are no longer in use, as only the tools that achieve some level of maturity have managed to survive and remain active.

III. DEVELOPMENT

In this section, we explain the underlying architecture of the platform, detailing the technologies used on each side (client and server) and the implementation of the formal metrics considered in the early phase of the development.

A. User *environment architecture*

The user environment is organized as a client-server architecture, with separate technologies for each side. The languages used on the client side, which is divided into different modules, are JavaScript, HTML and CSS, accompanied by technologies like AngularJS or Polymer. Polymer is used exclusively to create web components, whereas AngularJS is used, among other things, to integrate the components into the portal.

As shown in Figure 3, the client and server side are split into different modules. The client side is divided into four modules, each of which pursues different goals to the others. The first is the client interface, which takes care of defining what users see and how they interact with the portal. The second is responsible for connecting with the server side in order to send and receive data, which also offers components depending on the data that it receives. The third module is responsible for interconnecting components that are part of a user profile. Using this module we can take measurements illustrating how a user interacts with the components. Finally, the fourth module collects the measurements gathered from user interaction with the dashboard.

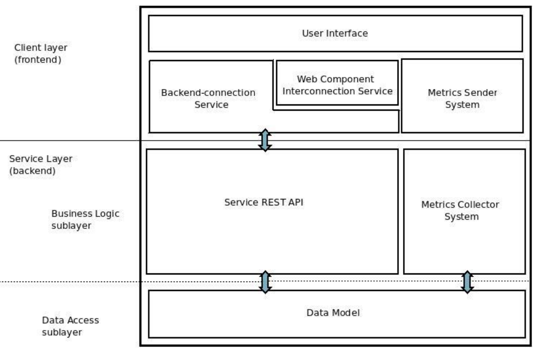


Figure 3. User environment architecture (Picbit)

The server side, on the other hand, is divided into two layers. One layer addresses the application logic and the other is responsible for data persistence and storage. The first layer is divided into two modules, one of which defines a REST API to enable the client to access defined resources of different types, including user type, component or credential. We have defined a different API (application programming interface) for each resource, and another to support auxiliary operations. The other module is responsible for processing the metrics of the different components: it fetches the events generated on the client side, calculates the metric values and assigns the respective value to each component.

On the other hand, the data sublayer is composed of a single module that takes care of the persistence of the generated data. To do this, we used the NDB (non-relational database) Python API to define the entities required to store this information. We opted for a non-relational model, defining different entities to store information related to the data managed by the application.

We chose a non-relational model in preference to a relational model because of the way in which the data were to be generated. Relational models are perfectly well-suited to applications storing ordered data. For example, this model is ideal for projects where user data are to be stored. However, a non-relational model which is better at processing continuous query reception and should be used if the information is generated more continuously and it is not so important whether or not the information is ordered. Although we need to store user information in this case too, the priority is to store all the information received as a result of user interaction with the portal. Therefore we opted for a non-relational model.

B. *Technology selected for* the *user environment*

Figure 4 and Figure 5 show the distribution of these technologies on the client and server sides, respectively. Although some of these technologies are used to connect modules, they are not illustrated below as we are concerned with the module technologies.

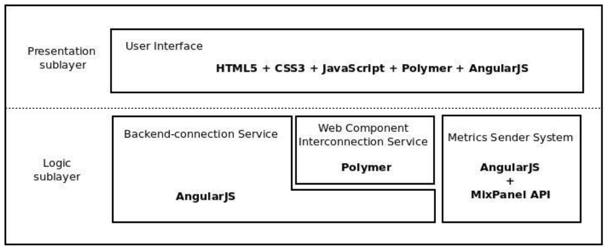

Presentation sublayer
User Interface
HTML5 + CSS3 + JavaScript + Polymer + AngularJS
Logic sublayer
Backend-connection Service
AngularJS
Web Component Interconnection Service
Polymer
Metrics Sender System
AngularJS + MixPanel API

Figure 4. Client-side technology diagram

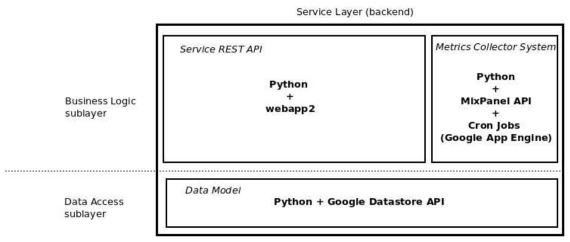


Figure 5. Server-side technology diagram

There are now a wide range of technologies available for creating a user interaction portal. On this ground, we had to select the ones best suited to the target objectives. The selected technologies are detailed in the following.

