Requirements Eng (2018) 23:33–61
https://doi.org/10.1007/s00766-016-0257-3

ORIGINAL ARTICLE

CrowdMock: an approach for defining and evolving web augmentation requirements

Diego Firmenich[1,2] · Sergio Firmenich[1,3] · José Matías Rivero[1,3] · Leandro Antonelli[1] · Gustavo Rossi[1,3]

Received: 12 March 2015 / Accepted: 30 May 2016 / Published online: 11 June 2016

Abstract Web Applications are accessed by millions of users with different needs, goals, concerns, or preferences. Several well-known Web Applications provide personalized features, e.g., they recommend specific content to users by contemplating individual characteristics or requirements. However, since most Web Application cannot consider all users' requirements, many developers started to create their own mechanisms for adapting existing applications. One of the most popular techniques for third-party applications adaptation is Web Augmentation, which is based on the alteration of its original user interface, generally by using scripts running at the client side (e.g., the browser). In the context of Web Augmentation, two user roles have emerged: *scripters* who are those users able to create a new augmentation artifact, and *end users* without programming skills, that just consume the artifacts that may satisfy totally or partially their needs. Scripters and end users generally do not know each other, and they have rarely a contact, beyond the fact that they use the same script repositories. When end users cannot get their needs covered with existing artifacts, they claim for new ones by specifying their requirements (called Web Augmentation requirements) using textual descriptions, which are usually hard to interpret by scripters. Web Augmentation requirements are a very particular kind of Web requirements for which there partially exist a solution implemented by the Web site owner, but still users need to *change* or *augment* that implementation with very specific purposes that they desire to be available in such site. In this paper, we propose an approach for defining and evolving Web Augmentation requirements using rich visual prototypes and textual descriptions that can be automatically mapped onto running software artifacts. We present a tool implemented to support this approach, and we show an evaluation of both the approach and the tool.

Keywords Requirements engineering · Web engineering · Web augmentation

✉ Sergio Firmenich
sergio.firmenich@lifia.info.unlp.edu.ar

Diego Firmenich
dfirmenich@tw.unp.edu.ar

José Matías Rivero
mrivero@lifia.info.unlp.edu.ar

Leandro Antonelli
lanto@lifia.info.unlp.edu.ar

Gustavo Rossi
gustavo@lifia.info.unlp.edu.ar

[1] Lifia, Facultad de Informática, UNLP, La Plata, Argentina

[2] DIT, Facultad de Ingeniería, Universidad Nacional de la Patagonia San Juan Bosco, Trelew, Argentina

[3] CONICET, Buenos Aires, Argentina

1 Introduction

Personalizing a Web Application consists in delivering specialized (in general adapted) contents to different users, considering their needs, goals, or preferences [6]. Experience has shown that devising a personalization mechanism that satisfies the requirements of all potential users is really challenging, particularly when the crowd of users is constantly growing. To make matters worse, users have evolved while using the Web and they have, day by day, more sophisticated and, in general, personalized requirements.

One of the answers for this evolution has been the development of techniques allowing users to adapt third-

party applications by themselves. Several communities working on these techniques have then emerged. One of the most popular approaches to achieve third-party applications adaptation is Web Augmentation [5, 10, 44]. Basically, Web Augmentation consists in adding, altering, or removing features of the application's user interface (i.e., the resources that the application's server delivers to the client after a request), by using software artifacts (usually scripts) that run on the user device, normally in the Web browser. Web Augmentation artifacts are used to adapt content, functionality, and presentation of existing Web sites. In the last years, the crowd of users has been able to satisfy many of their needs by themselves by creating and sharing Web Augmentation artifacts. There is much evidence showing the success of this technique as a way for adapting Web sites, from public scripts repositories (where artifacts for adapting very popular Web Applications, such as YouTube and Amazon, are installed thousands of times), to very domain-specific uses such as chemistry and biology Web sites [41].

Let's consider a simple example. Imagine a PhD student, called Peter, who is navigating DBLP[1] looking for some papers. When he finds a paper of his interest, he adds manually a new entry in his Mendeley Library.[2] In order to do that, he has to move papers' information from one application (DBLP) to another (Mendeley) by copying and pasting. He realizes that, by using a Web Augmentation artifact, he could improve his work by adding a link to each DBLP result that will open a new tab or window with the Mendeley page containing the form to add a new paper, automatically pre-filled with the DBLP paper information. However, it is not straightforward to develop such artifact to do that and he should acquire technical knowledge that is out of his interests. If he does not find an existing artifact for satisfying his requirement, he has two options. He can create a new one by using end-user tools (which are generally developed for very specific augmentations tasks and do not require programming skills) or he can ask for some of the existing communities to build the needed artifact. However, it is hard to communicate such requirement even for this single task. And it is harder in the context of Web requirements because users and developers are widely distributed. In our example, Peter cannot express orally his requirements, because he cannot meet the scripter, so he must write a specification of his requirements, and this is usually a problem. It is not difficult to imagine a more complex augmentation requirement or an evolution of this one such as to integrate Mendeley inside DBLP, to understand that soon Peter will realize that if he wants this adaptation, it is likely that he has to build it himself.

If the requirement is simple enough, Peter could be able to use WYSIWYG (*What You See Is What You Get*) tools such as Platypus [44] or WebMakeUp [13] for developing the artifact by its own without learning new programming languages. However, as we show later, it is hard to satisfy Peter's requirements with such tools since they do not even have the expressivity needed. In fact, several of the augmentation artifacts extensively used cannot be developed with this kind of tools.[3,4]

In this paper, we focus on the problem of building complex Web Augmentation software that is beyond end-user programming tools. When the desired augmentation is too complex and requires the use of low-level languages such as JavaScript, end-users' skills could not be enough to build their own augmentation artifacts. In cases like these, and Web Augmentation communities are an example, a communication channel is established between end users and scripters. In this way, end users ask for augmentation artifacts to scripters, which means that through this communication channel, end-user requirements are expressed. This kind of Web Augmentation requirements is becoming more relevant (as we explain in the following section) because it is actually a technique for personalizing and improving user experience when navigating the Web. Extracting requirements from Web Augmentation communities implies different issues to be tackled. These requirements are contextualized to a particular Web Application. Since this is an *augmentation requirement*, the implemented artifact would run over the UI of an existing Web Application, and consequently, most of the problem domain is already implemented. The augmentation layer just adds something missing or desired, that beyond of implying some programming logic, always adds, changes, or removes UI components. To make matters worse, this problem has been ignored (or at least under estimated), up to know, by the software engineering field, particularly by the requirements engineering community. Though the problem of managing requirements from a large crowd of users is not new, and it is a very challenging task [22], existing research focuses on more "institutional" software projects. However, our target end users do not belong or work in any specific company or institution and people solving their problems are, in general, volunteers. They tend to communicate informally, and the use of systematic approaches for the specification is absent. The possible kinds of augmentation functionality are usually far from trivial (See Sect. 3.3.2), and communicating requirements informally usually do not work.

[1] DBLP—http://www.informatik.uni-trier.de/~ley/db.

[2] Mendeley—http://www.mendeley.com/.

[3] https://openuserjs.org/scripts/YePpHa/YouTube_Center.

[4] https://openuserjs.org/scripts/JoeSimmons/YouTube_Auto_Buffer_Auto_HD.

The consequences are, as one might suppose, evident regarding satisfaction (many times requirements are not fulfilled), and when the underlying Web Application evolves or when requirements get old, there is no easy way to catch up again with the users' needs. Clearly, a systematic approach is needed. In a study we have done in this context, we found that among more than 4,500 artifacts from greasefork.org, 56.7 % of the artifacts have at least an upgrade (as a consequence of Web site upgrades and also for improving the functionality of the script). The 25 % of the upgraded artifacts have been written in the last 5 months. These statistics denote the dynamism such as communities and the need for a systematic way to allow end users to communicate their requirements to scripters.

In this sense, we believe that the way in which an augmentation requirement should be defined has to involve the definition of the *visual augmentation*. Additionally, we should note that different end users from the community could have similar requirements. We believe that a way to make easier requirement refinement and prioritization should be contemplated, since these are very common in software development, and current communities do not provide mechanisms for doing it. With all this in mind, it is clear that communicating and managing *augmentation requirements* is an important but yet unexplored topic.

We have been working in the field of Web Augmentation and Web Applications requirements for many years [25, 31, 32] and have identified the set of problems that need to be solved in this field. We have done this by carefully analyzing the requirements and products of hundreds of Web Augmentation artifacts in real Web Augmentation communities, and have come up with a set of recurrent type of augmentation requirements, achieving a taxonomy of Web Augmentation uses which is similar to the one described by a relevant survey work in the field [11].

Though our research proposes several contributions to the process of Web Augmentation software development, in this paper we focus specifically on the stage of requirements definition. We present a process for dealing with this kind of requirements, called CrowdMock. We also describe a specific tool, called MockPlug, for specifying augmentation requirements through augmentation-based mock-ups which are inspired by well-known mock-up tools [31]. We propose to complement the mock-ups with User Stories [8] and to deal with them collaboratively. Our aim was mainly to empower script requestors with mechanisms to communicate easily and accurately their needs.

The contribution to the specific field of requirements engineering is threefold: The first contribution is the identification of a new kind of problem that, so far, has been under estimated: the problem of requirements specification for augmentation software. The second is a process and a tool to support requirement specification and, in many cases, automatic generation of specified augmentation artifacts. Finally, and not less important, we present a way to synchronize and prioritize these requirements in an augmentation community. We think that our work is not only relevant but also seminal to the requirements engineering field.

The rest of the paper is organized as follows. In Sect. 2, we introduce the background of the area focusing mainly on Web Augmentation techniques and its most relevant communities. In Sect. 3, we present the core of our approach. In Sect. 4, we show the tools we have created for supporting our approach. In Sect. 5, we present the evaluation of our work. In Sect. 6, we survey the related works, focusing on similar processes, models, and tools. Finally, in Sect. 7, we conclude and comment some future work.

2 Background

As mentioned, a powerful mechanism for adapting third-party Web Applications is to use software artifacts running in the client side. This means that the adaptation is performed inside the Web browser by executing code that manipulates the loaded content. The resources that are available to be manipulated are mainly those composing the Web Application's UI (User Interface), such as HTML documents, CSS, JavaScript scripts, and images. Thus, Web Augmentation artifacts deal in general with the DOM (Document Object Model), which is an object representation of HTML documents and CSS (Cascading Style Sheets) which are a way for defining presentation of such documents. The most important Web Augmentation communities emerged around two kinds of augmentation artifacts:

- Userstyles A *userstyle* defines a set of CSS rules that replace the original ones of the Web site being augmented in order to change the presentation of its content. The most popular tool used to accomplish this is Stylish.[5] Stylish allows users to install userstyles (basically, CSS files). The most remarkable community around the idea of userstyles is userstyles.org, where users who may specify their desired CSS rules upload userstyles for their preferred Web sites, and other users may download and install them in their Web browsers. Nowadays, there are more than sixty thousand userstyles, most of them aimed to be applied over very well-known Web Applications such as Facebook, YouTube, Twitter, and Wikipedia. Some of the

[5] Stylish, https://addons.mozilla.org/es/firefox/addon/stylish/.

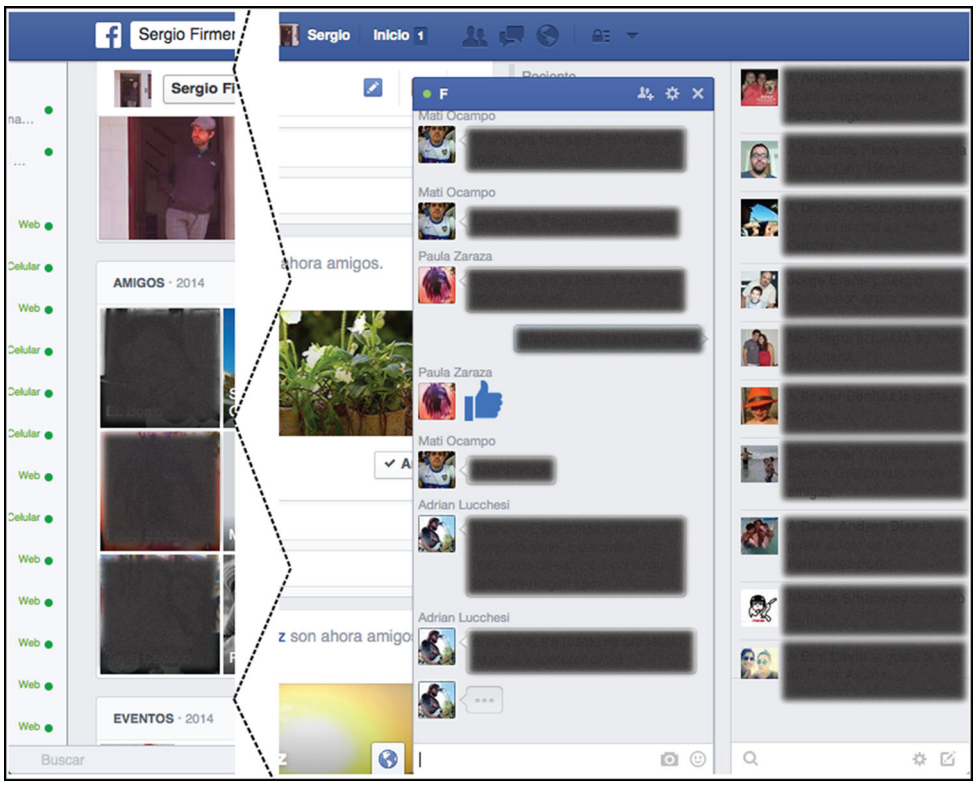

Fig. 1 UserStyle-based adaptation on Facebook

userstyles in the repository have more than forty thousand installations. Those users who do not know how to create new userstyle may ask changes on the existing ones (in a dedicated forum within the community) or also ask for a totally new userstyle in specific forums for style requests.[6] As an example, in Fig. 1 we present the result of applying a userstyle in Facebook. There are three main adaptations performed in the example. First, the chat contact list is moved to the left and also there is a semi-hidden panel that is totally shown when the mouse is over. Besides that, the chat window is larger, allowing users to see more messages in the conversation, which is very useful for group chats. Finally, as a consequence of moving the chat contact list at the left, the right panel is totally dedicated to show the last events. Others userstyles hide off-line people in the contacts list, etc.

- Userscripts deal with defining scripts to manipulate the DOM using JavaScript. With this kind of artifacts, it is also possible to add new features and eventually modify the functionality of the underlying Web Application—not only its look and feel as with *userstyles*. The first popular tool of this kind was GreaseMonkey,[7] a Firefox extension working as a JavaScript engine; it allows users to execute external scripts (developed by any user) when a particular Web page is loaded. Nowadays, there are equivalent engines compatible with other popular Web browsers like GreaseKit for Safari, Tampermonkey for Chrome, etc. It is important to stress that most of these extensions run the same type of JavaScript scripts, called *userscripts*. A userscript is an artifact containing both the JavaScript logic for manipulating the Web content and a set of metadata that indicates, for instance, which Web pages will be manipulated with it. There are several userscript repositories (GreasyFork,[8] UserScripts,[9] etc.) that offer, altogether, more than one hundred thousand userscripts. Some of these scripts were downloaded more than one million times. As well as with userstyles, users who do not know how to create new userscripts can review existing ones and also ask for a new userscript using forums like "Ideas and Script Requests."[10] As an example of the power of userscripts, let's consider the script called *YouTube center*. This userscript, whose last version was installed more than seventeen thousand times only from one of the available repositories, adds a lot of new functionality to YouTube video pages, such as further player configuration, audio options, layout

[6] https://forum.userstyles.org/categories/style-requests.

