Improving Legacy Applications with Client-Side Augmentations

José Matías Rivero[1(✉)], Matías Urbieta[1,2], Sergio Firmenich[1,2], Mauricio Witkin[1], Ramón Serrano[1], Viviana Elizabeth Cajas[1,3], and Gustavo Rossi[1,2]

[1] LIFIA, Facultad de Informática, Universidad Nacional de La Plata, La Plata, Argentina
{jose.matias.rivero,matias.urbieta,sergio.firmenich, gustavo.rossi}@lifia.info.unlp.edu.ar, mjwitkin@gmail.com, serrano.ramon.m@gmail.com

[2] CONICET, Buenos Aires, Argentina

[3] Facultad de Ciencias Administrativas y Económicas, Universidad Tecnológica Indoamérica, Quito, Ecuador
vivianacajas@uti.edu.ec

Abstract. Mobile devices have become the most prominent channel to access Web applications. While every mobile device platform like Android or iOS has their own application ecosystem, they are also often used to access Web sites which are not property rendered in such devices. Adapting existing sites to be usable on this kind of devices (with a reduced viewport), techniques like Responsive Design and Progressive Web Applications have appeared, proposing guidelines and good practices to cope with device interface limitations. However, these techniques require a notable engineering effort and cost to adapt sites to be mobile-friendly. In this work, we present an approach and tool that allows to quickly adapt an existing Web application to be usable for mobile devices. The approach does not require to redesign its frontend and can be applied even without having control of the servers in which the application is hosted since we use a client-side approach. To assess its applicability, a controlled experiment has been carried out in which we evaluated the usability of the adapted sites.

Keywords: Mobile · Software engineering · Responsive design Web augmentation

1 Introduction

The increasing usage of mobile devices [2, 26] allows users to access a variety of developed applications which have not being designed to be browsed from smartphones or tablets. Consequently, the user might face usability problems related to the required page, e.g. needing to zoom or scroll for reaching information or a feature available in a Web page which can detriment the application adoption in the long-term. Since the appearance of Responsive Design and Progressive Web Applications [9] concepts, application developers rely on guidelines to provide a fluid experience when browsing from mobile devices. These guidelines do not only suggest how content should be rendered but also brings new architectural and behavioral approaches to get real mobile experience such as off-line first support. The availability of guidelines for native, hybrid and Web development [12] gives support to new mobile solutions which are often built from scratch. However, for legacy applications which were not thought to be run in mobile devices, this implies migration or even re-engineering the application. The migration of a Web-based solution to a mobile one can be delayed, and even avoided, in many cases due to the high cost of re-engineering the application for a proper mobile access, which for small businesses could represent almost prohibitive costs.

The dawning of techniques like Augmentation [4] enable developers to enhance existing Web applications with new functionalities or contents by altering Web pages once those are loaded at the client-side. Any Web page may be enhanced through the manipulation of its Document Object Model (DOM); it is possible to add, remove or change content, styles and functionalities through local scripts or browser extensions without requiring changes on the server-side. This way, Augmentation approaches provide the power to add behavior or adapt existing sites directly in a non-expensive fashion, without any intervention of the owner of the Web application. Most augmentation approaches rely on the client-side for introducing new functional requirements such as new forms or actions. However, augmentation can also be used to restructure the application for improving non-functional requirements such as Usability or Performance. Deep changes in the way the application is browsed and perceived may improve its adoption on new devices even after the application is online.

Current trends in agile software development [10] rely on quick prototyping since they help to validate easily whether a new requirement meets the users' needs. For instance, agile approaches promote the incremental development of Minimum Viable Product (MVP) [18] delivered in a few iterations by considering the value-based contribution to the business. In this context augmentation approaches can be used to help a development team wanting to obtain a running example for the improvement of the User Experience of a Web application, or to analyze the result of including another Website's content. By using Web Augmentation, it is possible to perform advanced A/B testing without the need of branching the current version of an application, coding the new feature, and build and release both versions for comparison.

In this paper, we describe PortAug, an approach that uses Response Design guidelines, and non-expensive adaptation techniques facilitated by Augmentation to provide an efficient, assisted and production-ready adaptation approach for Web applications that are not specifically adapted for mobile devices. The nature of the approach makes it agile-ready, since it allows to work in short iterations and provide working software outputs after every iteration. Concretely we present:

(i) An approach for supporting the adaptation of sites, centered on the user interface, requiring minor user intervention and no re-engineering tasks
(ii) A supporting tool for (i)
(iii) A set of Architectural refactorings, that are performed without user or developer intervention, that help to improve application responsiveness and scalability.

In order to evaluate the effectiveness of our approach, we conducted an evaluation that measures how the application usability is improved with our approach.

The rest of the paper is organized as follows: In Sect. 2 we describe some background work; we discuss related work to the approach in Sect. 3 and describe the user-interface centric part of the approach in Sect. 4. In Sect. 5 we describe the details of an evaluation we did over the approach to assess the usability of the adapted Web applications using Responsive Patterns and assistance by the tool through a controlled experiment. Finally, in Sect. 6 we conclude the paper and discuss some future work we are pursuing.