1) Polymer

Web component development framework promoted by Google which implements W3C-defined standards. This technology is used as a library capable of managing the implementation, reuse and injection of web components. Web components are inserted into a new HTML document as if they were a new tag that is defined in the respective language.

Polymer is capable of implementing simple components, like a button, and even creating elements that may constitute a proper application.

2) Angular JS

Framework implementing the model-view-controller pattern (MVC), which is capable of creating dynamic web applications, aimed at increasing the availability of frontend development tools, and simplifies and eliminates the web application code. In our case, it is used to integrate different web components into the portal.

3) App Engine

The Google App Engine is used as the platform as a service (PaaS) to deploy the portal, exploiting its manageability and maintenance features. Some specialized services built into the platform, like NDB for information management and storage and Memcache for temporary data storage in cache memory, are also used. NDB is just a storage service offering an API for operating on the Google App Engine Datastore, whereas Memcache is a service that operates like a cache memory for storing some data that need to be saved during the user session.

4) Mixpanel

MixPanel satisfies the need for a service capable of monitoring user-portal interaction behaviour and collecting component interaction data. MixPanel stores these data for later retrieval using an API.

The collected data are used to see if changes to the latency, completeness and usability metrics affect user-perceived component quality. To do this, different sentences (mixpanel.track (name_event, [properties_event])) [18] are included on the client side that sends the data to the service (the module has to be have been loaded as specified in the reference).

5) Bower

The user platform is responsible for managing which dashboard version is served to the user. The Bower framework is used to manage the dependencies of a particular dashboard. The components belonging to a dashboard version are contained in a specified bower file, loaded from the client side. The platform server side is responsible for specifying the components that are part of the dashboard for the client side.

C. *Determination of end user-based metrics*

User-application interaction data are useful for calculating quality metrics for the components that they are using. The first thing to do in order to assess component quality is to look at which metrics are best suited for the stated goals and which user actions the platform will offer.

Firstly, we decided to use a small number of metrics to get a rough idea of the quality of the components. These metrics are calculated internally by the component and do not require user interaction.

We had to decide which of the set of metrics studied for the SOA architecture and mashup to include in the first version of platform. Due to the complexity of the metrics presented in Section VII, we decided to look for metrics that were easier to define, leaving the adoption of the metrics specified in the related work section for a more advanced stage.

The metrics considered under these circumstances are as follows:

- **Completeness**: aims to measure the accuracy of the output component data. It takes into account a set number of messages, gathered first from the social network server and then from the component, and checks that the messages received from both sources are equal.

$$GeneratedOutputData \div SearchedOutputData$$

The metric value is generated by assigning values to the different accuracy levels shared by both sources. Assignment is nonlinear, that is, 70% completeness is not equivalent to a metric value of 7, that is, a reasonable weighting system has to be defined to try to understand how missing data affect component quality.

- **Latency**: is defined as the time that it takes to execute the component from the time when the query is sent to the server until the component displays the data on screen.
- **Data refresh time**: aims to determine the time that it takes the component to refresh the information when there is system data input. For example, how long does it take for a component to visualize the new information from a tweet published at a specified time? Different automatic refresh times are tested to find out which is the best accepted by the users. The refresh time is set by measuring the difference between the time at which the message is displayed by the component and the time at which the message is received by the server.
- **Usability**: evaluates user interaction aspects, covering most aspects of usage. As this is such a broad dimension, theoretical usability is first evaluated based on the checklist published and approved by W3C. The procedure is to rate the component against the checklist items to assign the first rating. This aspect will later be rated more directly through site rating. In this manner, the theoretical usability can be compared against real usability.