[7] https://addons.mozilla.org/en-us/firefox/addon/greasemonkey/.

[8] http://greasyfork.org/.

[9] http://userscripts-mirror.org/.

[10] http://userscripts-mirror.org/forums/2.html.

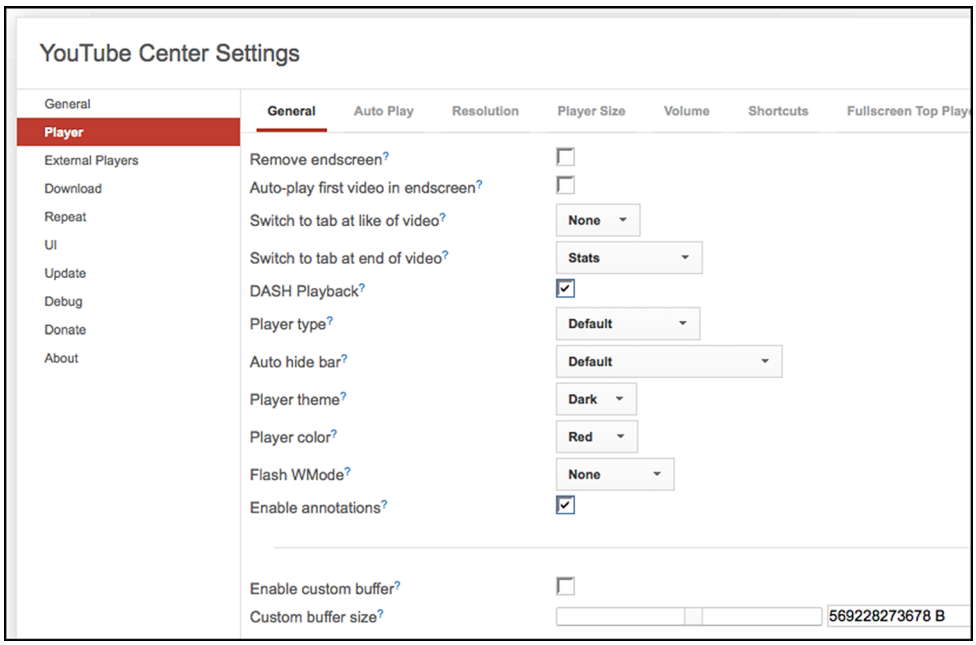

Fig. 2 UserScript-based adaptation on YouTube

options, UI enhancements, videos, and audio downloading. Each user may customize all these new extensions by a complete configuration menu shown in Fig. 2.

These communities, where thousands of Web Augmentation artifacts are shared with end users, are strongly based on scripters' programming knowledge. In this sense, end users are pushed into the background role of consuming existing artifacts. However, there are several research works published in the context of end-user programming, which has been defined as "programming to achieve the result of a program primarily for personal, rather public use. The important distinction here is that program itself is not primarily intended for use by a large number of users with varying needs" [21]. The Web is a very good platform for enabling end-user programming (tools) in order to adapt or integrate Web content. In fact, this has been happening for a long time, since several tools and approaches have emerged in order to allow users to specify changes in their preferred Web sites by means of visual tools. For instance, in the context of Web Augmentation, Platypus [44] was one of the first tools pursuing the claim "what you see is what you get" (WYSIWYG); it allows end users to specify the expected changes over the Web pages with visual tools and then to export a userscript materializing them. Although we later review some of these approaches in the related work section, it is important to mention that the *augmentation effect* achieved by this kind of tools does not reach the expressiveness and complexity of the more popular scripts available in public repositories.

3 The CrowdMock approach

CrowdMock is a user-driven approach for defining and evolving Web Augmentation requirements. In the approach, every stakeholder (as scripters or end users) may collaborate in the management of the requirements. However, we have oriented the process and the supporting artifacts and tools in order to take advantage of scripter's technical knowledge with the goal of providing fast and usable solution sketch before implementing the definitive version of the script. In this way, and considering Web Augmentation communities, where the contract between stakeholders (scripters and end users) is really weak, it is important to quickly provide a solution sketch in order to have very fast acceptance tests even before of building the final artifact. With this in mind, the way in which that requirements are specified should be really close to the expected solution and facilitate a rapid artifact development. In this section, we first present the general approach. Then, we review some important details about both products and activities. We also describe which type of augmentation requirements can be managed with CrowdMock, and finally, we show a case study illustrating the whole approach.

3.1 The CrowdMock process in a nutshell

In CrowdMock, different stakeholders (users and scripters interchangeably) involved in the system (the Web Application to be augmented) collaborate actively in specifying the augmentation requirements. The approach is based on

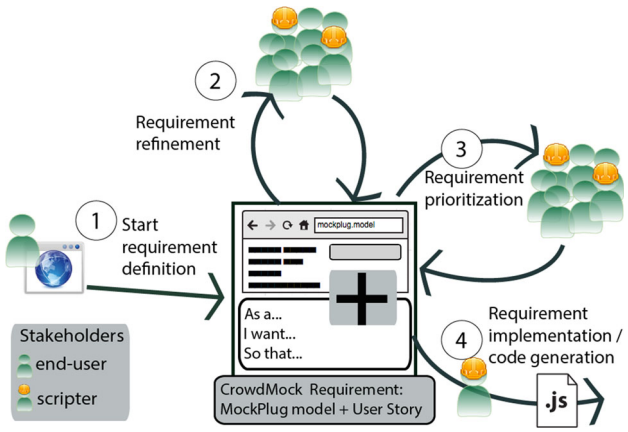

Fig. 3 Overview of the approach

the use of two kinds of existing artifacts: User Stories and UI mock-ups. User Stories are usually the requirement artifact used mainly in widely adopted agile development methodologies. At the same time, we took mock-ups (artifacts that are also extensively used in agile context) as a strategy to describe an augmentation requirement in the form of UI components that are woven into an existent Web Application; in this sense, our approach does not use *traditional* UI mock-ups, but it let users specify live, high-fidelity prototypes that are actually woven on the target Web page.

There are four activities defined in our approach that rely on the two basic requirements artifacts mentioned before: (1) requirements definition, (2) requirements refinement, (3) requirements prioritization, and (4) requirements implementation [9]. Figure 3 shows the activities and elements involved and their relationships. The requirements definition activity is related to producing the first requirement specification. A stakeholder writes a User Story and defines a first mock-up about a particular need; they both together specify a *CrowdMock requirement*. The requirements refinement activity deals with improving or enriching the requirements artifacts defined previously. This is the starting point of a collaborative process since one stakeholder can improve the requirements specified by another one; in this context, a traceability relationship between artifacts appears since a requirement is "refined by" another stakeholder producing a new enriched version. Requirements prioritization is mainly supported by a common social activity: voting. This activity allows the community to indicate, to those participants with programming skills, which are the requirement version most people desire to be implemented. However, other social activities, such as discussion forums about the requirement, are also contemplated. The order between activities 2 and 3 may vary since a requirement may be evolved at the same time that users are voting it.

Finally, activity (4)—requirement implementation—is supported by the automatic generation of code based on the MockPlug metamodel; though we briefly comment it in Sect. 4.4, a complete description of this activity, which is outside the of scope of this paper, can be found in [17].

Note that, although in Fig. 3 we just distinguished between scripters and end users, in following sections, we show how these two kinds of stakeholders have specific views and tools for the requirement specification.

We have developed a Web-based environment that supports all these activities. Our system is composed of a server-side and a client-side application. The server-side application, called UserRequirements, is a repository where the requirements are centralized and managed. The client-side application, called MockPlug, is a tool that empowers users with mechanisms for injecting mock-ups into existing Web Applications. All the tool support is available online,[11] and we describe the whole system in Sect. 4.

3.2 CrowdMock: products and activities

This section describes in detail the products and activities of the proposed approach. As commented before, our approach is based on two artifacts: User Stories and mock-ups.

A User Story is a description in natural language that captures what the user wants to achieve. User Stories are used within agile software development methodologies and generally adjust to a template that considers three attributes: a *role*, a *goal/desire*, and a *reason* [8]. The *goal/desire* represents the requirement that the application must fulfill. The *role* defines the kind of user who interacts with the application in order to use the feature described by the *goal/desire*. Both attributes refer to elements within the scope of the application. In contrast, the *reason* belongs to the context of the application, and it states why the user requires the application to provide the functionality described in *goal/desire*.

Mock-ups are a way of prototyping UI on paper or using digital computer images in order to agree on presentation requirements (among others) with end users. Commonly, a mid- or high-fidelity mock-up of a UI will look like the real UI, but it will not perform any function. It has been shown that when mock-ups are used in software development, the cost and effort of the development is reduced [30]. One of the main advantages of mock-ups in the development process is that they allow defining interaction and presentation aspects very early. This avoids future changes in the application being built because of mistakes or errors in the requirements gathering stage. Also, mock-ups can be used

[11] UserRequirements and MockPlug: http://www.userrequirements.org.

as a jargon-free language to communicate requirements between users and analysts or developers [27]. In our approach, a mock-up is represented by what we call a *MockPlug model*.

A MockPlug model contains the definition of alterations made over the original Web page's DOM in order to define a prototype of the expected augmentation. At first glance, a MockPlug model can be seen as a high-fidelity mock-up [39]. From the point of view of end users, actually it represents a prototype of the script they expect; however, from the point of view of scripters, while they refine a model they are also visually "programming" what the script is going to adapt or augment in the Web site's UI as well as specifying low-level aspects for the code generation. Different from the idea of using mock-ups to describe some visual and interactive aspects of the desired solution (using them only as a documentation artifact), we take a MockPlug model as the first, partially functional version of the script, i.e., such as a live script that is evolving. A refinement of the MockPlug model is, in this sense, a second version of the script. Because of this special reuse, we refer to our mock-ups as Runnable Augmentation Mock-ups (RAMs).

In our approach, an instance of the MockPlug metamodel, described in [17], is associated only with one User Story, but one User Story can be referenced or *split* in multiple MockPlug models. In comparison with other mock-up tools, the novelty of MockPlug models (the form in which we formalize RAMs) is that they are defined over existing Web Applications, showing a real augmentation. Additionally, they are not "just" an image but the aggregation of a set of (interface) objects with precise semantics both in terms of their own behavior and in terms of their effect in the target Web page. This feature makes the resulting requirements artifacts to be very close to the definition of the expected implementation and, as is briefly shown in Sect. 4.4, allows the incorporation of real functionality to fast acceptance test before the automatic generation of Web Augmentation code.

In our approach, User Stories have another role related to the dynamic of the Web Augmentation communities. The current repositories of scripts provide typically a few ways of search augmentations: to browse the repository by relevance, browse the repository by target Web site or just by searching by text on the script's descriptions. The decision of using User Stories for this purpose was motivated because they are the most used documentation method in the context of Agile Methodologies [24] which in place are the most adopted development approach actually according to recent surveys.[12] In the context of CrowdMock, they are used as a textual way of extracting the essence of the requirement being implemented which, complemented with the target Web site and the text provided on the RAM's widgets, give us a powerful way to retrieve relevant requirements. This includes the role of the user involved (for instance, logged user, anonymous one, and administrator), what is exactly this user wants to accomplish through the augmentation and optionally the reason because he or she requires it, following the classic User Story format *As a <type of user> , I want <some goal> so that <some reason> .*

Our approach consists of four activities; however, since this paper is mainly concerned with Web Augmentation requirements specification and management, the following explanation contemplates only three of them. The first one is requirements definition, which consists in specifying a requirement using two artifacts: User Stories and mock-ups (RAMs). The stakeholder writes a User Story description and he or she also defines a related RAM by using MockPlug. This activity implies writing for the first time a requirement (User Story and mock-up) that has never been specified before. In fact, the activity of specifying requirements implies two more basic activities of the classic requirements engineering approach: eliciting or defining the knowledge and specifying it. Since the same person is the one who own the knowledge about the requirement and specify it, our requirements definition activity is closer to the requirements specification activity in agile development than in a classical approach, since stakeholders have an important and irreplaceable role in the elicitation. The following activity consists in the refinement of one of the two artifacts composing a CrowdMock requirement, the User Story and/or the RAM. In both cases, the new version of the artifact is derived from the previous one, and then we are able to trace the requirement evolution.

Finally, the last activity is requirements prioritization. For this purpose, we propose to use voting, a simple social activity. In particular, we decided to use a "like" tool instead of another common activity such as ranks, because it showed to be more effective in filtering requirements [2]. Nevertheless, the requirements refinement activity also means in a certain way their prioritization, since this refinement implicitly means that stakeholders consider that this requirement is important enough to be refined. Prioritization is supported by Agile Methodologies when the backlog is organized in a stack with the most important User Stories well described at the top and the least important are barely described at the bottom.

3.3 Expressivity: what augmentation requirements can be defined with CrowdMock

First of all, it is necessary to understand which is the nature of Web Augmentation requirements, i.e., which kinds of

[12] 10th Annual State of Agile Survey—http://stateofagile.versionone.com/.

requirements are present in Web Augmentation communities. We have analyzed existing Web Augmentation artifacts in two ways: first by typifying the alterations over Web pages, and later by analyzing these alterations from a more traditional requirement engineering's point of view.

3.3.1 Analysis of Web Augmentation artifacts

If we take into account existing taxonomies that classify which kinds of alterations are usually carried on by Web Augmentations artifacts, we can find a coarse-grained classification in a recent survey [11]. This research work establishes that augmentation is currently used mainly to adapt Content, Layout, Navigation, and Functionality, through adding, removing, or changing elements in the augmented website.

However, in order to ensure that our approach supports a more fine-grained taxonomy, we have analyzed the top 50 userscripts from one of the most important repositories (greasyfork.org). We downloaded, installed, and used each of these scripts, and then we classified the adaptation effects and strategies which are not mutually exclusive (i.e., a script could include more than one), obtaining the following classification:

- Content-related, most of the analyzed scripts augmented Web pages through content manipulation, which can be subclassified in:
  - Filter/Hide content: 2/50 scripts filtered or hid content in some way. For instance, one of the scripts hides the results series episodes that have been visualized before.
  - Remove content: 11/50 scripts removed content, with the main objective of advertisement suppression.
  - Add content: 42/50 scripts added content from the same application, loading it from different pages. For instance, a script for YouTube added information to the search results, loading it from the video page of every individual search result item.
  - Integrate content from other Web Applications: 8/50 scripts integrated content from other Web sites by obtaining it dynamically. For instance, to integrate IMDB rating in other Web Applications for the same domain such as http://www.filmaffinity.com.
- Behavior-related: several of the analyzed scripts added behavior that changed the original functionality of the Web Application:
  - Add behavior: 38/50 scripts added some kind of behavior. For instance, one of the scripts added a *minimap* to a game (called Agar).
  - Remove behavior: none of the scripts among the top 50 removed any behavior.
  - Alter behavior: 4/50 scripts altered some kind of behavior. For instance, one of the scripts changed the behavior of the right click.
- Layout-related:
  - Reorder layout: 14/50 scripts modified the layout in some way.
- Interaction-related:
  - Add navigation: 14/50 scripts added one or more navigation features in some way. For instance, one of the scripts converted plain text into real links.
  - Add keyboard interaction: 7/50 scripts added new keyboard shortcuts. For instance, some of them added further keyboard control for YouTube video pages.
  - Add mouse interaction: 5/50 scripts added new mouse event-based behavior. A common feature was to add *onmouseover* event to images, in order to show them bigger when the mouse cursor is over them.
- Regarding to the target items: some scripts augment (applying some of the listed changes) a single information item per page while other augment several of them.

