Development Iterations Based on Web Augmentation and Context Tasks

Lucy Gutierrez Marticorena[1][0000−0002−4656−4624], Leonardo A. Morales[1,3], Leandro Antonelli[2], Gustavo Rossi[2], and Diego Firmenich[1]*

[1] DIT, Departamento de Informática Sede Trelew
Facultad de Ingeniería, UNPSJB, Chubut, Argentina.
*dafirmenich@ing.unp.edu.ar
[2] LIFIA; Facultad de Informática,
UNLP, La Plata, Bs. As. Argentina
[3] IPCSH; CONICET-CENPAT,
Puerto Madryn, Chubut, Argentina

Abstract. The use of prototypes in requirements engineering has widely known benefits since they actively involve the stakeholders in the development process. Web Augmentation techniques make it possible to build prototypes relying on existing web applications. Thus, high fidelity mockups can be quickly generated. One of the most critical activities is dividing requirements into tasks and managing them through the development process. This paper proposes an approach that includes high fidelity mockups into the Task-oriented Development approach. The proposed approach consists of the following steps: (i) end-users specifies requirements, (ii) a product owner verifies and prioritizes the requirements, (iii) tasks are defined and included in a kanban board, (iv) developers should provide the functionality, and (v) the product owner should approved the functionality. The main contribution of this approach is to integrate the requirements specified through web augmentation mockups, into the development environment via a task-oriented development approach. Thus, developers will have a rich context that facilitates the understanding of the requirements. At the same time, the management of the development process will have benefits because of the traceability between tasks and requirements. This paper describes the approach proposed, called "WAMRI", and an application of its usage, as well as a tool to support the application.

Keywords: Web Engineering · Web Augmentation · Requirements Engineering · Software Development Environments · Task Contexts · Software tools

1 Introduction

Software development approaches based on mockups and prototyping techniques have been used since decades ago. They were proven to be effective to communicate between engineers with end-users [37, 4]. More recently [41] and [48] have

shown how to involve end-users allowing them to define their needs using text and layout tools to visually specify requirements. Some model-driven approaches, such as *Mockup-driven development* [49], propose to evolve prototypes into models. However, from a methodological point of view, it is not really frequent and convenient [40].

Traditional techniques commonly used today to define requirements include Use Cases, User Stories, and Prototyping. The three techniques rely on defining requirements in artifacts isolated from the software application. Thus, from the analysis of the requirements to its implementation, there is a wide gap and this gap demands a huge effort to understand the requirement placing it within the application context, as well as to identify the source code related to the functionality.

The identification and limitation of the work context has been a matter of previous research. The approach called MyLar [29] allows the developers to focus on carrying out their labor within the boundaries of a work context. This approach evolved to MyLyn[46] that makes it possible to build and collect the work contexts related to the execution of some specific development tasks. Thus, the development environment can be adapted and prepared for allowing to the developer manipulate a limited set of elements, having as a consequence an increase in their productivity [30].

Moreover, requirements artifacts and the running software application follows two different threads, where both elements should be synchronized. That is, the requirements evolve, so the requirements artifacts should be updated to match the implementation. This task is very demanding, and it is even more time consuming to update mockups, since they have many details. Nevertheless, it makes no sense to update a mockup that represents the application GUI, when the application GUI can be itself used as a base for the mockup.

Web Augmentation techniques allow web users to work directly with the application, modifying the content (information displayed) and functionality of the existing web applications. Although Web Augmentation is primarily used for customization, these techniques with an adequate support of tools can be used to define new requirements [15]. Moreover, while describing the requirement the tools are also capable of identifying the modules of the application that need to be modified, providing to the developer some clues about the source code that needs to be modified.

Let's consider as an analogy civil engineer or architecture. It would be very useful to plan the extension or modification of a house at the same time that the inhabitants live in there. Thus, preferences and needs can be better assessed and described. In the virtual world of the application, the benefits are even more promising considering the great variety and quantity of stakeholders that usually exist behind an application deployed over the Internet. Even more, the augmentation techniques make it possible for different users to discover and define different versions of their same needs, and these can be prioritized and negotiated through adequate support of tools [16].

The contribution of this paper is providing an approach that involves the end-user in the definition of the requirements through the technique of Web Augmentation. The requirements are anchored to the real application that need to be modified and the proposed approach also provide clues about the modules (source code) that should be considered for modification.

The proposed approach is based on two fundamental pillars. One of them, is the definition of requirements using high-fidelity web mockups with augmentation techniques for their construction. Particularly, this paper proposes the use of a novel software artifact, called *Runnable Augmentation Mockup* [15]. The second pillar is the usage of augmented task contexts through the integration of our web augmentation mockups into the development environment. Although there are some previous proposal, our approach combine software artifacts with task contexts.

The rest of the paper is organized in the following way. Section Background and related works, provides a deeper description of the conceptual and technological background for understanding the proposed approach. Then, section WAMRI *(Web Augmentation Mockup Rapid Iteration)* describe the contribution of this paper, that is, an approach of rapid development increments in web applications based on mockups and augmented task contexts. Section four, describes examples of the proposed approach and tools to support the approach. Then, some discussion is provided. Finally, some conclusions and future work are presented.

2 Background: Web augmentation and Runnable Augmentation Mockups

To provide clarity and describe web augmentation [6] techniques in more depth, it could be mentioned that it is often said that augmentation is to the web of what augmented reality is to reality [12]. Because the end-user of web applications executes them in a browser on the client-side, there when the web apps are rendered, they have the opportunity to modify their appearance and behavior. Technically this can be accomplished in several ways, but the most widely used are *Browser Extensions* and *User Scripts*. About the former, these are generally obtained through Browser Extension markets [23] where users can find and install the extensions of their interest. These extensions are developed by third parties, generally, to work on the different web applications used by the end-user. Just to mention an example, the Dark Reader[11] extension apply dark themes overall visited websites or a few ones selected by the user. This extension has more than 3 million installations.

The execution of the extensions is managed by browsers, and technically they are capable of doing incredible things by altering, modifying, and combining content and functionality of the websites visited by the user. All this making use of the Web Extension API [8, 43] and manipulating the DOM Trees corresponding to the web applications used.

Fig. 1. Mockplug Web Extension: prototyping over an existing web application.

On the other hand, user scripts are software artifacts written in Javascript by end-users of the web, who build and share them through repositories forming different communities [25]. These user scripts are then executed using a user-scripts management extension [5, 24]. This extension is aware of the websites visited to insert these Javascript artifacts when the user visits the augmented ones.

Although these techniques are quite well known and popular, it is worth considering that they are not so much given their real potential and the immense number of web users [13]. In addition, it is important to mention that these artifacts are built and maintained naturally and totally decoupled from augmented websites and that drastically affects their survivability to augmented website changes.

In a previous work that we have called Crowdmock [16], we have used these augmentation techniques to allow end-users to elicit, define, and prioritize the augmentation requirements. Crowdmock provides an engineering approach in the construction and maintenance of user scripts. Part of the proposed process was based on the Mockplug [15] tool that allows end-users to start the process by defining the requirement on the augmented website itself.

Figure 1 shows a capture corresponding to a prototyping session using the Mockplug tool on a web application that we have developed for a particular domain related to a flora study on the Patagonia region [1]. It shows how the end-user dragged and dropped a representative image of a map on the application and added an annotation describing their need.

This new type of prototype, which we have called RAM (*Runnable Augmentation Mockup*), allows adding different kinds of elements such as text and form fields, buttons, lists, images, etc. Among other things, it also allows to select elements present in the UI and change their position or remove them. It also allows adding pocket widgets. That is, collecting elements in a pocket while browsing the web, and then mixing them with those existing in the system on which there is a need. In the case of the figure 1, this is what was done with the map element.

The tool also makes it easier for an advanced user to incorporate behavior into the elements. So that it not only allows generating visual representations of the requirement but also allows generating dynamic representations of it. Distinguishing this last possibility we have called them RAM.