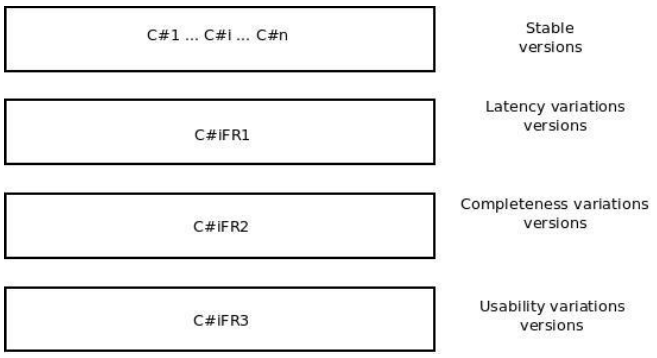
The assumption is that the metrics are established by comparing data output by the social network server and by our components. Accordingly, a series of conditions should be agreed with the social network provider by means of service level agreements (SLA), specifying a service between the above service provider and service users.

At this stage of the research we have analysed mature web components developed by our work group. Mature components are subject to continuous development, as feedback is received from the values recorded for the dimensions measured during the research (completeness, latency and usability).

The analysed components play the role of specific social network consumers. We developed several versions of each individual component including slight variations (as shown in Figure 6). Each version introduces a change that will have an impact on one of the dimensions to be measured in order to determine the impact of this variation on the overall quality of the component. Accordingly, there are four versions of each component:

- Stable version of the component, with high metric values.
- Version including changes to latency dimension.
- Version including changes to data completeness dimension.
- Version including changes to usability dimension.

The different versions described above are used to compose different dashboards for one and the same user and evaluate the user experience in each case. For each user, there are four different dashboard variations, and each variation includes components from a specified version.


C#1 ... C#i ... C#n
Stable versions
C#iFR1
Latency variations versions
C#iFR2
Completeness variations versions
C#iFR3
Usability variations versions

C#: Version of a given component
FR: Final Release
Figure 6. List of component versions

As a result of user interaction with the components in their dashboard, MixPanel fetches the events generated on the client side, acting as an analytical platform. Each event is associated with a particular dimension.

A distinction is made with respect to the dashboard from which this event is fetched in order to calculate the metrics for each dashboard. Each event type is included in the calculation of one of the four defined metrics, which are also divided into two different groups of metrics:

- **Inter-user metrics.** Measured on the interaction events between all the platform users.
- **Intra-user metrics.** Measured on the interaction events associated with a particular user.

IV. VALIDATION

Once the platform has been tested internally, users can start to interact with the components in order to collect data from real users. The aim of this process of validation is to find out what impression the components make on real users who are given the chance to rate these components. This will output metric values assigned by users and the internally output values will be able to be compared with the ratings based on end-user interaction with the platform.

A. Data collection on user interaction and metric calculation

User interaction is monitored to ascertain how users perform with the different components built into the portal.

TABLE I: CHARACTERIZATION OF USERS

Characterization	***Group 1***
Gender	
Male	15
Female	0
Age	
20-30 years	13
Over 30 years	2
Educational attainment	
Secondary School	5
Vocational Training	1
Bachelor's Degree	5
Master's Degree	4
Employment	
Student	5
Employee	2
Both	8
Experience and previous knowledge	
Python	15
JavaScript	14
HTML	12
CSS	14

Users interact with mature components. The users of the beta versions advise on which aspects of the components could be improved. New component versions will then be created that behave differently with respect to different aspects. Only one characteristic of each component will be changed, such as refresh time or an intentional data input error to alter a metric. These components are assigned a default rating greater than they warrant based on their real quality.

When users log in to the portal, they are presented with a random version of the components, and their interaction is monitored. They will then be able to rate their user experience. The user ratings should gradually correct the component quality rating until it stabilizes at a more realistic value.

During user interaction with the portal, data, such as the time spent using each component, portal tab closure or how long the user was logged in to the portal for, are sent to MixPanel. These data are later collected in order to calculate the respective metrics. The main purpose of these data is to validate user outcomes, for example, by not storing a rating by a user that has only interacted with the component for one second, as it would be unreliable.

Data is collected by means of a daily server task which fetches and stores the data from MixPanel. When these data are available, another task is launched to recalculate the metrics and update the respective values.