  Augment a list of items in Web page: 11/50 scripts repeated the same augmentation in a same Web page, one time for each similar target items in a list. For example, one of the scripts augments all Google Image results with a direct link to the image, which save some interaction steps to the user.
  Augment a single item in a Web page: 39/50 scripts performed an augmentation for a single item in the page. For instance, one of the scripts augments YouTube video pages with a new layout and also adds information to the video description.

Besides these 50 userscripts, we have browsed an additional 50 in different communities, reviewing descriptions, screenshots, user comments, etc., and noted that all of them address similar kind of augmentations listed above. Even more, we have studied userstyles too (from userstyles.org), a different kind of augmentation artifact. While only 14 of the top 50 userscripts were focused on layout changes, most of userstyles are implemented for this kind of augmentation. The reason behind this, as we said in Sect. 2, is related to the different technologies upon userscripts and userstyles are implemented.

3.3.2 Classification of Web Augmentation requirements

According to [11], where authors have analyzed several Web Augmentation extensions, this technique is mainly used for three main purposes: *refactoring* (for restructuring

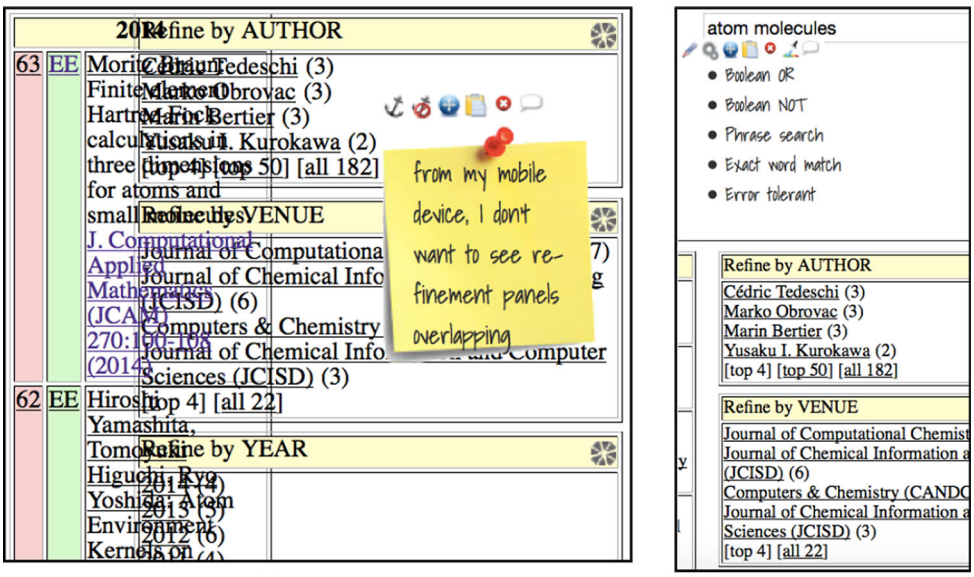

Fig. 4 DBLP usability enhancements. **a** Responsive DBLP requirement, **b** query modifier menu

Web pages to improve non-functional attributes), *customization* (for leveraging the existing Web page to perform the same functionality but in a more personalized way), and *modding* (for adding functionality to the Web page that originally was not conceived). We can establish a relationship between these three categories and the typical requirements classification: non-functional requirements (NFRs that include *refactoring*) and functional requirements (FRs that include both *customization* and *modding*). Considering that a CrowdMock requirement is composed of a User Story and a RAM, our approach is clearly oriented to FRs, given that both RAMs and User Stories are used to specify functional requirements. Most of the existing Web Augmentation approaches and repositories are oriented to FRs. This has sense because the technique is for *augmenting* a Web site, and then, some well-known non-functional requirements such as security are impossible to be improved only with Web Augmentation techniques because probably the application's code at server-side needs to be changed, while the augmentations are restricted to and applied on the client side. Nevertheless, some kind of non-functional requirements can be tackled within our approach.

The survey introduced above has let us identify the most common application aspects adapted by augmentation artifacts. These common kinds of alterations allowed us to define the set of what we call augmentation meta-requirements, which are classified on non-functional and functional requirements as follows:

- *Non-functional requirements* As we said before, it is important to notice that non-functional requirements cover extremely different sort of things such as efficiency, reliability, portability, usability, and security [19]. Some of these NFR categories cannot be tackled with Web Augmentation because these are strongly dependent on the server-side implementation of the application. However, those categories closer to the UI, such as usability or accessibility, can be improved with this technique. When the non-functional requirement is somehow related to the UI, our approach allows end users (by using MockPlug) to convert part of the existing UI into RAM widgets, which may be edited or moved. This makes it possible to specify, for instance, new layouts according to the user preferences or needs or even to the device used to access the Web sites in order to improve the responsiveness (an NFR) of the UI. For instance, in Fig. 4a, a user has defined a MockPlug model in which he shows a UI overlapping problem in mobile devices—whose viewport is more limited than in conventional, desktop-based browsers. In this case, he would expect a more responsive Web site that adapts the UI when he is accessing from his mobile device.
  Some usability issues are not related to the existing layout but related to how the original functionality is used. For instance, the DBLP search engine[13] allows users to specify several conditions for searching. As an example, if a user searches for "atom|molecules," he is indicating the logic operator OR, if he or she adds a "$" to one token, then he wants an exact word match, etc. There are several conditions to use in DBLP searches, but probably it would be easier to use them if the user has a

[13] CompleteSearch DBLP—http://www.dblp.org/search/index.php.

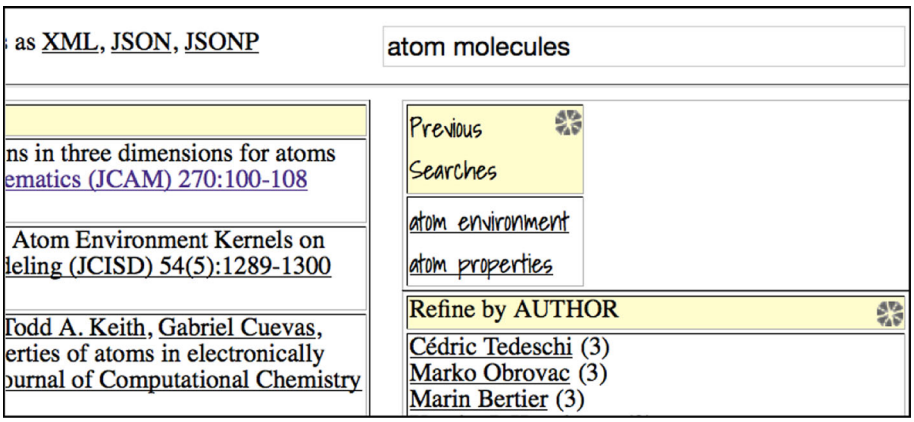

Fig. 5 New navigational panel based on previous searches

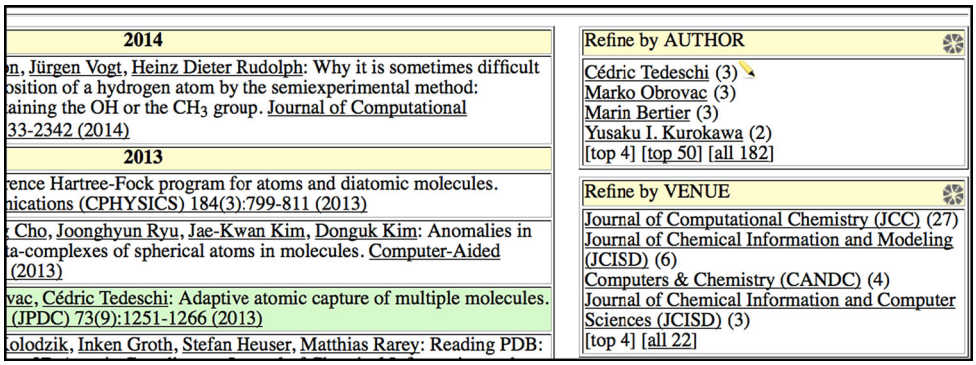

Fig. 6 Highlight author's papers on current page

query modifier menu that assists in the task of writing new queries. Such menu is shown in Fig. 4b.

- *Functional requirements* Regarding functional requirements, we have made a subcategorization considering most common augmentations in the repositories: navigation-based, behavior-based, content-based:
  - *Navigation-based* MockPlug models support the possibility of specifying requirements based on navigation, which involves the addition, alteration, and removal of anchors or behavior implying the execution of navigation actions at some point in third-party applications. As an example of such type of augmentation, we can show how to add a new panel to DBLP containing the previous searches that the user has made. MockPlug allows users to express easily this requirement by copying the UI of an existing panel (such as "Refine by AUTHOR") and edit its content, such as Fig. 5 shows. This addition implies a new set of anchors that trigger further navigations which were not present in the initial, non-augmented version of the page. In this example, a user has added the new panel "Previous Searches" as well by copying and editing the existing "Refine by AUTHOR" panel.
  - *Behavior-based* New functionality can be expressed with MockPlug with the addition of different kinds of widgets related to information entry and business process support. In addition, the alteration of how an existing functionality is already supported is possible through the introduction of changes in the original UI structure and behavior. As an example, let's consider the DBLP results Web page and its search refinement panels, particularly "Refine by AUTHOR" one. When the user clicks over one of the authors, another page is loaded showing the papers for that author. This can be useful in particular cases; however, sometimes it is annoying because the user needs to go to the previous page when he wants to see the original search. In this case, a user may want a new functionality for highlighting the author's papers in the current Web page, without navigating to a new page. A possible MockPlug model for this new functionality is shown in Fig. 6.
  - In this figure, two widgets have been added. First, a "highlight icon" was added next to the first author from the "Refine by AUTHOR" panel. Second, one of the result papers was selected as a widget ("paper

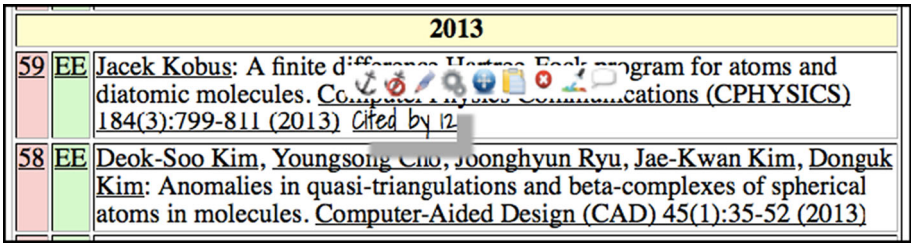

Fig. 7 Resulting paper augmented with "Cited by X" anchor brought from Google Scholar

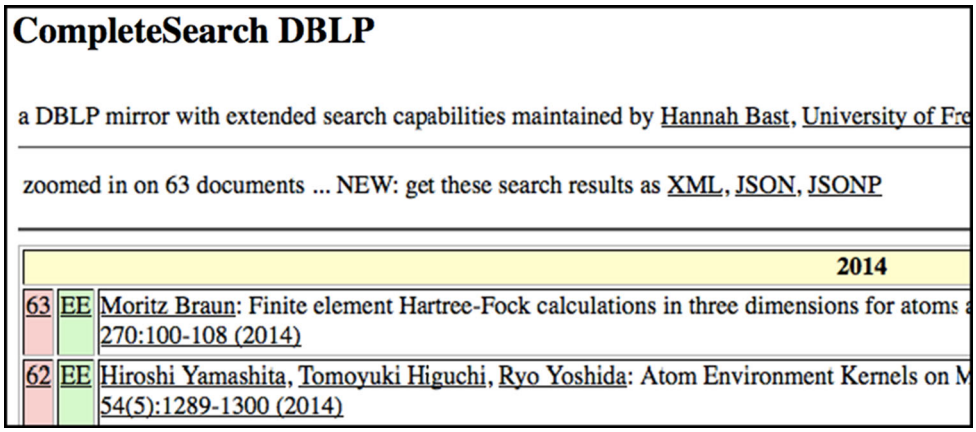

Fig. 8 Original DBLP UI

widget") for being manipulated. Finally, the *highlight icon* widget is set for changing the *paper widget*'s style when the user clicks on it. The new style is copied from another existing element in the Web page (Fig. 7).

- *Content-based* Requirements about new content may be expressed by removing existing content or even by bringing new content from other Web sites to indicate some desired integration. This is a technique widely used in Web Augmentation. Imagine a user who wants to see how many references a paper has at Google Scholar. Then, he may search one of the DBLP papers at Google Scholar, obtain the "Cited by X" anchor and add it to the corresponding paper at DBLP. Since this anchor will preserve its properties (such as textual content and *href* attribute, which were brought from Google Scholar), any other user who sees this MockPlug model will be able to understand which its aim is.

Besides these specific kinds of requirements, any other feature that cannot be represented or prototyped by a concrete visual component in the UI may be contemplated and described by special annotation widgets, as usual when using common mock-ups within traditional development processes. Some examples of annotation widgets are the sticky note in Fig. 4. This allows expressing augmentation requirements in textual ways but with a visual context. However, it should be noted that all the UI alteration aspects presented in the previous subsection are covered explicitly by our approach, which implies defining part of the behavior and UI alterations using the tool, in such a way that no textual annotations describing extra features to be implemented were required.

3.4 The CrowdMock process by example

This section is based on the example about the Peter's requirement, mentioned in the introduction. The original Web page of CompleteSearch DBLP (see Fig. 8) only shows some information (title, authors, etc.) for the resulting papers, but it does not include the abstracts, which is one of the changes that Peter wants. Besides that, he also wants to add the paper easily in Mendeley. For these two aspects, Peter has defined a first RAM, shown in Fig. 9. This RAM includes two anchors, "See Abstract" and "Add to Mendeley."

Here, we show one of the relevant parts of our approach; instead of defining these requirements textually, drawing them by hand or using existing mock-ups tools, CrowdMock allows specifying augmentation requirements graphically and interactively through its supporting tool MockPlug [17], creating a RAM instance. This tool is explained in Sect. 4.1.