These techniques not only facilitate the involvement of the end-user in their role of the stakeholder from the point of view of requirements engineering, but they have also allowed us to go further and provide tools that facilitate their participation in component testing and references maintenance of DEOIs (*DOM Elements of Interest*) [17]. Being these, two very important aspects for software artifacts built with augmentation techniques.

However, Crowdmock's objective is that the end-user community can get hold of the augmentation artifacts that allow them to satisfy their own customization needs, precisely those that the owners of the web applications did not originally provide. To do this, the end-user just begins by specifying and share their needs with RAM models through the Mockplug tool.

3 Related Works: Tasks and Context in Software Development

The context of a task is the set of interactions with tools and artifacts that the developer performs during its execution [36]. The same artifact (for example a user story, the source code, or a test specification), could be involved in more than one task and its relevance will surely be different in each case. There are two prominent approaches to measuring this relevance. On the one hand, the degree-of-interest (DOI) model takes into account the frequency and decay of interactions to continuously calculate the degree of interest of each artifact [31]. On the other hand, the FDA model measures the frequency and duration of each developer-artifact interaction, also considering the passage of time (or aging) of these interactions [36].

In the development of Software, understanding how the dynamics of interactions and artifacts work in the context of tasks allows the development of tools that facilitate the development process. Multiple tools have been proposed to free engineers from costly mental burdens (such as searching for artifacts or changing context) in order to increase their productivity and achieve higher quality products.

In order to attend to user productivity, and in relation to task planning, there are works such as [34] in which a real-time multiple workflow scheduling scheme is proposed to schedule workflows dynamically with a minimum cost and under different time constraints. Nevertheless the benefit of this approach it is necessary to organize the tasks in an acyclic directed graph structure in order to proceed. Another work [61] proposes an approach that uses the network encoder-decoder with short-term memory (LSTM) along with attention mechanism to generate predictions from activity logs and create schedules for allocating tasks. This proposal avoid the effort to manually build an acycic directed graph since it uses the activity logs. Nevertheless, [35] states that it is more effective the use of context than task-based integration.

There are multiple factors of context that can be taken into account in the work of Software Engineers and their interaction with the development tool or IDE. Some examples of these are: overall developer experience, IDE experience, complexity of the artifact in development, or even the day of the week the task is performed. In [20], thirteen of these context factors were selected, and a model was presented through which it has been possible to improve the precision and synchronization of the recommendations produced in the IDEs. Focusing on the so-called systems of recommendations for software engineering (RSSEs), [20] allows categorizing tools linked to the context of tasks in the development environment. Another group of tools focuses on offering relevant code examples, such as Strathcona [26] and Code Conjurer [27] (which, unlike Strathcona, does not require manual selection of the code snippet to perform the query and present a useful code snippet). We believe that recommendation is a valuable feature combined with code example, since it is the key in developer works: finding the source code to modify. Some other approaches focus on improving the use of the commands available in the IDE, such as SpyGlass [59], while others focus

on adapting the user interface presented by the IDE, such as Mylyn [31] and AutumnLeaves [54]. And other tools focus on adapting the ordering of the recommendations offered to the developer, such as OCompletion [50] and Quick Fix Scout [45]. We believe that it is not necessary to change the functionality or appearance of the view, since the developer is familiar with it.

The aforementioned Mylyn is one of the most popular tools and its objective is to help the developer focus on those artifacts that are most relevant to their coding tasks [28]. This plug-in, for the Eclipse [14] integrated development environment (IDE), implements the DOI model to filter Java classes, methods, and variables. Thus, this tool provides a kind of search tool to help the developer. Nevertheless, the developer needs to know what should he search for and if he does, he need to do the effort, while a recommendation is more efficient.

In more recent solutions an advance can be noticed in the treatment of the context of software engineers. [7] introduces Devy, a Conversational Developer Assistant (CDA) that enables developers to focus on their high-level development tasks. It is a voice-controlled prototype with a pre-defined set of automated Git and Github tasks. "CognIDE" [58], meanwhile, is a proposal that integrates psycho-physiological information from the developers (obtained through sensors) to the IDE. The data processed by this tool is presented both in the source code editor and in external visualization boards. In both cases, the proposals pursue the goal of helping developers to input the information to the tool. Nevertheless, in the first case, it is necessary to give some information orally while in the second case, the sensors also demands input from some explicit stimulus.

Previous cited works only considered source code as context, but MylynSDP [47] based on the aforementioned Mylyn tool considers some other phases (and artifacts) of software development, such as requirements and testing. MylynSDP extends the DOI function of Mylyn to work on three detected disadvantages: include artifacts from other phases of software development such as use cases and test cases; collect information on other characteristics of artifacts such as frequency or file type; third to prevent tasks from being manually defined. We believe that the inclusion of requirements is crucial since it provides a richer and higher level context that the source code.

Murphy [44] also proposed a new dimension regarding contexts since he focus on historical and activity information, which then serve as input for the creation of tools that actually increase human intelligence. This is a kind of machine learning solution that can be interesting approach in which the recommendations are adjusted according to the criteria finally adopted by the developers. We can find some examples of these tools in the work [38], such as chatbots (cognitive computing) in conjunction with recommendation systems (machine learning). The paradigm proposed in this work aims to take advantage of context information (implicit or explicit) to assist developers in real time and in the different scenarios in which they are working.

Considering the ways of building software today, "Agile Methodologies" are mature and widely adopted in the industry. These methodologies proposed a paradigm shift in order to increase the success rate of software projects. With

their practices, they seek to keep the client involved in all phases of development, deliver software early and continuously and adapt quickly to changes that may occur [19]. From this new way of working new artifacts emerged and gave rise to new interactions for software engineers in their day-to-day work. Therefore new contexts emerge as well. Work on [33] has shown that incorporating Continuous Integration and Continuous Delivery tools into the IDE has increased their use, and it has also improved the workflow for developers. The Agile Methodologies most used in the industry today are Extreme Programming, Kanban, and Scrum [52]. Some of them user combined [53]. In Agile Requirements Engineering, the most frequently used artifacts are User Stories, Prototypes, Use Cases, Scenarios, and Story Cards [57]. Thus, it is valuable to include artifacts related to agile methodologies in the context.

4 WAMRI

This section is organized in the following way. The first subsection describes WAMRI briefly. The second subsection describes general characteristics of the approach. The third subsection describes the steps of the approach. Finally, the last subsection conceptually describes the main components of the proposed aproach.

4.1 WAMRI In a Nutshell

From an engineering conception, our approach raises the possibility of including, within the development process of an existing web application, integrated tools that provide a framework for the elicitation and definition of new functional requirements. At the same time, it takes advantage of the prototyping information, at the precise moment and place in which the developers carry out the tasks corresponding to the resolution of the requirements.

According to the proposed approach, it is the end-users who, through the use of the web application and based on their own experiences and needs, are in charge of laying out their functional requirements. This is achieved by enriching the web application by adding web augmentation capabilities to define their requirements. These web augmentation capabilities, let end-users add artifacts over the existing web applications that are essentially high-fidelity mockups and can be used later to provide the development environment with new kinds of information about the user requirements. Thus, new ways of reverse engineering in the construction and maintenance of web applications are possible.

In this way and as can be seen in the diagram of figure 2, the end-user as stakeholders that are interested in new functionalities, use the web application accompanied by special new tools for the definition of requirements, using prototypes via web augmentation techniques. The information obtained by the augmentation artifacts nourishes the development process, and developers through the development environment, visualize information of interest linked to the tasks to be carried out to satisfy these augmented requirements.