B. Analysing data normality (normal use conditions)

Normal data refers to data that are repeated over a long period of time. For example, the collection of training data can take up to a week. The data collected over this time are useful for establishing a baseline that we will take to be the normal behavior. By establishing this baseline of normality, we can assure that the use conditions of both the modified and standard component versions are as similar as possible. Accordingly, it is more feasible to draw conclusions about the behavior of the components from the user viewpoint.

V. EVALUATION

The first component evaluation was conducted in a very controlled environment with a very definite user profile. This profile matches users aged from 20 to 30 years with programming experience. The evaluation was held on the development work group premises. A total of 15 users were assembled for 10 minutes (the profile of the users can be viewed in Table I). They were given some brief instructions and asked to interact with the platform and components to complete a number of tasks. This test was conducted as a litmus test. However, we intend to run tests with other user profiles before releasing a stable version, as we believe that this platform has the potential to be a real solution for users and not just a mere test box.

The purpose of this study is to establish a correlation between the defined target metrics (completeness, latency, data refresh time and usability) and user opinion in order to determine the success of the metrics and measure web component quality from two perspectives. The metrics are a formalization of aspects that are considered to have an impact on component quality and have to be compared with user-perceived quality. This determines how sensitive users are to a deviation from the metric baseline values.

The experiment lasted no more than 15 minutes and was divided into several parts. During the first part, users were given a brief description of the purpose of the survey and of the platform, as well as some very basic instructions to follow. During the second part, the user performed the tasks and a team member made observations. During the third part, the users completed the survey addressing their opinion of platform use and some personal and professional data in order to put together a profile of the users that participated in the experiment. Two members of the development team were with the user at all times. One team member answered any questions that the user had about interacting with the interface and the other took notes on the actions that the user performed to complete the set tasks.

Below, in Figure 7, is a photo of the interaction process.

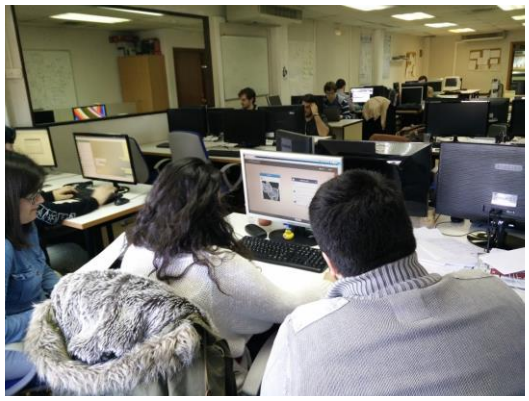


Figure 7. Interaction process.

After receiving instructions, the user started to interact with the platform and completed a series of tasks (log into the system using the network of choice, add two components to the work environment, move one of the added components, delete the unchanged component, log out and log in again using the network of choice). This did not take them longer than five minutes. As they performed these operations, some users made comments that were taken down by the experiment observer. These comments will be discussed later along with the results and opinions of the users that took the survey. After interaction, the users completed a survey on their user experience. This survey is available at [19]. Based on the above interaction and survey, we were able to infer a number of quantitative and qualitative findings, as well as gather the impressions that the platform made on users.

The results of the survey and the values selected during the user interaction will be analyzed by the work group in order to change the aspects that users found hardest to use. The main goal of this study is to improve the platform for alpha testing involving a larger number of people in order to prevent misunderstandings of its features. User comments after platform use are very important for this purpose.

In order to assure that user characteristics did not bias the study, we conducted an ANCOVA. The analysis showed that user characteristics had no impact on the analysed features. Thus, there is no statistical evidence of the results being biased by the users who took part in the evaluation. In any case, more studies will be executed with a higher and more heterogeneous population in order to completely rule out the possibility of the results being biased.

First, let us detail how the users expressed their opinion. We were able to gather user opinions in different ways. First we analysed the comments that the users made while interacting with the platform. These are usually comments suggesting improvements or pinpointing aspects of the platform that are not absolutely intuitive. These comments were taken down by the experiment observer. Second we analysed the opinions that the users expressed in the surveys taken after interaction with the platform. These opinions were mostly consistent with what users had mentioned as they performed the tasks. All these comments and opinions are discussed below.