As we mentioned before, our approach proposes to complement the RAM with a User Story; Peter has also written the corresponding one:

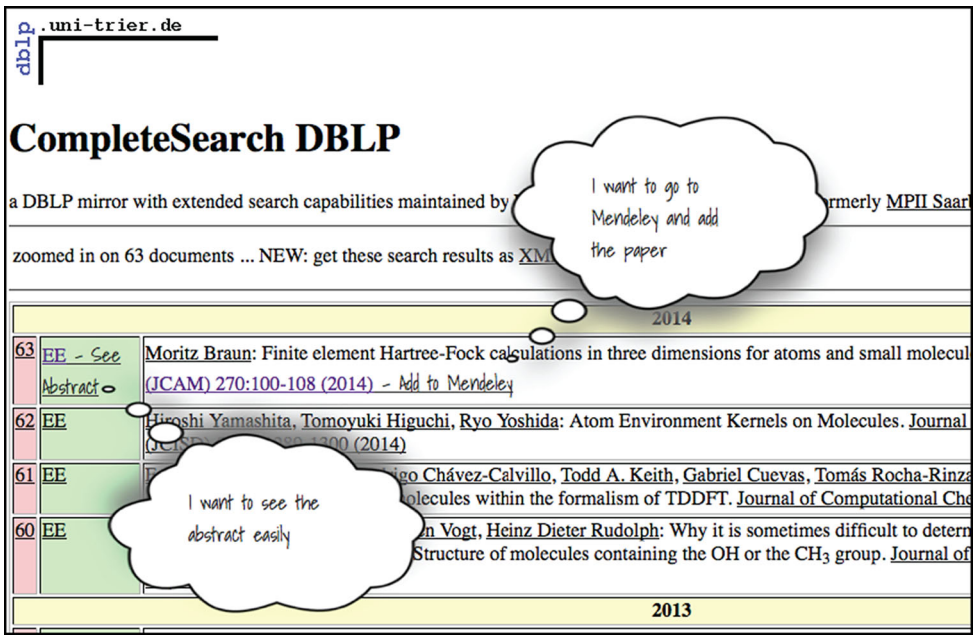

Fig. 9 First MockPlug model defined for Peter's User Story

As a: researcher
I want: to easily add papers in my Mendeley Library from DBLP resulting papers when their abstracts are relevant to my research
So that: I can survey the papers I want to read later

Now that we have introduced the original Peter's requirement specification, and in the rest of the section, we describe the process and activities involved to share, refine, prioritize, and implement it.

For instance, let's imagine another user, *John*, who considered that it is easier to perform Peter's task by integrating Mendeley into DBLP instead of navigating to Mendeley and filling the form with the paper's information in that Web site, as Peter proposed initially. With this in mind, John refined the original Peter's User Story in this way:

As a: researcher
I want: to integrate Mendeley into DBLP search for easily add papers to my library when the abstracts are relevant for my research
So that: I can survey the papers I want to read later

In this case, John has not defined a new MockPlug model; however, another user of the community, *George*, who read both versions of the User Story, decides to collaborate by defining a new MockPlug model for John's User Story. In this way, he defines a more complete prototype considering the integration of both the abstract and the Mendeley's Web form for adding a paper. First, he took the original MockPlug model defined by Peter, which has two anchors. His resulting model, shown in Fig. 10a, is pretty similar to Peter's one a priori. However, instead of navigating to external Web pages, these anchors were defined for toggling the corresponding UI component. The "See Abstract" anchor, for a given paper, shows/hides the corresponding abstract, as shown in Fig. 10b; additionally, the "Mendeley" anchor shows/hides the Mendeley Web form inside DBLP (see Fig. 10c). Note that this kind of behavior specification may be established because the widgets that users manipulate, instead of being just styled boxes as in common mock-up tools like Balsamiq,[14] are actually real DOM elements. Users may configure his behavior by using visual tools provided by MockPlug.

In Agile approaches, there is a refinement session (which was previously called grooming backlog session [8]) consisting in refining the backlog by adding, removing, or changing tasks in it. The difference between our approach and this kind of sessions is that our refinement is collaborative, in the sense that one end user (who may also be a user with programming skills) writes one requirement and another one can refine it. The same person can perform the whole definition, as in regular agile approach where the product owner is the one who concentrates this activity. Nevertheless, our approach encourages the collaborative refinement, because two or more different people are analyzing the same requirement in the same way they are also validating it at the same time. In Agile approaches, the validation occurs mainly in the review session after the requirement is implemented. In our approach, the validation occurs previous to implementation (in most of the cases) as recommended in classic requirement engineering

[14] Balsamiq Mockups—http://balsamiq.com/products/mockups/.

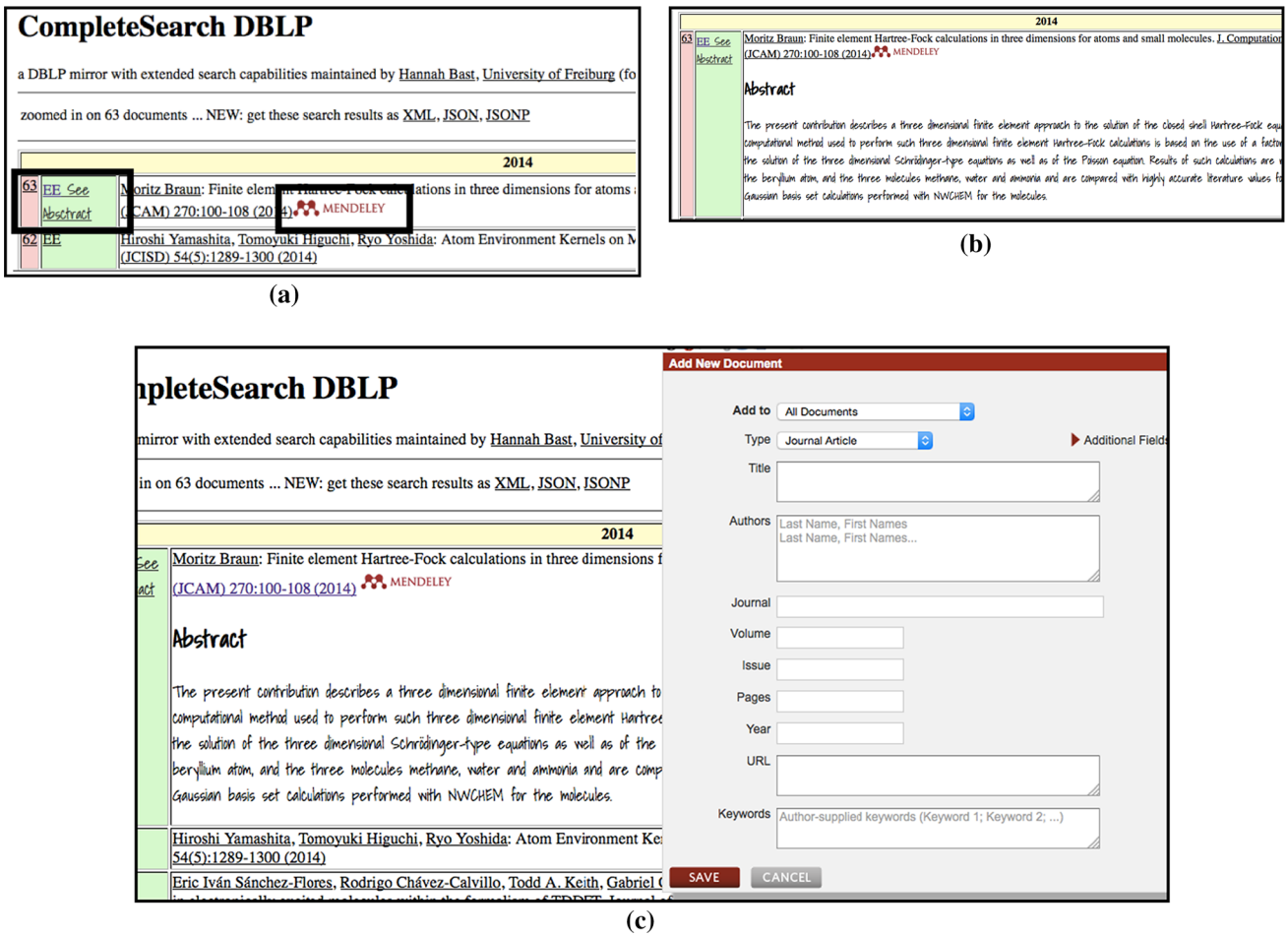

Fig. 10 Requirement defined over DBLP for supporting the task of adding papers in Mendeley's library. **a** DBLP result page augmented with "See Abstract" and "Mendeley" options. **b** When the user clicks on the "See Abstract" option, it shows the paper's abstract obtained from the Editorial's Web page. **c** When the user clicks "Mendeley" button, it shows a pop-up corresponding to the Mendley's Web form for adding a paper in the user's Library

approaches, with the technical assistance of the implemented tooling. Moreover, the use of RAMs in the requirement definition provides a benefit to validation since stakeholders can see and even interact with the final UI before it is implemented.

In Fig. 11, we show the evolution of Peter's requirement, which depicts the refinement activity. The presented tree in the figure shows the different versions of the requirements and its precedent version. Each node represents a version, and the label is the user who created it, except for the root, whose label is the requirement name. As it can be seen, there is first a branch corresponding to the User Story refinement (performed by John), and then another version of the requirement is achieved when another user (George) created a MockPlug model responding to this last version of the User Story. This last version of the MockPlug model is presented in Fig. 10c.

As the last comment, note that the requirement prioritization activity is contemplated also in Fig. 11, in which the last MockPlug model defined by George is highlighted for being the most voted by the crowd of users. However, as it will be explained in Sect. 4.3, other aspects of the implementation effort could be taken into account in order to prioritize a RAM version.

4 CrowdMock Implementation

Our approach is supported by two tools, namely MockPlug and UserRequirements. In this section, we describe the features of both of them and how they work together.

4.1 MockPlug for end users

MockPlug is a client-side tool deployed as a Web browser extension designed for allowing end users to *plug* mock-ups on existing Web pages—what we called RAMs. Using this tool, end users can specify their augmentation

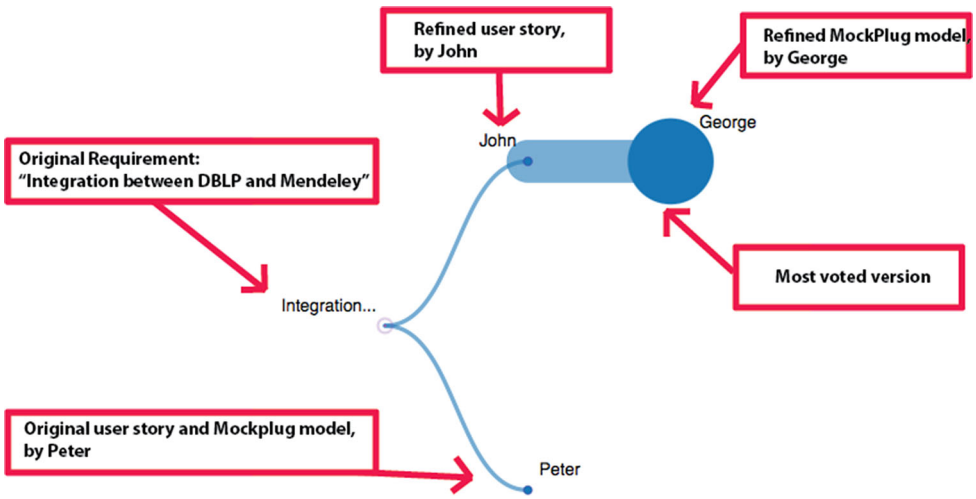

Fig. 11 Evolution of Peter's requirement

requirements inside their natural context: the target Web site. Although an augmentation requirement expressed using MockPlug is related to a User Story, in this section we focus mainly on the MockPlug tool and the MockPlug model.

One of our priorities in the design of the MockPlug was to provide a tool that allows users without any technical knowledge to define their RAMs. A MockPlug model represents a set of intended modifications over an existing Web site. Users materialize these modifications by adding or manipulating different kinds of *widgets*. From the user's point of view, these widgets have a hand-drawn visual style, as usual in tools such as Pencil[15] or Balsamiq.[16] However, MockPlug renders widgets as ordinary DOM elements, which enables the possibility of specifying further behavioral aspects very easily, achieving high-fidelity mock-ups [39].

A user may drag widgets from the MockPlug Panel and drop them into the desired position on the current Web page in order to define his or her requirements. Figure 12 shows the MockPlug Panel and, in particular, the widgets palette tab. The same figure shows also how the user has added a web link "Add to Mendeley" on the result search page of DBLP. Since our tool works with real DOM elements, a Widget insertion has a real effect in the UI Web page code loaded in the Web Browser. To make clear this point, we show in Fig. 13 the same fragment of the Web page before the insertion and after the insertion. Note that at the bottom of Fig. 13, where code altered by MockPlug is shown, there is a new anchor that has several attributes specifically set for the use of MockPlug. This is one of the key features of the RAMs used to specify augmentation requirement in the approach.

Although the MockPlug palette has predefined components (including widgets such as lists, buttons, and text fields), users may also convert any existing DOM element into a widget that can be further manipulated (we call them *Collected widgets*). For instance, let's imagine a user who desires a different refinement panel order, thus changing the page layout. Then, he probably wants to *cut* each panel and *paste* them in another place. For doing it, the user may *collect* the existing "Refine by VENUE" panel (Fig. 14a) and work with it as any other supported widget in the tool, moving it (for instance) to the top (Fig. 14b).

When widgets are *collected* in other Web sites, we call them *Pocket widgets*. Pocket Widgets can be used to express augmentations reusing functionalities already present in existing web sites.

In Fig. 15, we show the Pocket tab in MockPlug. This tab allows users to use the "Collect" functionality, which is available on every Web site. When the user clicks this button, some highlighting is enabled and the user may select specific DOM elements that will be then available in the pocket.

As an example, consider the example shown in Fig. 7, which is about integrating the "Cited by X" anchor from Google Scholar in DBLP search results. For specifying this requirement, the user collects that anchor from Google Scholar. By using the pocket on that Web site, he or she selects the necessary DOM element, shown in Fig. 16a.

Finally, when the user comes back to DBLP in order to continue defining his requirement, he opens the Pocket tab in MockPlug, where the "Cited By 12" widget is available. At this point, a Pocket Widget may be inserted in two ways. On the one hand, it can be inserted as a DOM Element, copying all the children nodes if there any. In this

[15] http://pencil.evolus.vn.

[16] https://balsamiq.com.

Fig. 12 Widget palette tab of mock-up/RAM panel

Original Web Site Code

```
<td>
    <a href="http://www.dblp.org/pers/hc/b/Braun:Moritz.html">
        Moritz Braun
    </a>: Finite element Hartree-Fock calculations in three dimensions for atoms and small molecules.
    <a href="http://www.dblp.org/db/journals/jcam/jcam270.html#Braun14">
        J. Computational Applied Mathematics (JCAM) 270:100-108 (2014)
    </a>
</td>
```

Web Site Code altered with MockPlug

```
<td>
    <a href="http://www.dblp.org/pers/hc/b/Braun:Moritz.html">
        Moritz Braun
    </a>: Finite element Hartree-Fock calculations in three dimensions for atoms and small molecules.
    <a href="http://www.dblp.org/db/journals/jcam/jcam270.html#Braun14">
        J. Computational Applied Mathematics (JCAM) 270:100-108 (2014)
    </a>
    <a href="" class="mockplug-widget" mockplug-widget-id="mpwidget-anchor1" mockplug-abstract-widget="mpwidgetanchor">
        Add to Mendeley
    </a>
</td>
```

Fig. 13 Widget insertion effect

case, the added widget may inherit the visual style from the Web page in where is inserted, thus losing in this way relevant visual style. On the other hand, a Pocket Widget can be inserted as an image, in order to let users preserve its original style and appearance.