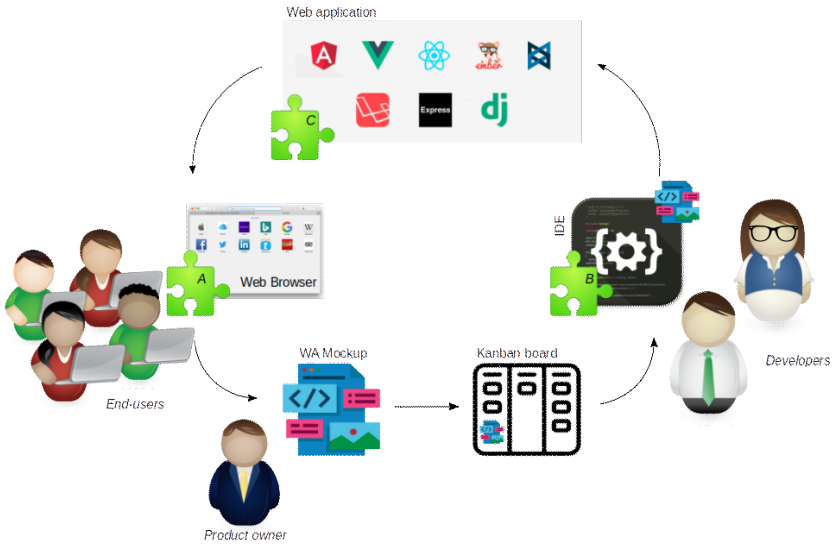

Fig. 2. WAMRI General Diagram.

The diagram depicted in figure 2 includes the following steps: (i) end-users specify requirements, (ii) a product owner verifies and prioritizes the requirements specified by the end-users, (iii) tasks are defined and included in a kanban board, (iv) developers code the functionality to satisfy the requirements which (v) the product owner should test to be approved. Thus, three roles are involved in the method: end-user, product owner, and developer. End-user is the role in charge of specifying requirements. The product owner is the role in charge of prioritizing the requirements, identifying and solving conflicts, and validating functionality after the developer implements the requirements. The developer is the role in charge of analyzing the requirements and implementing them.

In the same way, figure 3 allows us to distinguish the three main software components that support the approach: an extension for the browser **(A)**, an extension for the development environment **(B)**, and an extension for the web application **(C)**. These components are also indicated in the general diagram in figure 2.

Thus, the extension of the browser is used by end-users to specify requirements and the product owner to verify them. Then, the extensions make use of the Kanban Tool Support to manage the creating and updating of the tasks in the board. Finally, the combination of the web server extension and the IDE provides the information about the files needed to be edited by the developers.

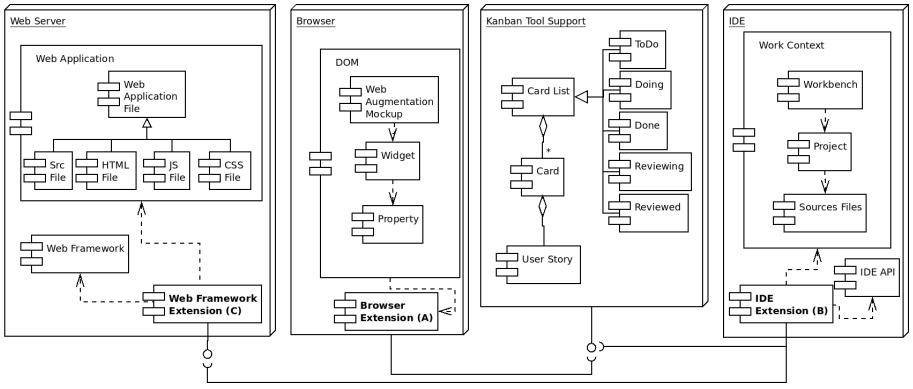

Fig. 3. WAMRI Components Diagram.

4.2 General characteristics of the approach

Foremost, it is important to emphasize that the method applies to developing new features of an application already developed and deployed in production. The running software application is the base for specifying new requirements. The key product to specify requirements is the web augmentation mockup. It is a mockup described directly on the software application to be extended in its functionality. That is in contrast to the traditional mockups designs approaches and tools, where the mockup artifacts are described in extern documents. The webmockup is also enriched with a User Story. Both elements are used to describe every requirement. Webmockups are runnable because it can contain widgets actions that triggers some functionality. Requirements are used as a mean of communication between end-users and products owners. Every requirement is available for every end-user, thus they can collaborate in the definition. This evolution creates a tree where end-user can also vote to express their agreement with the requirements. Finally, the product owner has the responsibility to define the final version of the requirement [16].

After the requirement is accepted by the product owner, it is included in a kanban board, which is a communication means between product owner and developers. The board includes the following columns: (i) To Do, with the task to be performed, (ii) Doing, with the tasks picked up by developed, (iii) Done, with the finished tasks that need to be validated by the product owner, (iv) Reviewing, with the tasks being validated and (v) Reviewed, with the tasks finally validated by the product owner. It is important to mention that the board is automatically updated according to the actions performed by the roles. That is, in order to have a task in ToDo, the product owner should have accepted a requirement.

The use of the software application to describe the requirement makes it possible to relate the requirement with the files needed to modify in order to satisfy the requirement. This is an advantage of the approach. There is a relationship between the requirement and the files to be edited. Another advantage is that

there is no need to learn to use a tool to specify requirements, and no need to change from the tool to specify requirements and the running software.

The method proposed is based on an agile life cycle, since the documentation is simple and light. Moreover, it is iterative and incremental, since end-users request new functionality, it is developed and deployed, and end-users can request new functionality. Finally, it is based on an evolutionary prototype. The literature considers an evolutionary prototype when the prototype is improved and finally is transformed into the final application. In this case, it is even more extreme, since the application also contains the requirement.

4.3 Steps of the approach

This section describes every one of the steps of the proposed approach. The first step is the requirements specification. In this step, a group of end-users specifies requirements. The product owner does not elicit requirements from them. Requirements are specified directly in the web application using augmentation. Thus, the implementation begins in some way with the specification, since a new widget can be placed in the real position of the application and the developer will have to implement the logic and functionality needed. It is important to mention that requirements not only consider adding widgets, it also consider removing, moving, combining, etc...

The second step is the requirements verification and prioritization stage where the product owners review every requirement looking for requirements repeated or in conflict. If there are two repeated requirements, the product owner should accept only one. By contrast, if there are two requirements in conflict, the product owner decides (in combination with the end users) the specification. For example, it could occur a conflict when two end users request a delete functionality to work in different ways. One end user requests a delete function to remove one element, while another end user requests a delete function to remove the element chosen, performing delete in cascade with the related elements. In this situation the product owner decides, in discussion with the end user involved if necessary, the best approach. It is important to mention that this discussion can be carried out in situations where there is no huge amount of end user. For example, in a global platform like social medias is not possible, but in a back office of a bank could be. Finally, the product owner accepts a requirement and prioritize it.

The third step is the definition of tasks stage where every requirement accepted by the product owner is translated to an individual task in the kanban board positioned according to the prioritization. Files related to the task are traced by the tool that supports this approach. Thus, the task is described using three elements: (i) a webmockup defined in a real application, (ii) a User Story described by the end-user, and (iii) the files needed to be edited which are identified by the tool proposed in this approach.

The last two steps are coding and user acceptance tests. In coding stage, the developer chooses a task, then the IDE displays a list of the files needed to be edited. Thus, the developer does not need to search for them, he can focus on

write the source code. When he finishes the task, the product owner performs the last stage: user acceptance test, where he checks whether the functionality satisfies or not the requirements. If it not, the product owner restarts the iteration.

4.4 Conceptualization of principal components

As mentioned, the fundamental components that support the approach are the web augmentation mockups for the definition of the requirement (Component A), as well as the extension of the functionality of the existing web framework, to be able to inspect the source code files of the deployed web application to be modified (Component C).

As described above, these two artifacts interact mediated by the development environment extension (Component B). Components A and C, are the key pieces that support the approach since the first one contains the requirement information provided by the end-user, and the second one, can exploit this information on the web server that deploys the application where the requirement was created.

Concerning Component A, it can be seen in figure 4, the metamodel of a web augmentation mockup. Each RAM model is made up of the following: the URL where it was created, a set of widgets, their properties, their insertion strategies, actions, etc. Thus, the models created by end-users in their augmentation requirements are represented by particular instances of this metamodel.