The observer took note of the users' first impressions of platform use, possible improvements or any aspects that they did not find altogether intuitive. Additionally, we recorded whether or not the user performed the task. We found that actions related to user dashboard modification are the hardest for the users to complete. The dashboard-related tasks with the highest error rate on the part of the users were add and modify dashboard components with an error rate of 60% and 80%, respectively. Exceptionally, we had to help some users out, in one case to add and in two cases to modify components. The interaction of these actions needs to be redesigned in future platform versions for the purpose of improving interface usability. After observing user behaviour, our conclusion is that the best option in this case is to enable users to move components around the workspace using the drag and drop feature. The dashboard management tool also requires simplification.

The users did not have much difficulty with the system login and logout actions, and 60% used the same social network in the first and second logins that they were asked to perform in the experiment. Thus, we conclude that users understand the concept of creating a platform profile using one of their social networks.

The opinions reported by users in the surveys often matched what they had said while completing the tasks. The survey was composed of short-response and multiple-choice questions. The question statements were neutral in order to prevent response bias. Short-response questions were used to elicit user ratings of particular aspects of the user experience or check whether they remembered the steps required to complete a particular task. The multiple-choice questions ascertain the level of agreement/disagreement with different items on a scale from 1 (strongly disagree) to 5 (strongly agree). The survey results are shown in Figure 8.

The subjective ratings of users with respect to the platform and the components were positive. Of the comments received, it is noteworthy that around 53% of users positively rated the workspace simplicity in terms of interaction and design, and none of the users rejected the idea of using the platform daily.

As regards improvement suggestions, all users recommended a change in component management and suggested associating contextual menus with components or redesigning their associated gestures. They also advised increasing the salience the platform's help section in order to assist any users that have trouble performing any of the possible actions.

Generally, the components made a good impression on users. They highlighted the fact that they were well designed and covered an acceptable range of social networks. However, as the study primarily targeted platform management, further studies will be required to gather more conclusive results in this respect.

On the whole, the study revealed aspects of the interaction that would need to be redesigned and provided a preliminary picture of what users think about the concept and target functionality of the platform. Based on the ratings, we can say that the idea of linking the publications of several social networks on a single page will be grounds enough to attract users to our platform and thus be able to gather information from the designed metrics. Nevertheless, some of its features needed to be improved to make it more intuitive and easier to use.

Figure 9 and Figure 10 show snapshots of platform screens used in the testing, and some screens that were displayed to users. Generally speaking, the results of the survey shown in Figure 8 encourage us to forge ahead with platform development, taking into consideration the survey findings and increasing the platform functionality.

VI. RELATED WORK

For decision making on which aspects were to be measured for the usability metric, we searched the literature for papers on quality assessment for similar applications and

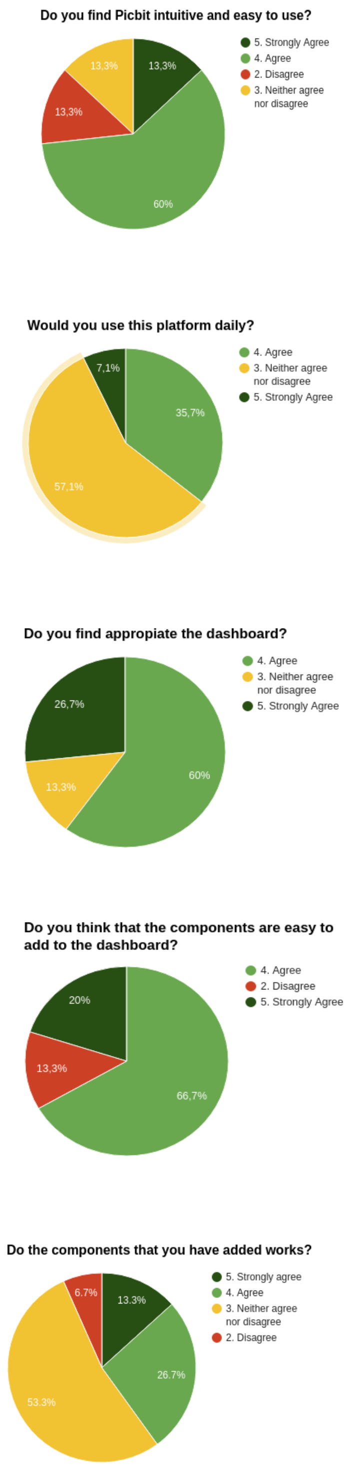


Figure 8. Results of the usability survey

specifically references related to service oriented architecture (SOA) [10] and mashups [11].