4.2 MockPlug for scripters

Despite scripters can make use of the same MockPlug functionality already available for end users (see Sect. 4.1), they may specify further aspects about the requirement that

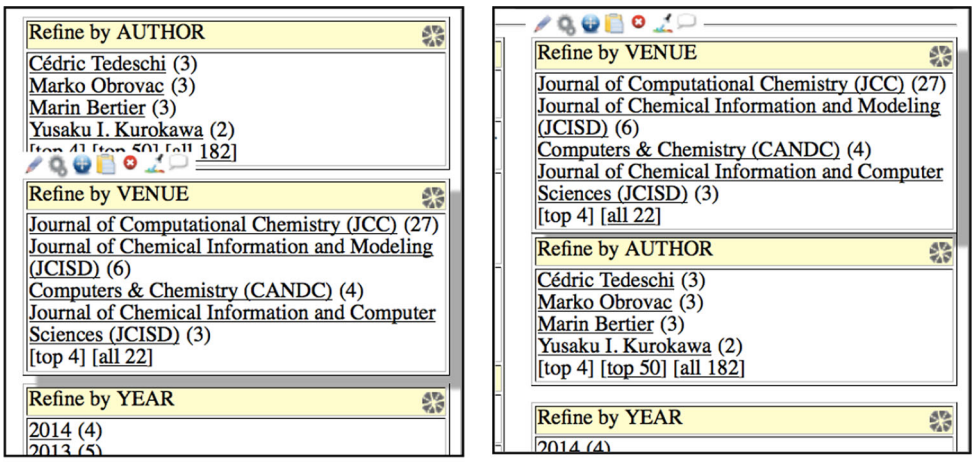

Fig. 14 Collected Widget and its manipulation. **a** "Refine by VENUE" panel converted into a widget, and **b** "Refine by VENUE" relocated to the top of refinement panels

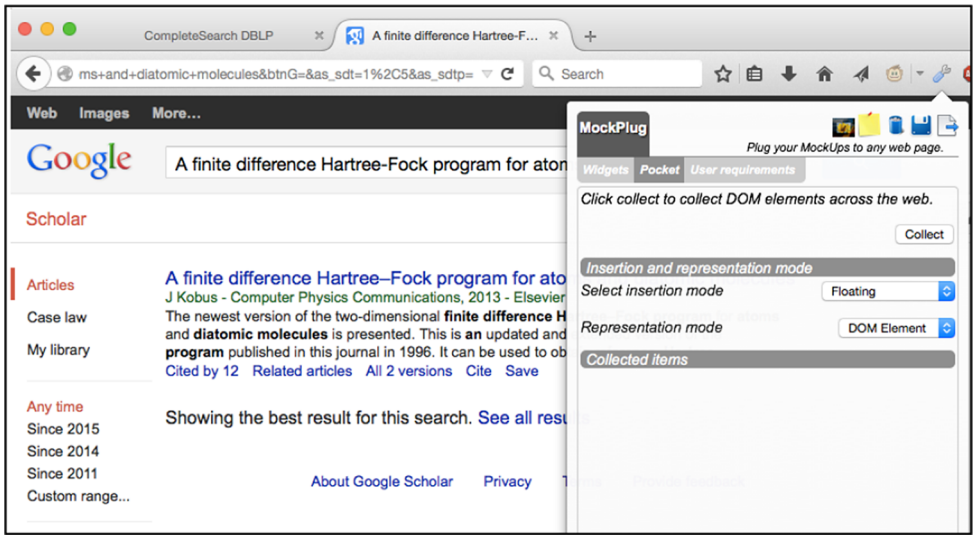

Fig. 15 MockPlug's Pocket tab

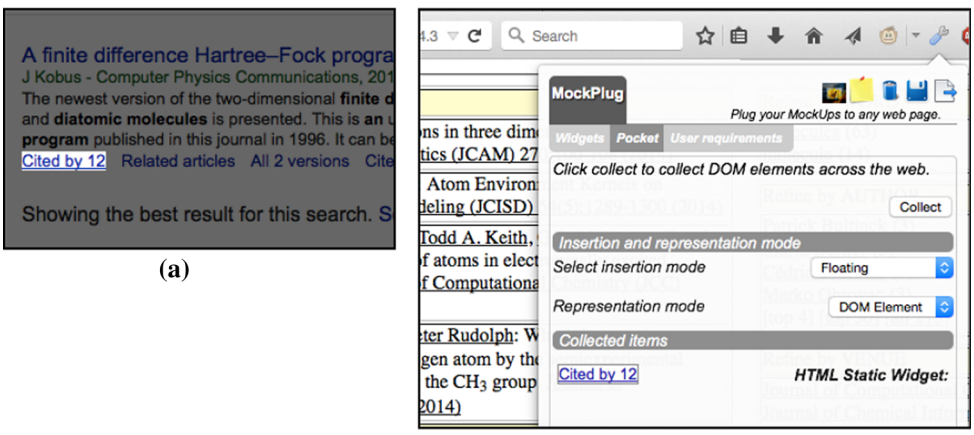

Fig. 16 MockPlug's Pocket tab. **a** Highlighting target DOM element, **b** collecting a DOM element

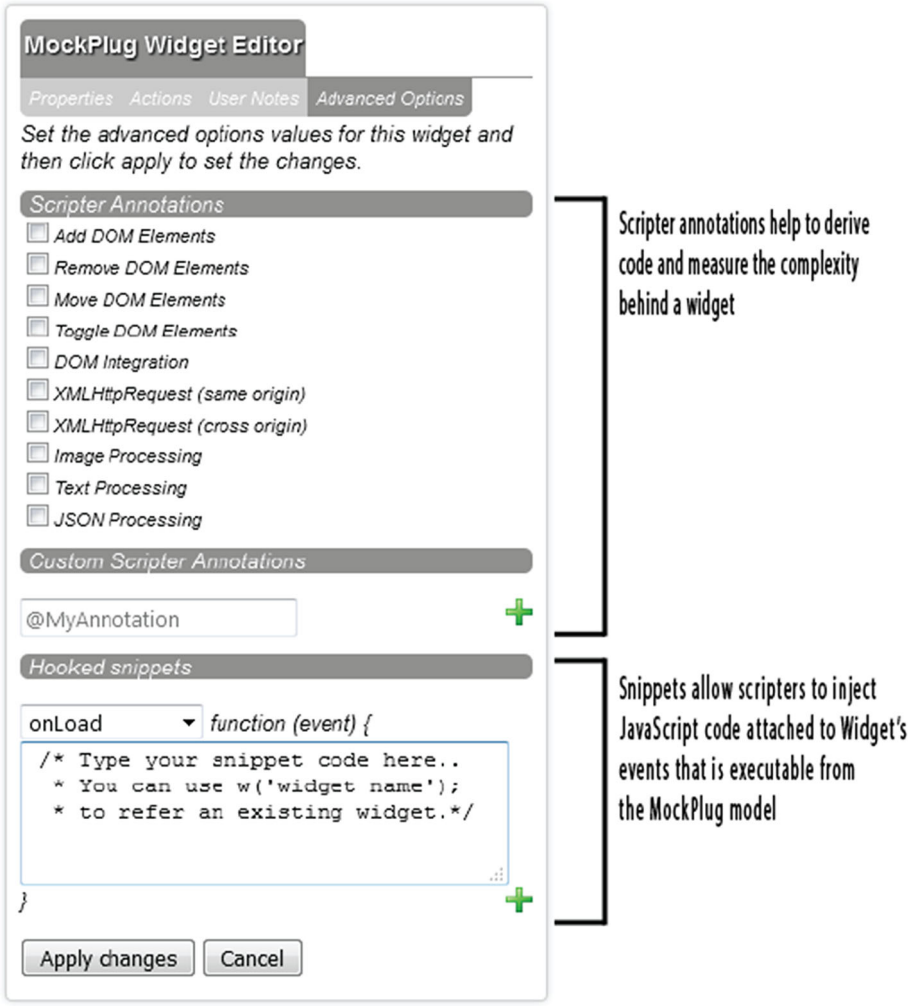

Fig. 17 Widget edition for advanced users or scripters

will help to (1) make closer the definition to the final solution, (2) to improve how the requirements are weighted and estimated (for identifying the most convenient version to implement), and (3) determine how the final script is generated.

We have envisioned two ways to empower scripters in the refinement of the requirement, both of them related to the widget edition, which is appreciated in Fig. 17. When editing a widget, there is a tab named "Advanced options" that allow scripters to specify low-level aspects of the widget behavior. There are two ways: selecting "annotations" and "code snippets."

Annotations make possible to define some aspects related to the complexity behind a widget. For instance, if the requirement implies to obtain images from Google Images to be integrated into another Web site when the user click a button, then a scripter can establish that the widget representing that button has the annotation "XMLHttpRequest (cross-origin)." In this way, the scripter is explicitly specifying that a request to another Web page has to be done for implementing the requirement. Additionally, if a scripter wants to refine the requirement by actually obtaining the images from Google Images, he may add a JavaScript snippet and attach this code to a particular event. Then, the next time that the interested user opens the MockPlug model, he can interact with the RAM with some functionality really implemented.

4.3 UserRequirements

UserRequirements is a requirements repository with social features that manages CrowdMock requirements, described through a RAM and a User Story. UserRequirements supports requirements definition, refinement, and voting. Figure 18 shows different views of Peter's requirement listing its different versions in the repository. Each version is composed by a User Story description, the widgets used in its underlying MockPlug model, and the number of votes. The user can open and see the MockPlug corresponding to a concrete augmentation requirement version,

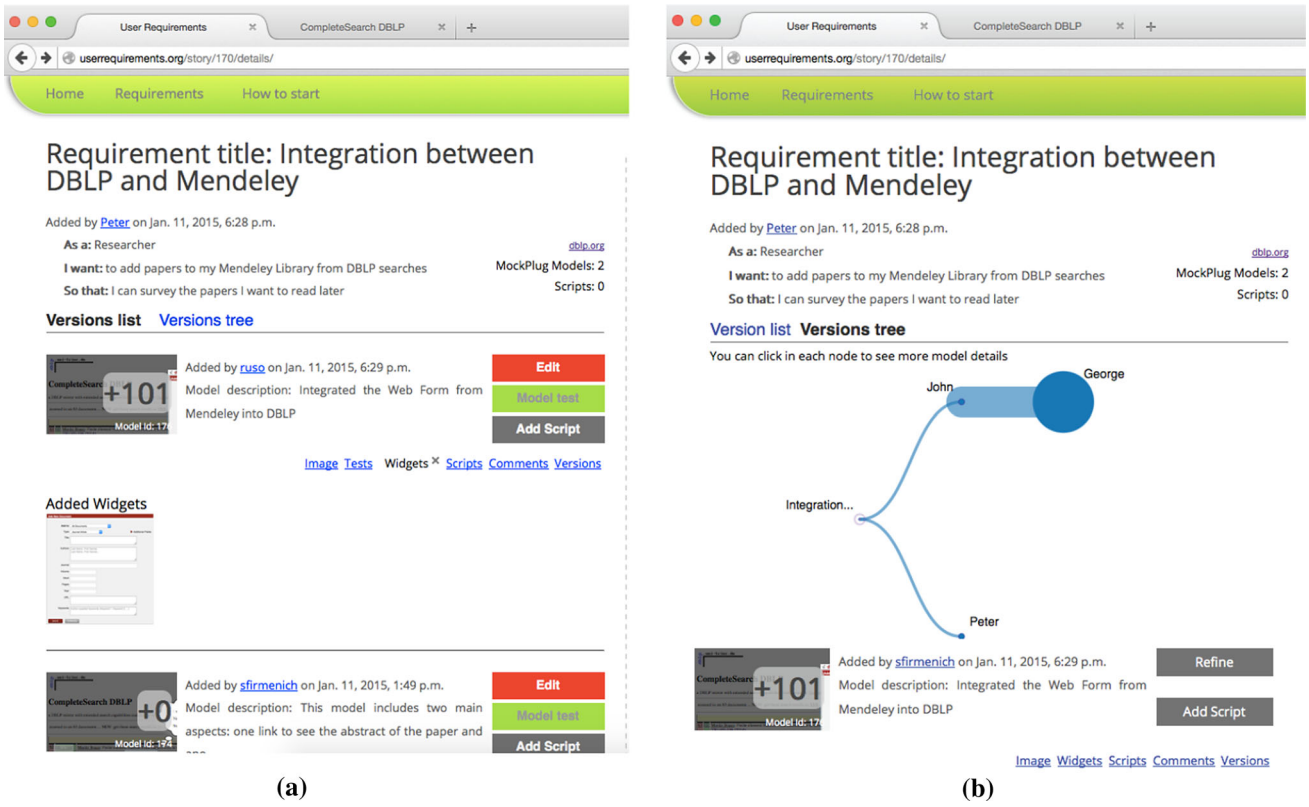

Fig. 18 UserRequirements: requirement details. **a** Requirement view: version list, **b** requirement view: version tree

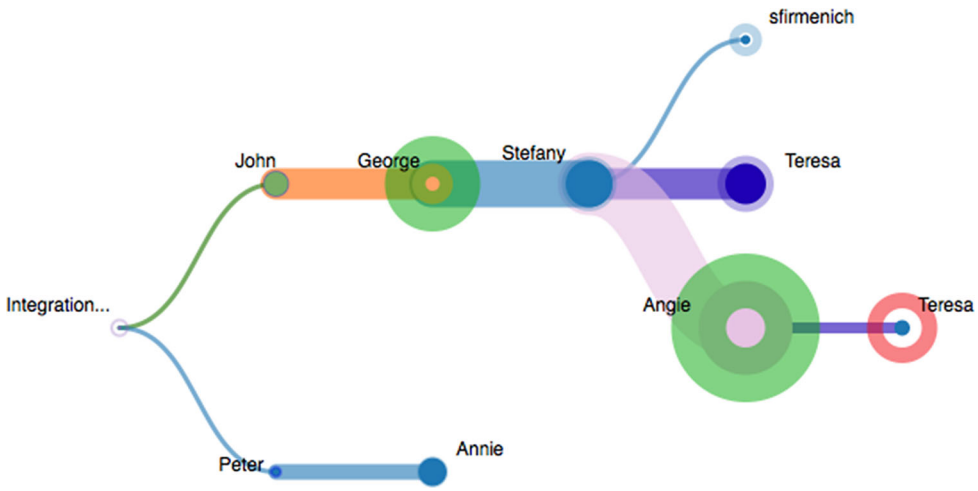

Fig. 19 A User Story evolution tree in UserRequirements

among several other options. A requirement version can be refined using the *edit* button. The requirement list is ordered by votes, but the user can see the evolution of the requirement through a version tree as shown in Fig. 18b. In this view, the user can click on a particular tree node to visualize the requirement and he or she can also create new branches. The versions tree view is a valuable tool for understanding the evolution of a requirement; the community can quickly see which is the most voted version, the most discussed one, and also if the user who defined the original requirement accepts or rejects some of the versions. Just as an example, in Fig. 19 we show a more complex tree, in which other users have been participating.