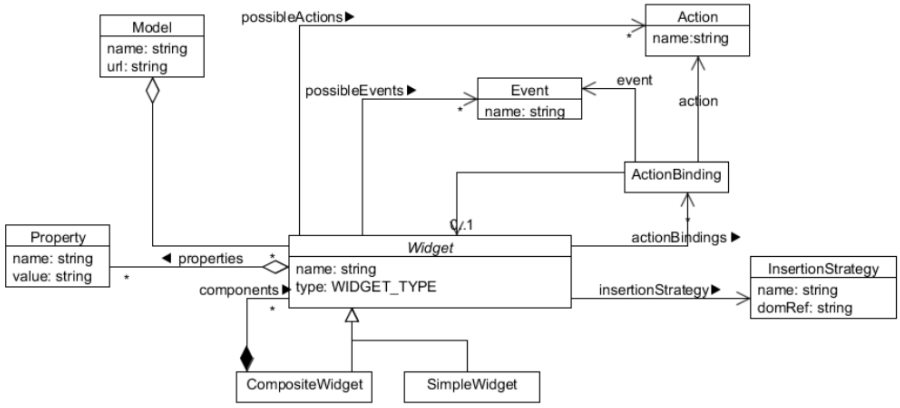

Fig. 4. Web augmentation mockups metamodel.

As for the C artifact to extend the functionality of the web framework, this is conceptualized as a web service. Requested, given a specific instance of the previous metamodel, it will be capable of responding with a list of source files and lines of code of potential interest to resolve the requirement.

This service, attached to the services offered by the web server that deploys the application on which the requirement was created, may, depending on the technology used, be implemented in many ways. However, conceptually, an example compatible with any Web Application Framework based on the well-known and popular architectural pattern Model View Controller (MVC) is next described in more detail.

Most popular web frameworks are based on the MVC pattern and its derivatives (MVT, MVP, MVVM), and they achieve the separation of concerns by dividing the application into distinct types of components and source files. Thus, based on the component's nature, there are component types and files for the business model and backend (application domain logic), there are component types for view files and templates (user interface logic), and finally, there are component types for controllers and files where the application events and inputs logic are located.

Therefore, a requirement specified by a specific web augmentation model, over an MVC Framework based application, will be related to a set of views, models, and controllers. These will be defined in certain files, and within those files, very commonly composed of hundreds of lines of code, only some snippets will be relevant to solve the requirement.

Figure 5 shows an algorithm for obtaining as an outcome these files and these lines of relevant code for a particular web augmentation mockup model, created on an application deployed in a Web MVC Framework.

```
component_c (Mockplug_Model model) : Files_Of_Interest
{
        files_of_interest <- {}
        files_of_interest.controller <- match (framework.controllers,model.URL)
        for each model_name in controller.snippet {
                files_of_interest.models.append (find (framework.models,model_name))
        }
        views_files <- filter(controller.snippet,VIEWS_FILE_EXTENSION)
        controller_views<- match(framework.views,views_files)
        for each view in controller_views {
                for each widget in model.widgets {
                        widget_lines <- find_compatible_widget_lines(view.HTM_text, widget)
                        files_of_interest.widgets.append(view, widget, widget_lines)
                }
                files_of_interest.views.append(view)
        }
        return files_of_interest
}
```

Fig. 5. Component C generic algorithm for a MCV Framework.

It can be seen that the algorithm can meet its objectives in a polynomial time $O(n^3)$. At a lower level of abstraction, foreseen in any implementation of this

algorithm, it will be probably necessary the use of regular expressions, XML or configuration files processing to find the controllers corresponding to the URLs of the mockups, as well as to lean on the characteristics of each framework and its technology, and its libraries, to programmatically browse the directories and the text of its view and model files. In the generic algorithm of figure 5 that operations were abstracted with the pseudo primitives match and find.

Finally, it will also be necessary performing widget definition searches based on their HTML Text properties. Like in the case of sub-algorithm of figure 6, which looks for widgets in an HTML view file. And for that, it is important to highlight that the widgets in the web augmentation mockup, may have been pre-existing in the application, or may have been added from the default tool palette or from any other application apart on the entire web. In either case, as is expressed in the metamodel, widgets are inserted into the augmented application using an Insertion Strategy. There, an existing reference to the DOM Tree relative application host element is used, which is the one used to search for the corresponding widget code definition lines (widget.insertionStrategy.domRef).

```
find_compatible_widget_lines (HTML_TEXT html, Mockplug_Widget widget ) : Line_Number[]
{
        compatible_widget_lines <- []
        target_html_node_properties <- ["ID", "NAME", "CLASS"]
        relative_widget_html_node <- relative_widget(widget.insertionStrategy.domRef)
        for each property in target_html_node_properties {
                property_value <- relative_widget_html_node[property].value
                lines <- match( html, relative_widget_html_node.TYPE, property, property_value)
                compatible_widget_lines.append (lines)
        }
        return compatible_widget_lines
}
```

Fig. 6. Component C sub-algorithm for find widgets relative lines in views.

5 Examples and Tool Support

The examples in this section are of an intranet web application used by biology scientists to record, visualize, and manage data about the flora and its distribution in the territory. These scientists use mobile applications to conduct flora recognition campaigns, then sync their devices with the intranet web application when they return to their labs. Four developers make up the software development team, two of them focus on mobile applications and the other two on web applications. The product owner role is played by the leader of the development team. Showing our contribution from the use of web augmentation mockups to the software development process, let's think about a possible scenario to present the tools in that context.

5.1 The end-users define their needs

Rose, a biologist, has already registered 40,000 specimens from 377 different species using the mobile app. She has collected dozens of samples from these specimens, and she is now in her lab, using her stereoscopic magnifying glass to identify the species that she has not been able to recognize with her naked eye in the territory. Once the species' scientific name has been determined, it must be registered using the web application, associating it with the synonym used during the exploration campaign with the mobile app. Figure 7 shows how Rose accesses the form that she uses to record the scientific names from the list of different species.

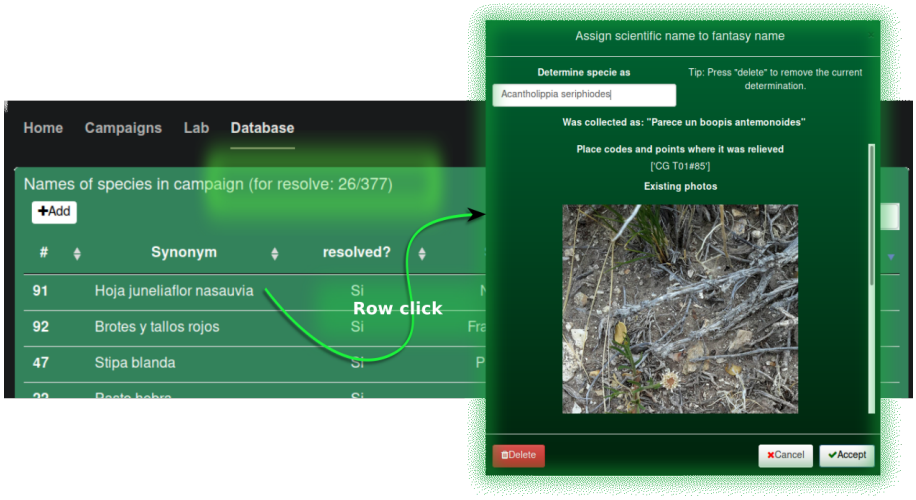

Fig. 7. Intranet web application.

While Rose is working in the lab, the web app displays a legend that shows how many fantasy names do not yet have a scientific name assigned (in figure 7, this number is 26/377). However, she has no way of knowing how representative these species are in comparison to the entire sample, and she needs to know how many of the 40,000 specimens sampled already have a scientific name and how many do not.

Then, using the same browsing session and the Mockplug tool, Rose defines a new requirement to communicate her need to the IT team that maintains the application. Figure 8 depicts the definition of this simple requirement, in which the user indicates that she needs to add an element of information about the data she is working on.