After reviewing these papers [10] [11], the only proposal that matched what we were looking for was an article that designed a quality model for a SOA application. As we were unable to extract results from these papers, we found it very hard to compare our approach with the solutions presented in the referenced papers [10] [11]. The tables below show how their authors measured the metrics for this model.

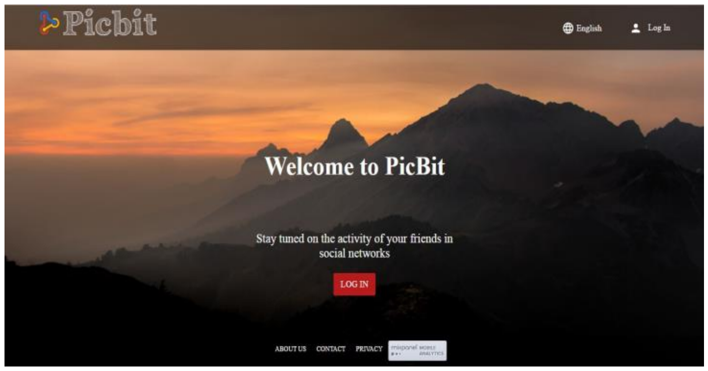
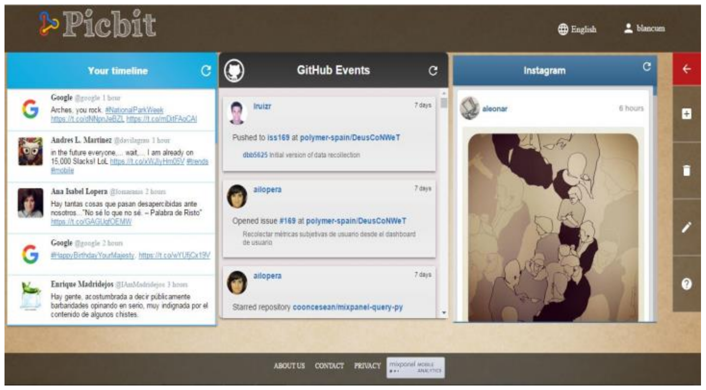

Picbit
English
Log In
Welcome to PicBit
Stay tuned on the activity of your friends in social networks
LOG IN
ABOUT US
CONTACT
PRIVACY
mixpanel MOBILE ANALYTICS

Figure 9. PicBit Landscape



Google @google 1 hour
Arches, you rock. #NationalParkWeek https://t.co/dNNpnJeBZL https://t.co/mDitFAoCAI

Andres L. Martinez @davilagrau 1 hour
in the future everyone,... wait,... I am already on 15,000 Slacks! LoL https://t.co/xWJlyHm05V #trends #mobile

Ana Isabel Lopera @lomaranas 2 hours
Hay tantas cosas que pasan desapercibidas ante nosotros..."No sé lo que no sé. – Palabra de Risto" https://t.co/GAGUgfOEMW

Google @google 2 hours
#HappyBirthdayYourMajesty. https://t.co/wYUfjCx19V

Enrique Madridejos @IAmMadridejos 3 hours
Hay gente, acostumbrada a decir públicamente barbaridades opinando en serio, muy indignada por el contenido de algunos chistes.

lruizr 7 days
Pushed to iss169 at polymer-spain/DeusCoNWeT
dbb5625 Initial version of data recollection

ailopera 7 days
Opened issue #169 at polymer-spain/DeusCoNWeT
Recolectar métricas subjetivas de usuario desde el dashboard de usuario

ailopera 7 days
Starred repository cooncesean/mixpanel-query-py

aleonar 6 hours

Figure 10. PicBit with some added components.