There are several aspects to be considered in order to understand the version tree visualization. First, it is important to mention that each user has an associated color

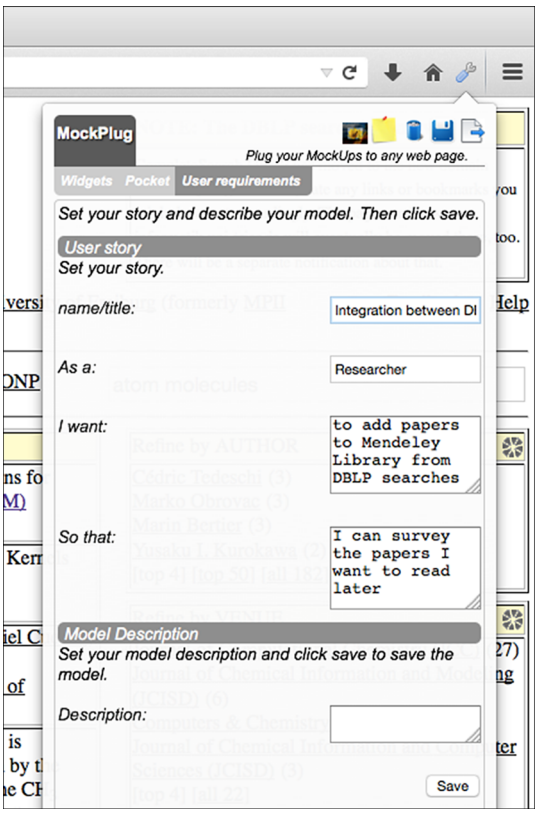

Fig. 20 Interaction with MockPlug: creating a new requirement from MockPlug

in the tree. This allows to easily visualize the contribution of each participant. Second, the size of tree nodes is defined according to their votes; then, bigger nodes are those that users consider better. However, other information about the requirement specification (such as scripters' RAM definitions) may be visualized in the platform. Third, nodes may have also a border stroke, whose width represents the discussion on the corresponding version of the requirement. If this stroke is wide, it means that it has several users' comments. Finally, the border stroke color has also a particular meaning. If this color is green, it means that the user who has defined the requirement originally (in the presented example, Peter) agrees with that version. If this color is red, then he or she is rejecting it. When this stroke color is the same that the color filling the node, it implies that the user who originally created it, i.e., Peter, has not given any feedback about the requirement version so far.

In the example from Fig. 19, we may easily see that the Angie's version of the requirement is the most voted (because it is the biggest node in it) and, at the same time, it was also commented by several users (a wide border stroke). In addition, Peter approved this version (green border stroke). In the same example, we may appreciate that Peter has rejected Teresa's version that has been derived from Angie's one (red border stroke).

As a final comment about the underlying implementation of UserRequirements, it is important to say that a requirement could be first created on the platform without an associated RAM. In cases like this, the user may choose among creating a single User Story, or a more complex requirement involving more than a single User Story, which is called *theme*. In this way, the same requirement could be composed by several user stories, each one with a RAM.

4.4 MockPlug: integration with UserRequirements

The CrowdMock approach is fully supported through the integration between *UserRequirements* and *MockPlug*. In order to create a new requirement in UserRequirements, a user may use MockPlug for defining the RAM expressing the augmentation required in the target Web site. After that, from the same MockPlug panel, he or she can save it in the UserRequirements platform. In order to complete this task, the requirement's title, a User Story, and a model description must be provided. Figure 20 shows how Peter can perform this task.

4.5 MockPlug metamodel and code generation

MockPlug let users specify runnable, augmentation-based mock-ups (RAMs). At an implementation level, if we want to augment a DOM with new widgets, we have to specify how those widgets are woven into the existing DOM. With this in mind, we defined and implemented the MockPlug metamodel. This metamodel defines the expressivity of MockPlug models and, consequently, how augmentation requirements are defined and what kind of requirements (i.e., which concrete augmentation functionality) can be specified. As shown in Fig. 21, both the name and the URL of the augmented Web site are part of the MockPlug model. It is worth noting the importance of the property *url* in a MockPlug model, since this is the property that defines which Web site (o set of Web sites when using a regular expression) the model is augmenting.

The specification of new widgets relevant for the requirement (Widget class) may be defined in the model. Widgets can be simple (SimpleWidget, atomic, and self-represented) or composite (CompositeWidget, acting as a container for another set of Widgets). All widget subtypes have several characteristics in common:

- They belong to a MockPlug model that has both a type and a set of properties.
- They are prepared to respond to several events, such as *mouseover* or *click*.
- They can react to an event with predefined operations defined according to the underlying DOM element type

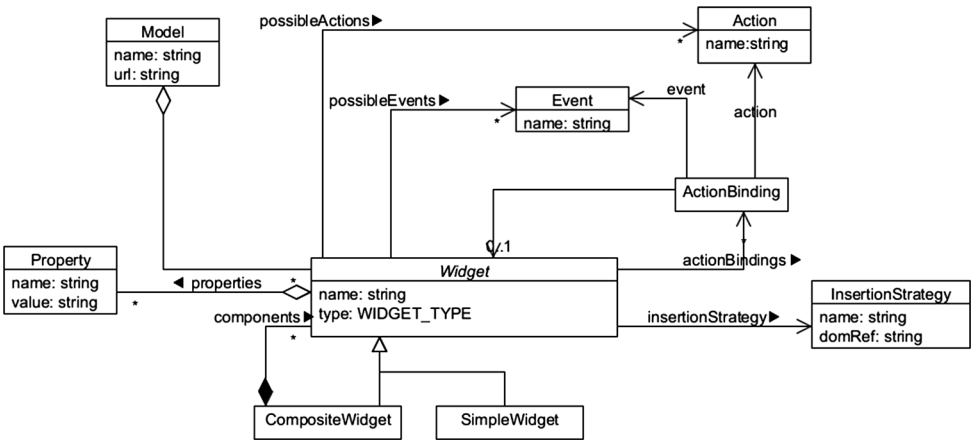

Fig. 21 MockPlug metamodel

> Scripters may define customized operations and attach these to Widget's events in order to express the real, executable behavior expected by users

Each widget has associated an insertion strategy (InsertionStrategy), which describes how and where it is inserted into the DOM tree (*float*, *leftElement*, *rightElement*, *afterElement*, *beforeElement,* etc.). These strategies, depending on the referenced DOM element (for instance, the parent), must use the *domRef* property or not.

As a first advantage, the MockPlug metamodel allows users to share and edit their requirements since the MockPlug tool can interpret them as runnable models and they are not just images or static visual elements added to the UI. A second advantage is that MockPlug models (defined through its metamodel) can be processed, in order to generate code, as it occurs in well-known model-driven approaches [20]. Although we are aware that the MockPlug metamodel is not expressive enough to generate all the necessary logic for all kind of requirement, we have successfully generated several code stubs that helped to develop the full, final userscript. In the context of Web Augmentation, it is complex to guarantee that the code generated automatically is going to be adopted by artifacts developers, which is a problem related to model-driven engineering field in general [40]. For instance, there are hundreds of JavaScript libraries to be used that can help to write Web Augmentation implementation code, as well as several ways for structuring it. To cope with this, we have defined an extension point for code generation. This means that developers may create their own code generator just creating a JavaScript object that responds to a specific message, receives as an argument the XML file that describes the MockPlug model, and returns an arbitrary complex stub of code. For this paper, we have developed one of the big set of potential code generators, which was used for the evaluation presented in the following section. For the sake of conciseness and coherence, we omit further descriptions of the code generation process that can be read in [17].

5 Evaluation

This section describes a preliminary evaluation of the CrowdMock approach and its supporting tool MockPlug. In particular, we focused on the capability of the approach to express requirements because this is its most important contribution and we defined a novel supporting process and tooling for that. We performed a controlled experiment focused on assessing the satisfaction achieved by experienced Web users (who played the role of end users) when specifying augmentation requirements on existing Web Applications. We also evaluated the satisfaction level of the scripts produced (by a group of developers who formed part of the experiment team) to implement such requirements.

The experiment compared the results of specifying requirements using CrowdMock against using common requirements definition approaches in the industry. In order to accomplish such comparison, participants had to produce the augmentation requirements specifications, then a group of developers implemented them, and finally participants had to evaluate how the implementation satisfied their original requirements. The goal of the experiment is described according to the Goal/Question/Metric (GQM) method formulated by Basili et al. [3].

> Analyze *CrowdMock and traditional Web Augmentation development approaches*
> for the purpose of *evaluation of the approach*
> with respect to their *effectiveness*
> from the point of view *of the potential users*
> With respect to the *satisfaction achieved by the users who specify requirements and inspect products constructed from the requirements.*

The description is organized mainly according to the template proposed by Shull et al. [34]. It has five main subsections. The first section ("Experiment Planning") describes the plan and protocol that was used to perform the experiment and analyze the results. Then, the "Deviation from the plan" subsection comments the execution details. After that, the "Analysis" subsection summarizes the data collected and its treatment. Then, Threats to Validity are analyzed, and finally, the results of the experiment and their implications are discussed.

5.1 Experiment Planning

This subsection describes the protocol used to perform the experiment and analyze the results.

5.1.1 Goal

The goal of the experiment can be refined into the following one:

> Goal: Analyze *CrowdMock and traditional Web Augmentation requirements specification techniques*
> For the purpose of *understanding their effectiveness*
> With respect to the *satisfaction achieved by the users who specify requirements and inspect products constructed from the requirements.*

5.1.2 Participants

A group of 20 participants were involved in this evaluation, 9 females and 11 males, aged between 22 and 60. All of them were professionals on Computer Science degrees from postgraduate courses on Web Engineering which were held in three different Universities: Universidad Abierta Interamericana (8 participants), Universidad Nacional de La Plata (8 participants), and Universidad Nacional de la Patagonia San Juan Bosco (4 participants). Participants did not have experience in *CrowdMock* while reported experience in other requirements engineering techniques. Thus, they were able to assess whether traditional techniques they are accustomed to use were more effective or not that *CrowdMock* according to the time and effort invested in requirements specifications. Moreover, since software engineering professionals have more experience in specifying and validating requirements, thus we preferred to involve these groups of people in order to avoid the bias that non-technical end user could incorporate to the experiment. Participants had experience ranging from 7 to 18 years in Web Engineering, some of them as developers, others as analysts, and some others as team leaders. Some participants were also teachers at the University. The majority of the participants were Argentinean, but there were also people from Ecuador and Sweden. The group of participants had diversity in experience (years, roles, country and context—academia vs. industry), and we think that this variety is beneficial for the experiment.

5.1.3 Experimental material

The techniques used to specify augmentation requirements were two. One of them was the tool we propose and is explained in previous sections. On the other hand, participants had to specify requirements using other technique already known by them, so they used a common word processor to build textual document with the specification, which included screenshots in some cases.

Other products used in the experiments were questionnaires.[17] There were two kinds of questionnaires, one to provide information about the task of specifying requirements and other to validate the functionality of the script developed to fulfill such requirements. The first questionnaire asked information related to the augmentation feature selected to be implemented, the difficulty perceived during the task, and the time they needed to perform it. The second questionnaire asked information about the script installation, execution, the script satisfaction (perceived by the participant), and eventually and optionally the reason.

5.1.4 Tasks

Participants had to perform two tasks: specifying augmentation requirements they desire and testing the resulting script that implemented it. After every activity, they had to fill in a questionnaire. They had to specify four requirements, two of them using an already known technique and the other two using RAMs. During the requirements specification tasks, they had to fill in a questionnaire (an *activity diary*) to measure the complexity of specifying requirements using the corresponding strategy. The activity diary consisted in the following information: beginning and ending time of the working session, activities performed, and problems aroused. After the initial questionnaires have been filled and when the scripts fulfilling the desired requirements have been implemented, participants had to install and test the scripts produced by scripters according to the requirements specified. Then, they had to fill in a questionnaire about the level of satisfaction of the scripts.

All the tasks were performed at home. Thus, the interchange of artifacts (questionnaires, scripts, etc.) was accomplished in the following way: (1) Requirements specified using an already known technique and the

[17] http://www.lifia.info.unlp.edu.ar/crowdmock/public/crowdmockexperiment.zip.

corresponding activity diary were recorded in a PDF document sent by email. Scripts were sent to participants via email. (2) Requirements specified using RAMs were referred through UserRequirement URLs in a PDF document within the activity diary also sent by email. Scripts were uploaded in UserRequirements, and participants downloaded them in their browser and filled in satisfaction questionnaires in Google Forms. The augmentation requirements had to be based on some general features that we provided over well-known existing Web Applications: IMDB[18] and YouTube.[19] Possible features for IMDB were as follows:

- F1.1. Filter the list of 250 best movies under certain criteria.
- F1.2. Add some information to the 250 best movies list.
- F1.3. Change the layout and/or content of a movie page based on the following allowed operations: (1) move elements of the page, (2) remove elements from the page, (3) add new widgets into the page (button, input box, menu, etc.), or (4) add contents related to the movie from another IMDB page.

Allowed features for YouTube were as follows:

- F2.1. Add information about the videos in the search results.
- F2.2. Change the layout and/or content of a video's Web page based on the following allowed operations: (1) move elements of the page, (2) remove elements from the page, (3) add new widgets into the page (button, input box, menu, etc.), or (4) add contents related to videos from another YouTube page.

We chose these specific set of general features since they contemplate the meta-requirements representing the most common kind of changes that existing augmentation artifacts perform over Web sites according to the survey presented in Sect. 3.3. Note that in all the cases, these general features are only partially specified, and participants had to decide how to define more fine-grained aspects. As we explained in Sect. 3.3, we made a survey that, beyond of being coincident to other studies [11], has shown that Web Augmentation is used mainly in three dimensions, which are content, functionality, and layout. Regarding the content, we showed that the majority of the augmentation requirements desired by users are related to adding content to Web pages, both to a single information item (for instance, a movie page in IMDB) and to a list of information items (such as a video search results in YouTube). Both ways to add content are different, because while the former implies applying the augmentation only to specific DOM elements, the later needs to apply the same process to a set of DOM elements iteratively. We wanted to be sure that our tool was appropriated for specifying both kinds of content-based requirements. Tasks F2.2(4) and F1.3(4) were aimed to define single content augmentation requirements, while F1.2 and F2.1 are meant to define an augmentation that must be applied over a set of DOM elements. Besides adding content, removing content is considered in F1.3(2) and F2.2(2).

As a result of the survey shown in Sect. 3.3, we discovered that adding functionality was the second most pursued kind of augmentation, at least in the context of the participant sample chosen for our experiment. Mostly, the functionality to be added is implemented by using forms, menus, buttons, etc. (i.e., interaction widgets that allow users to have further behavior). We have covered this kind of augmentation with F1.1, F1.3(3), and F2.2(3). Finally, layout reordering is another popular augmentation requirement. In this case, we have defined F1.3(1) and F2.2(1) in order to allow participants to define the specific requirements related to layout aspect.