Rose is unaware, but this method of specifying her need allows us to incorporate her requirement into the DOM Tree during the user's browsing session. Figure 9 depicts the Mockplug tool's modification of the DOM Tree. The el-

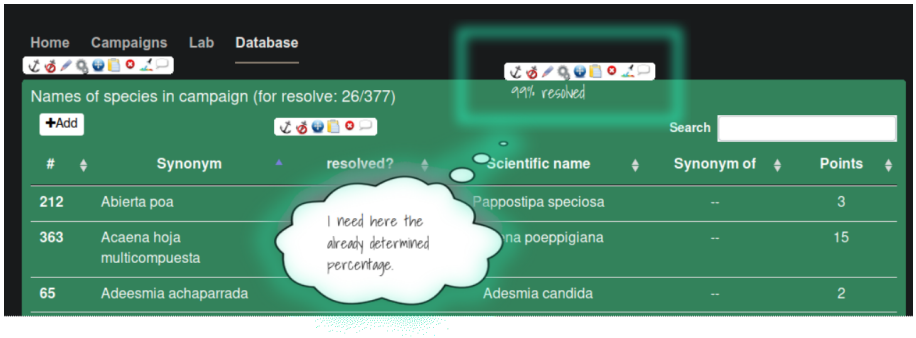

Fig. 8. Rose requirement.

ements corresponding to the requirement were added there. Furthermore, as shown in figure 8, existing DOM Tree elements relevant to the requirement were converted into requirement elements as well, such as the container div element for the data of interest.

Once Rose finishes defining her needs in terms of prototyping, she reports them by completing a simple form in which she adds a textual description of them using well-known user stories.

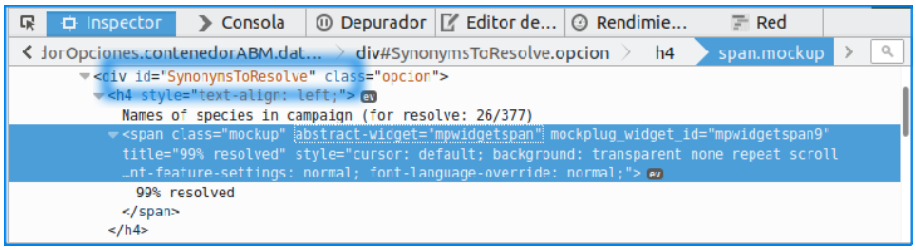

Fig. 9. Augmented DOM Tree in Browser Inspector.

5.2 The product owner verify and prioritize the requirement

User stories are automatically assigned to a specific list on the Kanban board of the product owner. There he can see the requirement's textual description as well as a screenshot of the mockup. As shown in Figure 10, from his board, the product owner will be able to verify and prioritize the resolution of the requirements.

It's worth noting that the product owner can resume the mockup created by Rose at this point in the process by clicking on the Kanban card details. This would allow him to evolve or adjust the required Mockplug mockup as he sees

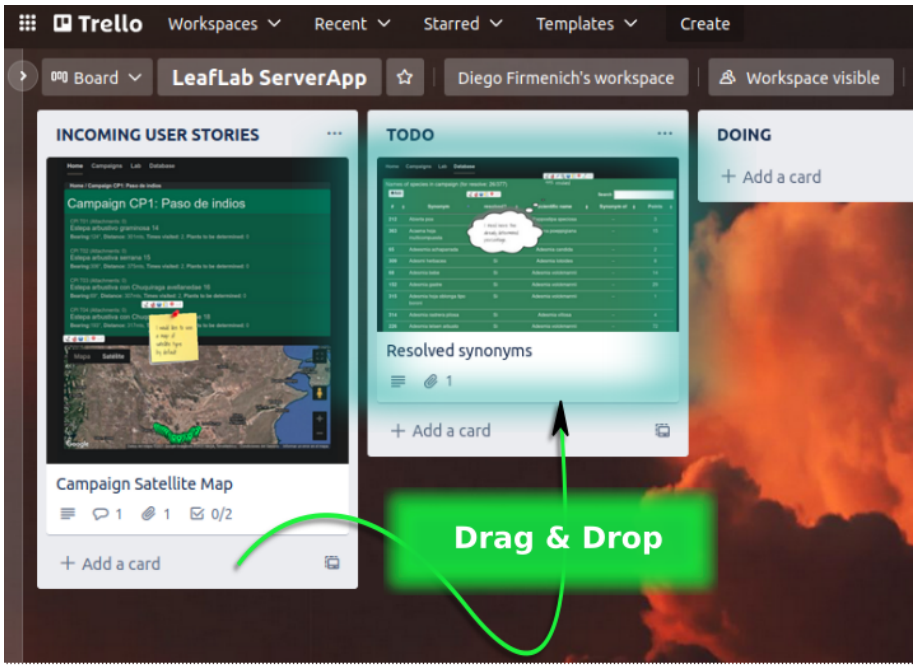

Fig. 10. Product owner Kanban Board in Trello.

fit using his own browsing session. In this case, the Rose need was validated and then accepted by simply dragging and dropping it onto the IT Team board's TODO list. At that time, the web developers are notified in their development environment.

5.3 Augmenting the context of the developer task

On the developer's side, all the previously described information becomes really important: the resulting DOM tree on the original Rose browser session, all the data about her story, including annotations, widgets, and their respective properties, and the URL where the requirement was prototyped.

As part of it work, it has been developed an experimental extension for the Visual Studio Code environment [39] to support our approach from the developer's point of view. Figure 11 shows a development session where the developer, through the main menu of the extension, has started solving the requirement mentioned above, which is shown to the developer on his right.

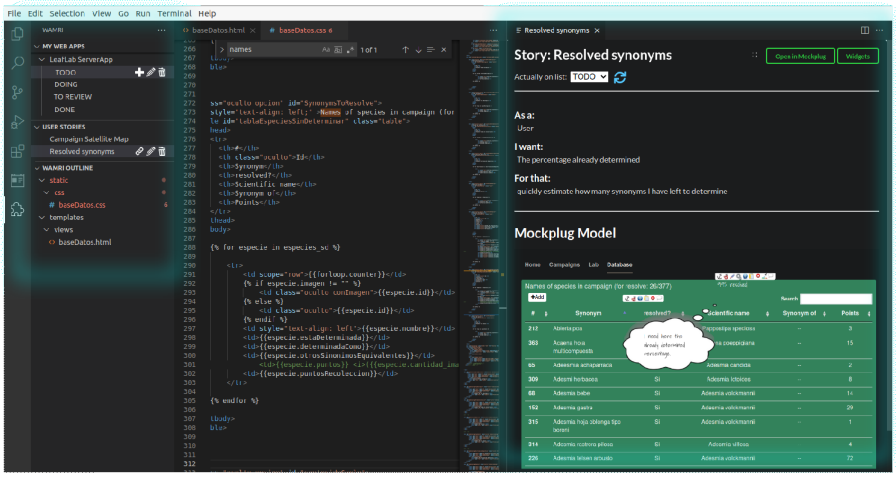

Fig. 11. WAMRI Tool Integrated on Visual Studio IDE.

In the rapid development iterations approach proposed, it was found important to think about the development contexts, as augmented contexts of development tasks. It can be seen in figure 12, how the extension is based on the information of the selected requirement to infer, through its WAMRI Outline component, which files probably need to be modified. This association allows the developer to have at hand the file(s) involved in each of the tasks that he must face when he selects the task on which he is going to work. Even more, as can be seen in the Figure 12, the developer has opened one of the files to edit it, and the cursor was automatically positioned where the HTML file has an element

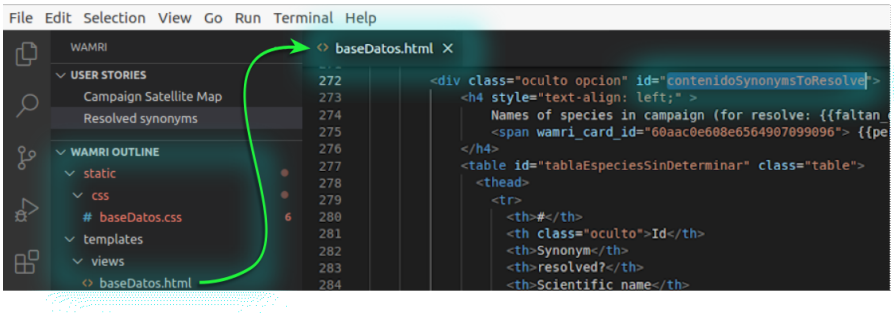

Fig. 12. WAMRI Task Outline.

of id "SynonymsToResolve". This action could carry out due to the information obtained from the augmented web mockup (relevant part of the corresponding DOM tree is showing in Figure 9). There, this particular instance of the web augmentation metamodel, Component A, were used by the extension to request the related components to the Component C, the web service extension.