TABLE II: INTERNAL METRICS

Description	***Internal metric***
Number of Operations	SIM_NO
Number of Fine-Grained Parameter Operations	SIM_NFPO
Number of Message Used	SIM_NMU
Number of Asynchronous Operations	SIM_NAO
Number of Synchronous Operations	SIM_NSO
Number of Inadequately Named Operations	SIM_NINO

TABLE III. EXTERNAL METRICS

Description	***External metric***
Number of Consumers in Same Level	SEM_NCSL
Number of Directly Connected Producer Services	SEM_NDPS
Number of Directly Connected Consumer Services	SEM_NDCS
Total Number of Producer Services	SEM_NTPS
Total Number of Consumer Services	SEM_NTCS

TABLE IV. SYSTEM METRICS.

Description	***System metric***
System Size in Number of Services	SM_SSNS
Number of Inadequately Named Services	SM_NINS
Number of Inadequately Named Operations	SM_NINO
Total Number of Messages Used	SM_TMU
Number of Asynchronous Operations	SM_NAO
Number of Synchronous Operations	SM_NSO
Number of Fine-Grained Parameter Operations	SM_NFPO
Number of Process Services	SM_NPS
Number of Intermediary Services	SM_NIS
Number of Basic Services	SM_NBS

The tables (Table II, Table III, Table IV, Table V and Table VI) below show the proposed metrics for analyzing the quality of a service oriented architecture (SOA).

Table II shows internal service metrics, which can be defined in the service code. These metrics can be calculated by means of static code review.

Table III addresses data that depend on user execution, as well as the number of simultaneous consumers.

Table IV refers to all the data that can be gathered from the system as a whole, taking into account defined operations, interactions and services.

Table V shows how the values of the metrics defined to assess the quality of the application are calculated. They use

TABLE V. DERIVED METRICS

Derived Metric	***Description***
Average Number of Directly Connected Services (DM_ADCS)	(SEM_NDPS + SEM_NDCS) / SM_SSNS
Inverse of Average Number of Used Messages (SM_IAUM)	SM_SSNS / SM_TMU
Number of Operations (DM_NO)	SM_NSO + SM_NAO * 1.5
Number of Services (DM_NS)	SM_SSNS
Squared Avg. Number of Operations to Squared Avg. Number of Messages (DM_AOMR)	(SM_NAO + SM_NSO / SM_SSNS)^2 / (SM_TMU / SM_SSNS)^2
Coarse-Grained Parameter Ratio (DM_CPR)	(SM_NSO + SM_NAO - SM_NFPO) / (SM_NSO + SM_NAO)
Adequately Named Service and Operation Ratio (DM_ANSOR)	((SM_SSNS - SM_NINS) / SM_SSNS * 2) + (SM_NSO + SM_NAO - SM_NINO) / (SM_NSO + SM_NAO) * 2

TABLE VI. DESIGN PROPERTIES – METRICS RELATIONSHIPS

Derived metric	***Design Property***
Average Number of Directly Connected Services (DM_ADCS)	Coupling
Inverse Average Number of Used Messages (DM_IAUM)	Cohesion
Number of Operations (DM_NO)	Complexity
Number of Services (DM_NS)	Design size
Squared Avg. Number of Operations to Squared Avg. Number of Messages (DM_AOMR)	Service granularity
Coarse-Grained Parameter Ratio (DM_CPR)	Parameter granularity
Adequately Named Service and Operation Ratio (DM_ANSOR)	Consumability

internal, external and system metrics to determine the values for these aspects.

Finally, Table VI shows how the calculated values are related to the defined properties to assess the quality of the application.

None of the above metrics were included in the first version of the framework, but we believe that they all potentially have a role to play in our service. They are to be included later, as they require more complicated calculations than the metrics that we have adopted.

VII. CONCLUSIONS

Formal standards do not adequately cover web component quality as they focus on different software architectures and exclude user interaction as a basis of measurements. Standards like ISO 25010 include a wide range of metrics, which are far from easy to apply to web components.

After reviewing the state of the art, we have found that developed best practice guidelines or recommendations with respect to quality web components and web component mashups are still preliminary. Research by both more formal organizations like W3C and developer communities that have emerged around technologies like Polymer or Bosonic has focused to date on concept formalization and supporting technologies.