Beyond the result of the experiment (i.e., the comparison between techniques for specifying the requirements, which is shown in the remaining of Sect. 5), at this point it is important to say that the definition of these five coarse-grained requirements was enough for obtaining concrete requirements specifications based on the mentioned popular augmentation requirements dimensions (content, functionality, and layout) that were assessed in our survey. For example, a participant noted that filters must be accessed in IMBD after clicking the option *Subscription administration,* but he wanted a direct button to filters. This issue was related to Feature 2.2, and the participant specified a requirement to add a button in the main window to access the filters (which merge aspects about content and behavior). Another participant wanted more information about videos related to music in the search results. Thus, he asked to include a link to the lyrics of the song. This requirement was related to Feature 2.1. Other participant refined R1.3 for adding YouTube videos to IMDB movies (trailers) in a modal window that is opened when the user clicks a new anchor. There were really different examples among the requirements defined by participants, most of them use at least one of the meta-requirements introduced in Sect. 3.3.1, while others are based on the combination of two or more of them.

The task of testing the scripts was very simple. Participants had to install them as an extension in the web browser, and after that, they had to visit the Web site they had referred in the requirements specification document in order to assess whether the functionality provided satisfied their needs. Since augmentation requirements are very

[18] Internet Movie Data Base - http://imdb.com; last accessed February 4, 2014.

[19] YouTube—http://youtube.com; last accessed February 4, 2014.

specific, it was easy for participants to state whether the script satisfied or not their requirements and it was not necessary to specify a list of acceptance criteria. Nevertheless, sometimes could happen that users were not fully satisfied and we provided the option "partially satisfy" for this situation. For example, in the requirement of adding a button to access the filters, the implementation may not satisfy the requirements because the position of the button is not the one that the user expected. In this case, this implementation can be assessed as "partially satisfy."

5.1.5 Hypothesis and variables

The experiment had only one goal: comparing the satisfaction perceived by participants when they test scripts which functionality was described using CrowdMock against other well-known techniques. Therefore, our hypotheses are as follows:

H_0: There is no difference in the satisfaction
by using CrowdMock and traditional approaches
H_A: There is a difference in the satisfaction
by using CrowdMock and traditional approaches

Three types of variables were defined for the experiment: independent, dependent, and control. The technique (i.e., CrowdMock, Narrative description, and User Stories) is an independent variable because it is what we vary in order to test the results. The level of satisfaction (i.e., "does not satisfy," "partially satisfy," and "completely satisfy") is a dependent variable because we want to measure how they vary according to the technique. Finally, features subset is a control variable because the set of features is constant and unchanged, and participants can specify similar kind of requirements from the features to make the comparison of strategies possible.

Although the questionnaires provide much information, we focused the objective of the experiment on the level of satisfaction because the main goal of requirements engineering is to capture the correct requirements to build the right product. Requirements usually refer to necessities, wishes, and expectations, and some of them are tacit, so stakeholders are the only one who can determine the satisfaction.

5.1.6 Experiment design

The experiment had a simple design within subjects that means that each participant (subject) of the experiment had to use two techniques (treatments): the technique under evaluation (CrowdMock) and other well-known technique well known by the participant. In this situation, the Maturation Threat of internal validity (the order in which participants apply the technique) is very important to cope with. We had used levels of independent variable (treatment); thus, we had made two groups of participants, one group used CrowdMock first while the other used CrowdMock in second place in order to counterbalance the effect of the order in applying the approaches.

5.1.7 Procedure

The experiment was organized in two phases. In each phase, the participants specified some requirements, then a group of scripters implemented the functionality specified, and finally the participants validated whether the implementation satisfied their original request specified as requirements.

At the beginning of the experiment (before both phases), some of the authors of this article introduced the general procedure and trained the participants on the common tasks of both phases. This training was performed as a part of a postgraduate course. Some participants have used in the first phase a technique they already known, and others have used MockPlug. Participants have also received training in MockPlug previous to the use of the technique.

After that stage, participants had 3 days to write two requirements inspired and limited by the list of features we provided; requirements had to be written at home without any kind of assistance. In one phase, participants have to use a technique they already known to specify requirements, and in the other one, they have to use MockPlug. After these 3 days, they had to send their specifications to the development team. Developers had 2 days to implement the corresponding augmentations through GreaseMonkey scripts. It is important to note the distinction between participants of the experiment who only have to specify requirements and a specific team composed by experienced developers in Web Augmentation who implemented them. This distinction is important because the experiment had the goal to test the communication between both groups through the specification produced by different techniques. After finished, scripts were sent to the participant who originally specified it and he or she had 2 additional days for installing, using, and evaluating it. During this process, every participant had to fill in the aforementioned questionnaires.

5.1.8 Analysis procedure

Given that we have no guarantee about a normal distribution of the data, we have used the Wilcoxon sign test, a nonparametric statistical hypothesis test, to assess whether one of the two samples of paired observations tends to show differences [42]. We apply this test to the level of satisfaction declared by participants.

Table 1 Distribution of the features selected

Feature	% selected
F1.1	12
F1.2	22
F1.3	17
F2.1	25
F2.2	24

Table 2 Distribution of techniques used

Technique	% used
Narrative description	26
Narrative description + mock-up	24
Mock-up	39
User Story	11

5.2 Deviations from the plan

Since execution was carried out according to the procedure defined, in this section we describe some characteristics of the experiment development.

The first task to be performed by participants was the selection of the general features to write the requirements. All the features were selected randomly. Table 1 shows that the distribution of selection of the features was quite balanced near the average of 20 %.

Participants performed the tasks in the schedule previously described. The techniques used, apart from our proposed technique, were traditional mock-ups, traditional user stories, narrative description, and a combination of mock-ups with narrative descriptions. Table 2 shows the distribution of the techniques used.

Regarding the usability questionnaires, the difficulty was measured in a scale ranging from "Very Easy" to "Very difficult," having "Easy," "Normal," and "Difficult" as intermediate options. When specifying requirement with traditional techniques, the difficulty was mostly perceived as "Normal." "Very Difficult" rank was not used at all. The difficulty perceived using CrowdMock approach had more situations ranked as "Very Easy" and "Easy," but it also had more situation ranked as "Difficult" and "Very Difficult." It is important to mention that situations where CrowdMock was ranked more difficult than other approaches were related to technological issues. For example, some participants reported problems when installing the plug-in, and other participants had problems about internationalization (for instance, people from Sweden and Ecuador). Table 3 shows the complete distribution of difficulty perceived. It is important to mention that the average time of applying traditional techniques was 198 min, while the average time of applying CrowdMock was 104 min.

Table 3 Distribution of difficulty perceived when specifying requirements

Level	% for traditional approach	% for CrowdMock approach
Very easy	5	11
Easy	21	26
Normal	63	42
Difficult	11	16
Very difficult	0	5

Table 4 Distribution of satisfaction perceived when specifying requirements

Traditional approach	CrowdMock approach	Quantity
Does not satisfy	Does not satisfy	1
Does not satisfy	Partially satisfy	1
Does not satisfy	Satisfy	1
Partially satisfy	Does not satisfy	1
Partially satisfy	Partially satisfy	7
Partially satisfy	Satisfy	11
Satisfy	Does not satisfy	0
Satisfy	Partially satisfy	4
Satisfy	Satisfy	12

The development team implemented all the requirements and sent the scripts to every participant. Nevertheless, participants were able to download and install the script in 95 % of the cases.

The level of satisfaction reported by participants who were able to install and test the scripts is summarized in Table 4. It describes all the possible combination about level of satisfaction according to traditional and CrowdMock approaches and the quantity of times the situation was reported by participants. In some cases, the application of both approaches was related to a similar level of satisfaction. In other cases, traditional approaches were reported to have higher level of satisfaction than CrowdMock. And in other, CrowdMock was reported with higher level of satisfaction. Results show that the majority of times both approaches reported the same level of satisfaction: "Does not satisfy"/"Does not satisfy," "Partially satisfy"/"Partially satisfy," and "Satisfy"/" Satisfy" with 20 times. Nevertheless, CrowdMock reported a higher level of satisfaction than traditional approaches: "Does not satisfy"/" Partially Satisfy," "Does not satisfy"/" Satisfy", "Partially Satisfy"/"Satisfy" with 13 times.

5.3 Analysis

In order to apply Wilcoxon sign test, W value and Z value must be calculated. We used 38 pairs of data corresponding to the situations where participants were able to specify, install the script, and verify the functionality provided. Then, these 38 pairs of data, has 20 pairs, where the satisfaction reported was the same for both techniques; thus, the sample was reduced to a size of 18 ($N = 18$). In this situation, we use the W value to evaluate our hypothesis. W value is 45. The critical value of W for $N = 18$ at $p \leq 0.01$ is 32 [26]. Therefore, H_0 cannot be rejected at $p \leq 0.01$. But the critical value of W for $N = 18$ at $p \leq 0.05$ is 47 [26]. Therefore, H_0 can be rejected at $p \leq 0.05$ [42]. We also calculated the effect size. Since our experiment compares groups of independent values, we calculated the Cliff's delta correlation coefficient instead of the generally used Cohen coefficient [7]. We obtained a value of 0.209. This value determines an effect size between small and medium.

Summing up, there is a substantial difference ($p < 0.05$) between both approaches in terms of the satisfaction of the script produced using different approaches to specify them and H_0 can be rejected.

We focused the experiment on the capability of our proposed approach to express requirements. In particular, the experiment was focused on assessing the satisfaction achieved by experienced Web users. Thus, the participants had to use two different techniques: CrowdMock and other technique well known by them, and compare the results. This situation was adverse to CrowdMock, because people were comparing a well-known technique against a new one. And it is expected that the well-known technique would have a better performance that the new one. Nevertheless, CrowdMock obtained good results. The average time of applying CrowdMock was almost half of the average time of applying well-known techniques. Then, CrowdMock was ranked better than the other techniques in difficulty perceived (if technical issues are not considered). And in this scenario, the level of satisfaction perceived was better in CrowdMock than in other well-known techniques. Even more, we do not choose a specific technique to compare with CrowdMock. We let participants to choose a well-known technique by each of them.

Thus, although we believe that we have to continue with the experimentation for measuring other aspects about our approach, such as how well our propose traceability is supported, we think that results are very promising and have shown that this is good way to express Web augmentation requirements. We think that the fact that CrowdMock relies on prototypes (mock-ups) applied over the real Web Application is the key of the success of the experiment.

5.4 Threats to validity

Wohlin et al. [42] group validity threats into four categories: conclusion, internal, construct, and external validity. The following paragraphs analyze different threats from each category.

Concerning the conclusion category, one possible threat is to violate the assumptions of statistical tests. In order to avoid this threat, we have used Wilcoxon sign test which is a nonparametric alternative to paired t test. Another threat from the conclusion category is the reliability of measures. Our experiment consists in ranking the satisfaction perceived by participants in a scale of three possible values: "no satisfy," "partially satisfy," and "completely satisfy." This rank can be easily and unbiased identified by participants, because if the satisfaction is not null or complete, it is partial. Thus, this measure is the core to ensure that our experiment is not biased. The last threat from this category to discuss is random heterogeneity of subjects. There is always heterogeneity in a study group. If the group is very heterogeneous, there is a risk that the variation due to individual differences is larger than due to the treatment. In our experiment, all participants are homogenous since all of them have a degree diploma and have experience in industry. Nevertheless, participants are heterogeneous in relation to years of experience, roles, country, and context (academia vs. industry).

The second category of threats to analyze is internal validity. Selection is the main threat to internal validity. In order to mitigate the effect of this threat, subjects were able to choose by themselves features to specify requirements and a technique to contrast to CrowdMock. Instrumentation is another threat to internal validity that we were concerned to tackle. For that purpose, we paid a lot of attention to the preparation of artifacts for the experiment. We used real applications, and we also determine real features. Moreover, since nowadays homeworking is a common practice, the context in which the main tasks were performed should not derive in any threat. The maturation threat does not impose a problem because the experiment lasted little time and experimental task in particular was very short, participants had only to specify two requirements, test the implementation and complete short questionnaires. In this way, the subjects would not have to worry about being bored or tired from the experiment.

According to the construct validity category, we observed that the experiment did not suffer from such threats referred to as hypothesis guessing, evaluation apprehension, or experimenter expectancies, because the people only had to produce the requirements specification and then, they had to contrast their expectations to the scripts produced. Moreover, the team in charge of implemented the functionality did not know the experimental

situation; thus, they could not be biased trying to provide implementation with defect on purpose. Although subjects were students and they could have been biased by their position, they did not know which variable described in the questionnaires was going to be analyzed.

Sjøberg et al. [35] state that many threats to external validity are caused by an artificial setting of the experiment. They mention the importance of realistic tasks and realistic subjects. Realistic tasks are concerned with the size, complexity, and duration of the tasks involved. Taking this into account, we set up an experiment which had the complexity of a small but real situation. Realistic subjects are concerned with the selection of subjects to perform the experimental tasks. In order to tackle this threat, we selected practitioners with real experience in Web Engineering. They had a wide range of experience, as well as different skills.

In order to avoid biases, we have also paid special attention on designing the questionnaires by following well-known practices. First, all the vocabulary used in questionnaires was opportunely introduced and explained during the presentation of the experiment, in the courses' face-to-face meetings. Besides the specific vocabulary, all participants were Spanish speakers, and since all questionnaires were written in Spanish, there were not mistakes introduced by any translation. Regarding questions design, we have used a combination of both closed-ended and open-ended questions. The first ones were used because they allowed us to define a priori the numeric values for the response choices (which were selective). The second ones were used in order to allow participants to express, without any limitation, why or why not the scripts satisfied their requirements. These answers were analyzed a posteriori, and fortunately, these answers were in concordance with the scaled answers, but providing us some additional details.

Regarding the administration strategy, we chose the self-administered questionnaires. This was important because we reduce any biases that could appear when there exists the intervention of interviewers.

6 Related works

Web Augmentation is an activity carried out by end users. At the end, they are who know their needs and, as we have pointed out in this paper, only those users with the necessary programming skills are able to create artifacts with particular adaptation goals and share them within Web Augmentation communities. However, since not all people have these skills, many augmentation requirements are relegated until they are completely understood. As far as we know, and beyond informal forums for asking new augmentation artifacts provided by the communities, there are not scientific works tackling the problem of how to specify and manage this kind of requirements. Nevertheless, there exist are several important works that aim to raise the abstraction level for programming Web Augmentation artifacts in order to make broader the spectrum of augmentation developers. Most of these approaches about end-user Web Augmentation are defined in terms of a subset of possible adaptations or domains. For instance, in previous works, we aimed to support inter-application tasks by integrating mechanisms based on Web Augmentation tools, which allow users to specify (for instance) the integration and the augmentation desired, using a framework and a domain-specific language [16]. Others authors have proposed end-user programming languages for Web Augmentation, although requiring some high technical skills [12]. The same authors have defined the WebMakeUp [13] approach, comprising an end-user tool allowing users to perform several kinds of augmentations. Regardless of these very important contributions, we strongly believe that most of the popular Web Augmentation artifacts are not possible to be specified only using this kind of tools. Several of the scripts available in existing repositories have more than one thousand lines of code, containing complex logic.