On the developer side all requirement data can be conveniently used to support the developer work and carry out his task with new suggestions that now can be automatically done by the tooling. For example, figure 13 shows how through the contextual editing menu, the developer can use the tool options to incorporate the base code of the widgets present in the web augmentation mockup.

Finally, when the requirement is resolved, developers release a new version of the application for review and, from their developer environment, move the card to the TO-REVIEW list on the kanban board so it can be noticed and reviewed by the product owner.

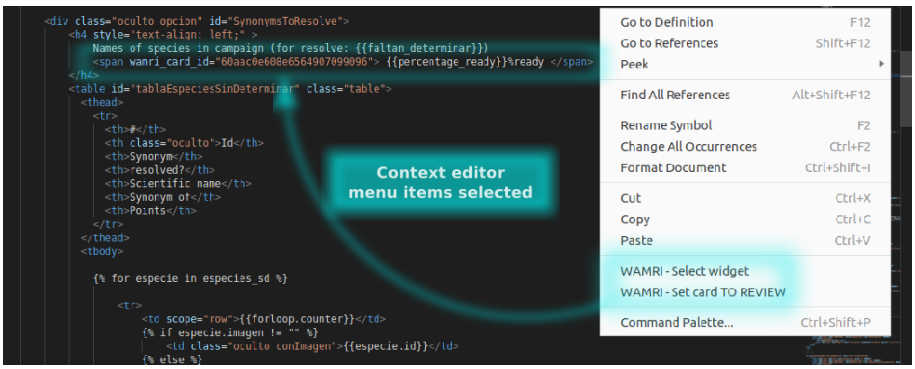

Fig. 13. WAMRI Commands in the Context Editor Menu.

5.4 Behind the scenes

Through Component A, all the information captured when the end-user was prototyping his requirement was collected from some part of the web application and contains the end-user annotations, the widgets with their properties, a snapshot of the browser tab that the end-user has worked on, as well as other complementary information about the application part where the requirement was prototyped. That is, information about the AJAX requests that may have occurred, the URL of the CSS files and rules of the elements selected, the Javascript files involved, etc.

On Component A we have carried out experiments with very positive results[15, 16] through its MockPlug implementation. In these experiments, participants specified their requirements through web augmentation mockups, and through traditional requirements specification techniques. As result, participants reported different levels of perceived difficulty. Through Component A, although the perceived difficulty was considered mostly normal, it was considered between easy and very easy many more times than using traditional techniques. Participants also reported that using this component, they spent half the time specifying their requirements compared to traditional techniques.

While the end-user finishes defining his requirement with these facilities, all the information of the web augmentation mockup, as well as the information about the AJAX requests that may have occurred, the URL of the CSS files and rules of the elements selected, the Javascript files involved, etc. All of that information is attached to the user story in the Kanban board using Trello's REST API [3].

Using this Trello REST API with the appropriate security keys from the development environment, the pending resolution user stories are listed, and with all the previously described information, a small but very important piece of the approach comes into play, which is the extension of the web framework to support our approach (component C in Figure 3).

This extension of the web application framework takes information from the requirement and returns information from the software project probably related to it: source files, component names, lines of code where they are defined, HTML templates, etc.

Over this component, it have been carried out a series of tests into two scenarios: in the previously described one, on the Django Framework, and with another web application that we have developed on the Flask Framework [42]. The data obtained from these test is published at FigShare [18].

These tests consisted of verifying that the suggested components of the application corresponded to those that should have been for different requirements laid out on the web apps. Based on the collected information, it can be stated that it was always possible to find the main source files as well as the lines of code where the main functions and methods are defined, which are the functions where the framework inversion of control happens. Also was always possible to find the HTML templates that the developer should modify to resolve the requirement. This is resumed in Table 1.

Table 1. Component C test results in Django an Flask Frameworks.

Property/Framework	Django	Flask
Found source file	100 % of times	100 % of times
Found line in source file	100 % of times	100 % of times
Found all HTML template files	100 % of times	100 % of times
Found all widget lines	40 % of times	50 % of times
Found all widget container lines	95 % of times	100 % of times
Found used domain classes	100 % of times	No applicable

It wasn't always possible to figure out which lines of the HTML template corresponded to the defined widgets in the mockup. This is because widgets are frequently dynamically generated on the DOM Tree in the user browser and are not textually present in the HTML template file. However, in most cases, it was possible to find a container for the widgets, which did make it possible to position the developer between the most relevant lines of the template to resolve the requirement.

5.5 The example in contrast

A summary of how the approach has affected the actors involved in the example scenario concludes this section. Table 2 shows which activities were done in a new way using the approach and associated tools.

Table 2. Activities in the example affected by the approach.

Activity	Actor	Traditional Agile Approach	WAMRI Approach
Requirement definition	End-user	User story, textual description, traditional mockup	Mockplug mockup
Requirement communication	End-user	E-mail	Mockplug User story form
Requirement elicitation	Product-owner	E-mail	Trello
Requirement planification	Product-owner	Trello	Trello and VS Code
Find files to be modified	Developer	Requirement and source code analysis	VS Code, WAMRI extension
Find lines to be modified	Developer	Source code analysis	VS Code, WAMRI extension
Find component implicated	Developer	Explore Source code project	VS Code, WAMRI extension

It stands out that the end-user never had to change her attention from the application she was using to specify her needs. In addition, she communicated

without knowing it with all the IT team involved, without having to send emails to any of them, and without having to verbatim describe any part of the application. The product owner did not have to interpret loose emails. Finally, when given the task to developers, they did not have to resort to their memory to know which files were related to that part of the application source code, nor did they have to perform text searches to find parts of the most important files related to the part of the system that should be modified.

6 Discussion

As it have been previously exposed, this work has two clearly defined fundamental pillars: the definition of requirements and their integration into the development environment through task contexts.

Concerning the use of RAMs, it is important to distinguish that the proposed approach deals with functional requirements in web applications that are already deployed since, the use of these web mockups is built over the application itself. This sets it apart from approaches like *Mockup-driven development* proposed in [49] where the prototyping work needs to be done from scratch using traditional mockup tools. We strongly believe that our new type of mockups make it easier to use these visual forms for the definition of requirements from the point of view of communication with the stakeholders, in stages in which modeling in methodological terms is much more expensive to carry out and to implement keeping aligned with the evolution of the applications [60, 40].

On the other hand, in contrast to end-user approaches such as Crowdmock [16] the main difference is that in those the end-users solve their objectives without the intervention of the developers of the web applications. In the present work, what is intended is that stakeholders have the possibility of communicating with the developers in a visual, agile, traceable way and from which it is possible to take real advantage by being able to intervene in the own development environment where the developer performs his tasks.

The main differences between MDD, Crowdmock, and WAMRI are summarized in Table 3.

Other authors [10] have observed that during a working day, the developer makes modifications to the source code that are related to more than one particular task, and proposes the automatic detection [10, 9] of the task that are working on. Attending the same problem approaches like [51] starts from a set of source components of interest and then enriches the task contexts automatically from the software project structure using its hierarchy. In the case of approaches like [2], the contexts are enriched from textual descriptions of the task, focusing on the fact that developers often use representative words for the names of classes and methods in the source code.

Instead, our approach proposes creating the tasks contexts from the definition of the requirement in visual as well as textual terms, and through these particular kinds of mockup indicate to the developer in his own work environment a few clicks away which files he will have to probably modify. Being that

Table 3. Main difference between approach's.