A platform that focuses on the interaction of user groups within social networks represents a real evaluation environment that reduces biases associated with artificial experimentation environments. The metrics of completeness, latency, data refresh time and usability are easily modified functionally, enabling the creation of multiple versions of the same web component.

Controlled exposure to real users yields user satisfaction metrics based on simple web analytics which can be analyzed through correlational studies. The preliminary results of the evaluation of the alpha version conducted by the developers who participated in platform design and development speak for the suitability of the selected approach.

In coming platform iterations, the platform will be released in an open Internet environment in order to corroborate the results reported in this paper in a broader context. This should test the hypothesis that formal component quality and user interaction metrics are correlated.

REFERENCES

[1] Articles. Web Components.org (Retrieved May 12th, 2016). Available at http://webcomponents.org/articles/

[2] The Gold Standard Checklist for Web Components. GitHub, Inc. (Retrieved May 12th, 2016). Available at https://github.com/webcomponents/gold-standard/wiki

[3] Web Content Accessibility Guidelines (WCAG) 2.0. W3C. (Retrieved May 12th, 2016). Available at https://www.w3.org/TR/WCAG20/

[4] Basic Quality Control Concepts. Philosophe. (Retrieved May 12th, 2016). Available at http://philosophe.com/testing/qc/

[5] Overson, J. & Strimpel, J. (2015). Developing web components. Sebastopol: O'Reilly.

[6] Main page. Web Components.org. (Retrieved April 4th, 2016). Available at http://webcomponents.org/

[7] Norma de Calidad ISO/IEC 25010. Iso25000.org (Retrieved April 4th, 2016). Available at http://iso25000.com/index.php/normas-iso-25000/iso-25010

[8] International Organization for Standarization. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models. ISO 25010:2011 Ginebra: ISO, 2001, 25 p.

[9] A Model for Web Services Discovery with QoS. S. Ran. ACM SIGecom Exchanges. Volume 4, Issue 1. Spring 2003. Pages 1 - 10.

[10] A Design Quality Model for Service-Oriented Architecture. B. Shim, S. Choue, S. Kim, S. Park. Software Engineering Conference, 2008. APSEC '08. 15th Asia - Pacific. Pages 403 - 410.

[11] A Quality Model for Mashups. C. Cappiello, F. Daniel, A. Koschmider, M. Matera, M. Picozzi. Web Engineering. 11th International Conference, ICWE 2011. Pages 137 - 151.

[12] Yahoo! Inc. About pipes (Retrieved April 12th, 2016). Available at http://real.pipes.yahoo.com/pipes/

[13] ConNWeT Lab (UPM). Welcome to WireCloud! (Retrieved April 12th, 2016). Available at http://conwet.fi.upm.es/wirecloud/

[14] Marmite: Towards End-User Programming for the Web. J. Wong. 2007 IEEE Symposium on Visual Languages and Human-Centric Computing. Pages 270 - 271.

[15] Service Oriented Architecture - SOA. QEDWiki: Putting a Web 2.0 face on SOA. IBM, Inc (Retrieved April 19th, 2016). Available at http://www-01.ibm.com/software/solutions/soa/newsletter/jan07/article_QEDwiki.html

[16] A. Bradley, D. Gootzit. Who's Who in Enterprise 'Mashup' Technologies. In Gartner Research, ID G00151351, 7th September 2007 (Retrieved April 20th, 2016). Available at: http://liquidbriefing.com/pub/Harmonia/IndustryAnalysts/whos_who_in_enterprise_mashu_151351.pdf

[17] JackBe's Presto: A Self-Service, On-DemandData Integration, Mashup Based, Dashboard-Oriented, Business Intelligence Tool. Beye NETWORK, a TechTarget company (Retrieved April 19th, 2016). Available at http://www.b-eye-network.com/view/15018

[18] Tutorial: Tracking your first event. Mixpanel (Retrieved April 5th, 2016). Available at https://mixpanel.com/help/reference/tracking-an-event

[19] Google, Inc. Encuestas Usabilidad (Retrieved April 4th, 2016). Available at https://docs.google.com/forms/d/1s0js0h3KoxcWNamlWQr9Wzblw-BIT1gCraI_e_g3Rdc/viewform