Several of the examples presented on this paper cannot be specified without the intervention of someone with programming skills, especially in imperative programming. In this way, and even when the mentioned approaches are enough to satisfy several augmentation requirements, those cases such as Peter's one are out of scope of most end-user augmentation tools. Our approach aims to fill this gap considering that currently there is evidence of cooperation between users with and without programming skills in the existing communities. In this way, we are confident that our work is a novel approach for supporting Web Augmentation activities, because we provide end users with mechanisms to specify their requirements and, based on these specifications, end-user developers may get a first script code generated automatically.

Note that although our work is focused on Web Augmentation requirements, several aspects of our approach are related to both collaboration in requirements management and Web requirement specification. Azadegan et al. [1] and Dheepa et al. [9] propose approaches similar to ours. They involve many stakeholders to identify requirements, and then, they discuss and prioritize requirements. Nevertheless, these approaches are different from CrowdMock because they propose more rigid steps while in a collaborative context, a more flexible approach is more suitable. Moreover, they divide their approaches in two main steps: First, they build a stakeholder structure and after that, they work on requirements. This imposes even

more rigidity, because it prevents that a new stakeholder can be included in the process after this stakeholder structure was built, while our approach is more flexible because stakeholders can be incorporated in any moment during the definition of the requirement. In particular, stakeholders involved in our approach can play two roles: they can be requester of functionality and they can also be providers. These two roles agree with the roles proposed by Vuković et al. [38].

An important aspect of our approach is prioritization. We provide a simple mechanism to prioritize using the "like" voting technique. Lim et al. [23] rank using a scale from 0 to 5; nevertheless, "like" has proved to be more effective in order to select more important requirements [2]. Shimakage et al. [33] rank using a more complex mechanism as analytical hierarchy process (AHP). Although it provides better results, it is complex to apply and demands much effort.

We agree with Wu [43] in the importance of collaboration in order to obtain the diversity that it is needed to support the creativity process and obtain a good and representative set of requirements. In this process, we consider that iteration and refinement are very important, in contrast to Ponzanelli et al. [28] which ignores refinements at all. In this collaboration context, it is very important to use models that are well understood by all participants. Thus, we use User Stories (structured colloquial descriptions) and high-fidelity mock-ups (RAMs) which complement each other. This is possible since our metamodel allows to specify behavioral features like responses actions to events in added widgets. Shimakage et al. [33] propose a similar model to ours, but they use GUI descriptions while we use mock-ups. Using UI mock-ups as a requirement elicitation technique is a growing trend that can be appreciated just observing the amount of different Web and desktop-based prototyping tools that appeared during the last years like Balsamiq and Mockingbird.[20] Statistical studies have proven that mock-ups use can improve the whole development process in comparison with using other requirements artifacts [30]. MockPlug, our mock-up tool, is specifically designed for the Web Augmentation domain, achieving high-fidelity prototypes really close to the final implementation, since the involved widgets are already DOM elements. In comparison with the mentioned tools, we think that taking as a base the existing Web site facilitates the specification process, since users never start from scratch and also start from the real, original Web site to specify their augmentation requirements. The use of mock-ups has been introduced in different and heterogeneous approaches and methodologies. They have been included in traditional, code-based Agile settings as an essential requirement artifact [14, 37]. They have been used as formal specifications in different model-driven development approaches like MockupDD [31], ranging from interaction requirements to data-intensive Web Applications modeling. In this work, we propose to specify augmentation changes using (among other techniques) mock-up-style widgets. Also, MockPlug combines mock-up-style augmentation techniques with well-known annotation capabilities of common mock-up tooling that can be also used as formal specifications in the future as in [27].

Sutcliffe et al. [36] propose a similar approach to ours. Their approach considers iterative cycles where different stakeholders produce and refine abstract models and concrete representation (scenarios, sketches, storyboards). The difference is that they need a role of moderator. Other approaches not only add new roles, but they also include more complex analysis of stakeholders. Lim et al. [23] prioritize stakeholders using a variety of social network measures in order to consider the vote of different stakeholders in different ways. Reenadevi et al. [29] include a rating of malicious stakeholders in order to detect how they can affect the product quality. In fact, they propose an algorithm for identifying non-stakeholders. Since we rely on collaboration, our approach is based on the premise that all stakeholders have the same influence and they are working with no malice.

7 Conclusions and future work

Web Augmentation has emerged as a mechanism to improve the user experience while surfing the Web. It has been widely adopted by a crowd of users as it can be seen in the multitude of script repositories. Augmentation goals are broad since users' requirements are broad as well. Some of the current tools are used widely; for instance, AdBlock Plus![21] has been installed more than 300 million times, allowing users to alter Web pages with a very particular goal, which is basically to remove intrusive advertisement. Other more powerful Web Augmentation engines (Scriptish, Stylish, or GreaseMonkey) have also millions of installations and altogether propose more than one hundred thousand augmentation artifacts used by more than several thousand users each one. As we have mentioned before, these topics are also being tackled in different ways in the research world. Different scientific works are based on Web Augmentation for improving Web Applications accessibility [4, 18], support user tasks and concerns [16], Web forms filling improvements [15], etc. These works have shown the power of augmentation techniques when applied to specific domains, designing specific tools and

[20] http://gomockingbird.com.

[21] https://adblockplus.org.

making easier the development of augmentation artifacts in the context of a particular domain. Also others works have shown that end users without programming skills are able to create their own augmentations [13], although reducing the expressivity and consequently supporting only a subset of potential adaptations, or DSL for Web Augmentation [12].

Altogether, the mentioned works place Web Augmentation as a powerful mechanism for Web personalization and customization. Existing Web Augmentation communities have a similar pattern in their users' composition. There are artifacts creators, artifacts users, and artifacts requestors. However, the current communication channel between creators and requestors makes harder to understand what users (requestors) need.

In this work, we presented the CrowdMock, an approach whose main goal is to improve how Web Augmentation communities work. One of our main aims was to improve the communication between requestors and creators. The CrowdMock approach is fully supported by MockPlug, a tool for defining augmentation requirements via high-fidelity augmentation-based mock-ups, and UserRequirements, a platform for managing these requirements. The approach lets users collaborate in the refinement process and choose the better version to be implemented, administrating in this way the effort made by creators who (at least in these communities) usually enjoy collaborating with each other and with requestors. Our approach not only helps them in understanding what a requestor expects, but also it provides a way to automatically generate initial code for implementing the augmentation that satisfies its requirements.

We strongly believe that Web Augmentation artifacts are going to play even a more important role in the future. We also think that if we want to go further in the adoption of Web Augmentation as a personalization mechanism, we need to provide better support to those users who are not able to create their own artifacts, but still have adaptations requirement. CrowdMock and its supporting tools are a novel approach in this sense, which shows that it is possible to define and manage Web Augmentations requirements in a systematic way.

At the moment, we are currently working on the improvement of several CrowdMock aspects. First, we are improving automatic code generation. This is really a challenge since creators have different views when programming augmentation artifacts due, for example, to the big offer of JavaScript libraries they may prefer to use. This gives us the premise that it would be necessary to improve the customization of how the code artifacts are generated if we pretend that the approach has a high adoption rate. Experiments about how creators may adopt the generated code are also foreseen. Even more, we are working on the inclusion of acceptance tests when an artifact is published in response to a requirement; in this way, the requestor may provide feedback about the implementation.

References

1. Azadegan A, Cheng X, Niederman F, Yin G (2013) Collaborative requirements elicitation in facilitated collaboration: report from a case study. In: 2013 46th Hawaii international conference on system sciences (HICSS), pp 569–578
2. Bao J, Sakamoto Y, Nickerson JV (2011) Evaluating design solutions using crowds. In: Seventeenth americas conference on information systems, August 4th–7th, pp 2013–2015
3. Basili VR, Caldiera G, Rombach HD (1994) The goal question metric approach. Encycl Softw Eng 2(1994):528–532
4. Bigham J, Ladner R (2007) Accessmonkey: a collaborative scripting framework for web users and developers. In: Proceedings of international cross-disciplinary conference on web accessibility (W4A 2007), pp 25–34
5. Bouvin NO (1999) Unifying strategies for Web augmentation. In: Proceedings of the tenth ACM conference on hypertext and hypermedia: returning to our diverse roots: returning to our diverse roots, pp 91–100
6. Brusilovsky P (2001) Adaptive hypermedia. User Model User-Adap Inter 11:87–110
7. Cohen J (1988) Statistical power analysis for the behavioral sciences. 2nd edn. Routledge, London
8. Cohn M (2004) User stories applied: for agile software development. Addison-Wesley Professional, Boston
9. Dheepa V, Aravindhar DJ, Vijayalakshmi C (2013) A novel method for large scale requirement elicitation. Int J Eng Innov Technol (IJEIT) 2:375–379
10. Díaz O (2012) Understanding web augmentation. In: Grossniklaus M, Wimmer M (eds) Current trends in web engineering. Springer, Berlin, pp 79–80
11. Díaz O, Arellano C (2015) The augmented web: rationales, opportunities, and challenges on browser-side transcoding. ACM Trans Web (TWEB) 9(2):8
12. Díaz O, Arellano C, Azanza M (2013) A language for end-user web augmentation: caring for producers and consumers alike. ACM Transactions on the Web (TWEB) 7(2):9
13. Díaz O, Arellano C, Aldalur I, Medina H, Firmenich S (2014) End-user browser-side modification of web pages. In: Benatallah B, Bestavros A, Manolopoulos Y, Vakali A, Zhang Y (eds) Web information systems engineering–WISE 2014. Springer International Publishing, pp 293–307
14. Ferreira J, Noble J, Biddle R (2007) Agile development iterations and UI design. In: Eckstein J, Maurer F, Davies R, Melnik G, Pollice G, (eds) Agile conference (AGILE), 2007. pp 50–58
15. Firmenich S, Gaits V, Gordillo S, Rossi G, Winckler M (2012) Supporting users tasks with personal information management and web forms augmentation. In: Brambilla M, Tokuda T, Tolksdorf R (eds) Proceedings of international conference on web engineering. Springer, Berlin, pp 268–282
16. Firmenich S, Rossi G, Winckler M (2013) A domain specific language for orchestrating user tasks whilst navigation web sites. In: Daniel F, Dolog P, Li Q (eds) Proceedings of international conference on web engineering. Springer, Berlin Heidelberg, pp 224–232
17. Firmenich D, Firmenich S, Rivero JM, Antonelli L (2014) A platform for web augmentation requirements specification. In: Casteleyn S, Rossi G, Winckler M (eds) Proceedings of international conference on web engineering. Springer International Publishing, pp 1–20

18. Garrido A, Firmenich S, Rossi G, Grigera J, Medina-Medina N, Harari I (2013) Personalized web accessibility using client-side refactoring. Internet Computing, IEEE 17(4):58–66
19. Glinz M (2007) On non-functional requirements. In: Requirements engineering conference, 2007. RE'07. 15th IEEE international, pp 21–26
20. Kelly S, Tolvanen JP (2008) Domain-specific modeling: enabling full code generation. Wiley, New York
21. Ko A, Myers B, Rosson M, Rothermel G, Shaw M, Wiedenbeck S, Abraham R, Beckwith L, Blackwell A, Burnett M (2011) The state of the art in end-user software engineering. ACM Comput Surv (CSUR) 43(3):21
22. Lim SL, Quercia D, Finkelstein A (2010) StakeNet: using social networks to analyse the stakeholders of large-scale software projects. In: Proceedings of the 32nd ACM/IEEE international conference on software engineering, vol 1, pp 295–304
23. Lim SL, Damian D, Finkelstein A (2011) StakeSource2.0: using social networks of stakeholders to identify and prioritise requirements. In: 2011 33rd international conference on software engineering (ICSE), New York: IEEE Xplore, pp 1022–1024
24. Lucassen G, Dalpiaz F, van der Werf JME, Brinkkemper S (2015) Forging high-quality user stories: towards a discipline for agile requirements. In: 2015 IEEE 23rd international requirements engineering conference (RE), pp 126–135
25. Luna ER, Rossi G, Garrigós I (2011) WebSpec: a visual language for specifying interaction and navigation requirements in Web Applications. Requir Eng 16(4):297–321
26. McCornack RL (1965) Extended tables of the Wilcoxon matched pair signed rank statistic. J Am Stat Assoc 60(311):864–871
27. Mukasa KS, Kaindl H (2008) An integration of requirements and user interface specifications. In: Proceedings of the 2008 16th IEEE international requirements engineering conference. IEEE Computer Society, pp 327–328
28. Ponzanelli L, Bacchelli A, Lanza M (2013) Leveraging crowd knowledge for software comprehension and development. In: 2013 17th European conference on software maintenance and reengineering (CSMR), pp 57–66
29. Reenadevi R, Dugalya P (2012) Identifying malicious stakeholders using algorithm For large scale requirement-elicitation. Int J Comput Commun Technol 3(6, 7, 8):106–108 **(ISSN (Print): 0975–7449)**
30. Ricca F, Scanniello G, Torchiano M, Reggio G, Astesiano E (2010) On the effectiveness of screen mockups in requirements engineering: results from an internal replication. In: Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement, p 17
31. Rivero JM, Rossi G (2013) MockupDD: facilitating agile support for model-driven web engineering. In: Sheng Q, Kjeldskov J(eds) Current trends in web engineering. Springer International Publishing, pp 325–329
32. Rivero JM, Grigera J, Rossi G, Luna ER, Montero F, Gaedke M (2014) Mockup-driven development: providing agile support for model-driven web engineering. Inf Softw Technol 56(6):670–687
33. Shimakage M, Hazeyama A (2004) A requirement elicitation method in collaborative software development community. In: Product focused software process improvement. Springer, Berlin, pp 509–522
34. Shull F, Singer J, Sjøberg DIK (eds) (2008) Guide to advanced empirical software engineering. Springer, London
35. Sjøberg D, Anda B, Arisholm E, Dybå T, Jørgensen M, Karahasanovic A, Koren EF, Vokác M (2002) Conducting realistic experiments in software engineering. In: Proceedings of 2002 international symposium on empirical software engineering, 2002, pp 17–26
36. Sutcliffe A (2010) Collaborative requirements engineering: bridging the gulfs between worlds. In: Intentional perspectives on information systems engineering. Springer, Berlin, pp 355–376
37. Ton H (2007). A strategy for balancing business value and story size. In: Agile conference (AGILE), 2007, pp 279–284
38. Vuković M (2009) Crowdsourcing for enterprises. In: 2009 World conference on Services-I, pp 686–692
39. Walker M, Takayama L, Landay JA (2002) High-fidelity or low-fidelity, paper or computer? Choosing attributes when testing web prototypes. In: Proceedings of the human factors and ergonomics society annual meeting, vol 46, No. 5. SAGE Publications, pp 661–665
40. Whittle J, Hutchinson J, Rouncefield M (2014) The state of practice in model-driven engineering. Softw IEEE 31(3):79–85
41. Willighagen EL, O'Boyle NM, Gopalakrishnan H, Jiao D, Guha R, Steinbeck C, Wild DJ (2007) Userscripts for the life sciences. BMC Bioinform 8(1):487
42. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer Science & Business Media, Berlin
43. Wu W, Tsai WT, Li W (2013) Creative software crowdsourcing: from components and algorithm development to project concept formations. Int J Creat Comput 1(1):57–91
44. Platypus: https://addons.mozilla.org/es/firefox/addon/platypus/