Feature/Approach	MDD	CrowdMock	WAMRI
Requirement defined by	End user, Developer	End user	End user
Requirement artifact	Traditional mockup, SUI Model, Tags	Runnable augmentation Mockup	Runnable augmentation Mockup
Prioritized and validated by	End user, Developer	User community	Product owner
Implemented by	Development team	End user	Development team
Applicable to	Web applications from scratch	Existing web applications	Existing web applications
Theoretical pillars	Model driven, Agile	Crowdsourcing, Web Augmentation, End User Programming	Task focused programming, Web augmentation, Agile

possible because the stakeholder has selected parts of the system when making the mockup. We believe that this can be a great contribution. Since, in traditional task-oriented approaches, when these tasks originate, there is no precise information on which files to work on. Either the task context is inferred as in the mentioned approaches, or the developer generates the context while he does progress in the resolution of the requirement by opening, modifying, and closing files of the development project in question. The recent bibliography highlights this disadvantage of manual task creation (on which the context is to be recorded) as an issue to be resolved [47].

There are numerous approaches concerning the problem raised that account for its relevance. Developers perform many different tasks per day, generating many task context switches[22]. In each of these context changes, the developer usually must read the documentation, the emails, the parts of the web, and the systems themselves[32]. There are recent studies[56] on the effects of high levels of developer stress on the quality of the products developed and how the aforementioned task-oriented approaches and others for time dimension administration such as the Pomodoro technique [55, 21], help to combat such effects.

In [44] Murphy describes a future and hypothetical scenario in which a developer, *Jessica*, is assisted in solving a task: An assistant prioritizes the requirements that were assigned to him by e-mail. After asking for the next action to be carried out, the assistant is in charge of loading the necessary environment to start coding. After solving the issue, the tasks of verification, measurement of results, and deployment of the new feature are carried out by the assistant. This way then allows this hypothetical developer to work without ever losing the central focus of her assigned task. Thus, by combining tools and recording interactions, it is possible to work on the context as a first-class element. Now, our approach can be thought of as an approximation to this ideal case enunciated,

taking advantage of the power of the augmented mockups. In our case, with the augmented mockup accompanied by a user story, the developer receives a new request in her IDE since the request has been added to their dashboard. The developer, having a notion of how to respond to the change, tells the IDE that she wants to work on the change request. The IDE loads the user story workspace, the components involved and leaves the cursor on the piece of software involved in the requested change. *Jessica* at this point never lost focus on the main task.

7 Conclusions and Future Work

The web is an incredibly massive and dynamic software platform. Year after year from its standards new functionality is incorporated, and today, it is really very difficult to imagine a future where the web may not be relevant. And, if that future existed, after more than 20 years of global level evolution, the legacy would be immeasurable. It is not at all risky to say that we will live using and evolving the web for many more years. Even more, new generations of developers with skills increasingly different from those of precursors will have to deal with an immense legacy. We believe that developer tools play a very important role in this regard and this work presents a new way of incorporate prototypes into the development process of existing web applications. This facilitate and speed up the exploring of the source files related to a requirement in a task-focused development way.

In technical terms, web applications are usually developed over some framework for the development of these kind of software and, about these type of frameworks, there is also a great variety and they are technically very diverse. Different programming languages are used and different architecture patterns such as MVC, MVT, MVP, MVVM, are adopted. And, as if the diversity is not enough, the most modern to be more comprehensive, use their own palette of components rather than simple HTML elements.

So in modern web front-end frameworks, we have identified situations where the ability to refer to a part of the code according to the intervened DOM element needs a little more attention. In these frameworks, a conversion of the code that the developer writes is carried out to generate the corresponding HTML, CSS and Javascript files. This allows developers to generate separate units, usually called components, that contain the graphical interface design plus its necessary logic. These components are then combined to integrate the entire application. This environment opens the doors to the challenge of developing an interpreter that allows dialogue between the HTML generated by the framework and the corresponding component highlighted here, in the context of the approach presented.

The proposed approach is fundamentally based on the integration of development tools. Some of the types of tools involved are well known and conceptually validated as is the case on tools for task-focused development. In the case of MockPlug that is used for the definition of the web augmentation mockups, we have carried out several experiments with very encouraging results. In previous

work [16], we showed that using these techniques for the definition of requirements has a direct and positive impact on the obtained software product. In these experiments, participants who played the stakeholder role found it easier to express their requirements in this way, defining them in half the time they spent with traditional techniques.

Concerning the WAMRI extension for VisualStudio Code, it should be noted that there are too many options in terms of technologies, and we have tried to take as a guiding criterion the incorporation of popular technologies in our experimental tools. In that sense, VisualStudio Code is one of the most widely used IDES today and several thousand extensions are available in its extension market. Similarly, Django, the Web Framework we've been experimenting with so far, is a very popular and mature framework. Soon we hope to be able to reach a good level of usability in our most recent tools. That will allow us to carry out new experiments over more applications and frameworks and get new feedback as well as discover new alternatives for improvements and challenges.

Author contributions Conceptualization, L.G.M., L.A. and D.F.; WAMRI Extension software, L.G.M.; Mockplug Extension software D.F.; investigation, All.; writing—original draft preparation, L.G.M., L.M. and D.F.; writing—review and editing, All.; supervision, G.R., D.F; project administration, D.F.;

Acknowledgments We would like to thank to Matías, Ian, Gastón and Nicolás, who has collaborated with us in the Trello REST API utilization and the migration of a part of MockPlug tool to modern JS.

Datasets The datasets generated during and/or analysed during the current study are available in the Figshare repository.
https://doi.org/10.6084/m9.figshare.20060363
https://doi.org/10.6084/m9.figshare.19096781

Declarations All authors declare that they have no conflicting interests. All authors declare that they do not have any competing financial interests nor any personal relationships that could seem to have influenced the presented work.

References

1. Almonacid, S., Klagges, M.R., Navarro, P., Morales, L., Pazos, B., Puigbó, A.C., Firmenich, D.: Mobile and wearable computing in patagonian wilderness. In: Conference on Cloud Computing and Big Data. pp. 137–154. Springer (2019)
2. Antoniol, G., Canfora, G., De Lucia, A., Merlo, E.: Recovering code to documentation links in oo systems. In: Sixth working conference on reverse engineering (cat. No. PR00303). pp. 136–144. IEEE (1999)
3. Atlassian: Trello REST API. https://developer.atlassian.com/cloud/trello/rest/api-group-actions/, accessed: 2022-02-03

4. Beynon-Davies, P., Holmes, S.: Integrating rapid application development and participatory design. IEE Proceedings-Software **145**(4), 105–112 (1998)
5. Biniok, J.: Tampermonkey. https://www.tampermonkey.net/, accessed: 2022-02-03
6. Bouvin, N.O.: Unifying strategies for Web augmentation. Proceedings of the tenth ACM Conference on Hypertext and hypermedia : returning to our diverse roots returning to our diverse roots - HYPERTEXT '99 pp. 91–100 (1999), http://portal.acm.org/citation.cfm?doid=294469.294493
7. Bradley, N., Fritz, T., Holmes, R.: Context-aware conversational developer assistants. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). pp. 993–1003. IEEE (2018)
8. Chrome: Developer extensions. https://developer.chrome.com/docs/extensions/, accessed: 2022-02-03
9. Coman, I., Sillitti, A.: Automated segmentation of development sessions into task-related subsections. International Journal of Computers and Applications **31**(3), 159–166 (2009)
10. Coman, I.D., Sillitti, A.: Automated identification of tasks in development sessions. In: 2008 16th IEEE International Conference on Program Comprehension. pp. 212–217. IEEE (2008)
11. DarkReader: Browser extension. https://darkreader.org/, accessed: 2022-02-03
12. Díaz, O.: Understanding web augmentation. In: International conference on web engineering. pp. 79–80. Springer (2012)
13. Díaz, O., Arellano, C.: The augmented web: rationales, opportunities, and challenges on browser-side transcoding. ACM Transactions on the Web (TWEB) **9**(2), 1–30 (2015)
14. Eclipse Foundation: Eclipse IDE. https://www.eclipse.org/eclipseide/, accessed: 2022-02-03
15. Firmenich, D., Firmenich, S., Rivero, J.M., Antonelli, L.: A platform for web augmentation requirements specification. In: International Conference on Web Engineering. pp. 1–20. Springer (2014)
16. Firmenich, D., Firmenich, S., Rivero, J.M., Antonelli, L., Rossi, G.: Crowdmock: an approach for defining and evolving web augmentation requirements. Requirements Engineering **23**(1), 33–61 (2018)
17. Firmenich, D., Firmenich, S., Rossi, G., Wimmer, M., Garrigós, I., González-Mora, C.: Engineering web augmentation software: A development method for enabling end-user maintenance. Information and Software Technology **141**, 106735 (2022)
18. Firrmenich, D.: WAMRI component c test results. https://doi.org/10.6084/m9.figshare.19096781, accessed: 2022-02-03
19. Fowler, M., Highsmith, J., et al.: The agile manifesto. Software development **9**(8), 28–35 (2001)
20. Gasparic, M., Murphy, G.C., Ricci, F.: A context model for ide-based recommendation systems. Journal of Systems and Software **128**, 200–219 (2017)
21. Gobbo, F., Vaccari, M.: The pomodoro technique for sustainable pace in extreme programming teams. In: International Conference on Agile Processes and Extreme Programming in Software Engineering. pp. 180–184. Springer (2008)
22. González, V.M., Mark, G.: " constant, constant, multi-tasking craziness" managing multiple working spheres. In: Proceedings of the SIGCHI conference on Human factors in computing systems. pp. 113–120 (2004)
23. Google: Chrome web store. https://chrome.google.com/webstore/, accessed: 2022-02-03
24. Greasespot: The weblog. https://www.greasespot.net/, accessed: 2022-02-03

25. Greasyfork: Safe an useful user scripts. https://greasyfork.org/, accessed: 2022-02-03
26. Holmes, R., Walker, R.J., Murphy, G.C.: Strathcona example recommendation tool. In: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering. pp. 237–240 (2005)
27. Hummel, O., Janjic, W., Atkinson, C.: Code conjurer: Pulling reusable software out of thin air. IEEE software **25**(5), 45–52 (2008)
28. Kersten, M.: Focusing knowledge work with task context. Ph.D. thesis, University of British Columbia (2007)
29. Kersten, M., Murphy, G.C.: Mylar: a degree-of-interest model for ides. In: Proceedings of the 4th international conference on Aspect-oriented software development. pp. 159–168 (2005)
30. Kersten, M., Murphy, G.C.: Using task context to improve programmer productivity. In: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering. pp. 1–11 (2006)
31. Kersten, M., Murphy, G.C.: Using task context to improve programmer productivity. In: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering. pp. 1–11 (2006)
32. Kersten, M., Murphy, G.C.: Reducing friction for knowledge workers with task context. AI Magazine **36**(2), 33–41 (2015)
33. Luo, L., Schäf, M., Sanchez, D., Bodden, E.: Ide support for cloud-based static analyses. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 1178–1189 (2021)
34. Ma, X., Xu, H., Gao, H., Bian, M.: Real-time multiple-workflow scheduling in cloud environments. IEEE Transactions on Network and Service Management **18**(4), 4002–4018 (2021)
35. Maalej, W.: Task-first or context-first? tool integration revisited. In: 2009 IEEE/ACM International Conference on Automated Software Engineering. pp. 344–355. IEEE (2009)
36. Maalej, W., Ellmann, M., Robbes, R.: Using contexts similarity to predict relationships between tasks. Journal of Systems and Software **128**, 267–284 (2017)
37. Macaulay, L.: Requirements for requirements engineering techniques. In: Proceedings of the Second International Conference on Requirements Engineering. pp. 157–164. IEEE (1996)
38. Melo, G., Alencar, P., Cowan, D.: A cognitive and machine learning-based software development paradigm supported by context. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER). pp. 11–15. IEEE (2021)
39. Microsoft: Visual Studio API. https://code.visualstudio.com/api, accessed: 2022-02-03
40. Molina-Ríos, J., Pedreira-Souto, N.: Comparison of development methodologies in web applications. Information and Software Technology **119**, 106238 (2020)
41. Moore, J.M.: Communicating requirements using end-user gui constructions with argumentation. In: 18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings. pp. 360–363. IEEE (2003)
42. Morales, L., Navarro, P., Cintas, C., Gonzalez-Jose, R., Ramallo, V., Delrieux, C.: Bulsarapp: Interactive visual analysis for surname trend exploration. IEEE Computer Graphics and Applications (2021)

43. Mozilla: Browser extensions. https://developer.mozilla.org/es/docs/Mozilla/Add-ons/WebExtensions, accessed: 2022-02-03
44. Murphy, G.C.: Beyond integrated development environments: adding context to software development. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER). pp. 73–76. IEEE (2019)
45. Muşlu, K., Brun, Y., Holmes, R., Ernst, M.D., Notkin, D.: Speculative analysis of integrated development environment recommendations. ACM SIGPLAN Notices **47**(10), 669–682 (2012)
46. Mylyn: Task and application lifecycle management. https://www.eclipse.org/mylyn/, accessed: 2022-02-03
47. Portugal, I., Oliveira, T., Alencar, P., Cowan, D.: Mylynsdp—process-aware artifact filtering based on interest. Journal of the Brazilian Computer Society **26**(1), 1–35 (2020)
48. Rashid, A., Meder, D., Wiesenberger, J., Behm, A.: Visual requirement specification in end-user participation. In: 2006 First International Workshop on Multimedia Requirements Engineering (MERE'06-RE'06 Workshop). pp. 6–6. IEEE (2006)
49. Rivero, J.M., Grigera, J., Rossi, G., Luna, E.R., Montero, F., Gaedke, M.: Mockup-driven development: providing agile support for model-driven web engineering. Information and Software Technology **56**(6), 670–687 (2014)
50. Robbes, R., Lanza, M.: Improving code completion with program history. Automated Software Engineering **17**(2), 181–212 (2010)
51. Robillard, M.P.: Automatic generation of suggestions for program investigation. In: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering. pp. 11–20 (2005)
52. Rodríguez, P., Mäntylä, M., Oivo, M., Lwakatare, L.E., Seppänen, P., Kuvaja, P.: Advances in using agile and lean processes for software development. In: Advances in Computers, vol. 113, pp. 135–224. Elsevier (2019)
53. Rodríguez, P., Markkula, J., Oivo, M., Turula, K.: Survey on agile and lean usage in finnish software industry. In: Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 139–148. IEEE (2012)
54. Röthlisberger, D., Nierstrasz, O., Ducasse, S.: Autumn leaves: Curing the window plague in ides. In: Proceedings of the 16th Working Conference on Reverse Engineering (WCRE 2009). IEEE Computer Society (2009)
55. Ruensuk, M.: An implementation to reduce internal/external interruptions in agile software development using pomodoro technique. In: 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS). pp. 1–4. IEEE (2016)
56. Sarkar, S., Parnin, C.: Characterizing and predicting mental fatigue during programming tasks. In: 2017 IEEE/ACM 2nd International Workshop on Emotion Awareness in Software Engineering (SEmotion). pp. 32–37. IEEE (2017)
57. Schön, E.M., Thomaschewski, J., Escalona, M.J.: Agile requirements engineering: A systematic literature review. Computer Standards & Interfaces **49**, 79–91 (2017)
58. Vieira, R.D., Farias, K.: Cognide: A psychophysiological data integrator approach for visual studio code. In: Proceedings of the 34th Brazilian Symposium on Software Engineering. pp. 393–398 (2020)

59. Viriyakattiyaporn, P., Murphy, G.C.: Improving program navigation with an active help system. In: Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research. pp. 27–41 (2010)
60. Whittle, J., Hutchinson, J., Rouncefield, M.: The state of practice in model-driven engineering. IEEE software **31**(3), 79–85 (2013)
61. Zhu, Y., Zhang, W., Chen, Y., Gao, H.: A novel approach to workload prediction using attention-based lstm encoder-decoder network in cloud environment. EURASIP Journal on Wireless Communications and Networking **2019**(1), 1–18 (2019)