2 Background

Web Augmentation is a set of techniques allowing to modify, enrich or adapt existing Websites from the client-side without altering the original site. It is mostly implemented installing scripts that run locally on the browser and it is being used by a growing and large number of users. Thousands of extensions for adapting Web content may be found at the Web browser stores, and significant communities support some of these tools. End-users and other stakeholders with programming skills may interact in such communities for the creation, sharing, and improvement of specific augmentation artifacts. For instance, the Userstyles community (http://userstyles.org) offers a wide number of scripts that augment Web sites by adding further CSS specifications that change the content presentation. Userscripts communities, such as Greasyfork (https://greasyfork.org/), offer repositories of scripts with a wider spectrum of purposes, since they support different weavers of JavaScript code (e.g. GreaseMonkey or TamperMonkey). In all these communities, no matter which tool they support, there is a dependency between users with and without programming skills, since not all of them can implement the solutions they need and ask others for help. In this light, some research works proposed End-User Development (EUD) approaches to let users specify their own augmentation artifacts; these works are discussed in the related work section.

Responsive Design [12] is a set of techniques that allow Web user interfaces to be defined in a way that it will adapt to different viewports in order to be usable in a variety of devices, including both desktop and mobile environments. This implies that, instead of writing different, isolated versions for specific devices, pages are written only once but with a set of rules and restrictions that allow them to adapt to the device characteristics. From the frontend layout perspective, Responsive Design requires and involves (1) a flexible, grid-based layout, (2) flexible images and (3) media queries, a CSS3 module that allows to write specific rules to be applied depending on the characteristics like the viewport size. While pages have to be designed considering these 3 technical considerations, several libraries and frameworks like Twitter Bootstrap [24] provide a set of tools and concepts that help respecting them. Libraries like these include, for instance, CSS3 media queries compliant containers for standard device sizes to ease development.

3 Related Work

In the field of native mobile applications (e.g. applications that are designed to run over a specific platform like OS X or Android), several approaches have been proposed to avoid the cost of writing an entire app from scratch for every platform to overcome the aforementioned portability problems. One way to cope with heterogeneity is to use Model-Driven Development approaches that allow to abstract concept in a platform-independent way, leaving detailed specifications to be coded manually. A summary of those approaches is described in [5]. In this context, [20] proposes a framework based on cloud services that allows to build applications in a platform-independent fashion. A similar approach is proposed by [17], in which a framework to adapt cloud content to different platforms is proposed. Under this approach, the content is specified only once and then only the adaptations for custom platforms have to be coded independently.

General user interface (UI) adaptation for mobile devices is an extensively studied field, including a range of semi-automatic and manually assisted approaches. In this context, [7] proposes a prototype of a tool that uses machine learning to recognize use patterns in a UI to generate adapted interfaces for mobile devices. While the approach is automatic, it requires final user intervention to assess the final UIs. Similarly, [1] proposes a tool that uses genetic evolutionary algorithms that is able to generate and deliver adapted versions of existing Web sites. This tool uses the existing content to create alternative designs considering designer preferences, device limitations and destination viewport size. While authors prove that this approach is feasible, the quality of the UIs generated is not comparable to those built manually and it still requires human intervention to define detailed preferences and solve conflicts in the generated UIs.

A very studied subtype of UI adaptation for mobile devices is devoted to Web sites. This type of adaptation consists in modifying the DOM tree – in client- and/or server-side – to make it usable for mobile devices using geometric algorithms, image enhancing, element removal, font scaling among other techniques. In this field, [3, 11] present a methodology for client-side personalization (i.e. in the Web browser) to existing Web sites oriented to users without detailed knowledge in design. While this work does not consider adaptations for mobile devices, it shows how client-side adaptation can be used for successfully implementing personalization and adaptation.

In the context of Responsive Web Applications, in [8] a Web page design method is proposed through a table adaptable to the resolution of clients, calculating the size of characters and other elements to obtain the best visual effect.

Nebeling et al. presented the Crowadapt approach [13] leveraging on the crow for assessing what adaptations provide the best usability improvement on the UI. Although this approach was focused on Web desktop improvement, in [14] the authors presented a tool for re-authoring the content of Web pages to work on cross-devices where the content is distributed and synchronized between devices. Although Nebeling's work and ours aim at adapting the UI to improve its responsiveness and usability, our work focuses on changes in the application's client-side using augmentations running in standard Web browsers instead of proposing a cross-device approach. Also, instead of giving a set of tools to freely transform the UI, we force to implement well-known patterns, layout and frameworks based on industrial-proven technologies like Bootstrap [24], facilitating higher level adaptations like, for instance, menu transformations for mobile devices. Additionally, we consider mobile-related problematics like poor connectivity and UI architecture refactorings to improve maintainability and facilitating interface re-engineering. Finally, both approaches present different strategies with the collaborative support for the crowd. Nebeling's uses its own platform to generate and run the adaptation rules while our approach uses a browser's extensions to design the enhancements and the adaptation script is distributed in well-known repositories like UserScripts [25].

The approach presented in this paper does not rely on machine learning or semi-automated algorithms but it allows non-technical users to build adaptations through structured wizards, relying on well-known Responsive Design patterns. Through the tooling provided, end users can build a mobile adaptation to an existing Web application, generating client-side adaptations that can be run locally and, if proven to work effectively, they can be easily implemented in productive environments making it available for all users. This assistance and simplicity allows to implement adapted version of existing Web applications in a quick and cost-effective fashion and also includes architectural and non-functional aspects that improve application performance and maintainability in the future.

4 Our Approach in a Nutshell

The PortAug process, briefly described in Fig. 1, starts selecting one Web page in a Web site that presents usability issues on a reduced viewport like, for instance, a mobile phone screen (Step 1). This process can be performed by any Web user (from now on, the User) with no technical knowledge i.e., with no knowledge in Web standards like HTML, CSS or JavaScript. After this step, one of the different responsive layouts have to be manually selected to adapt the page (Step 2). Step 2 and the rest of the further steps are accomplished with the assistance of a tool implemented as a Web browser extension, thus allowing to apply and test the adaptation in the same environment in which the non-responsive application is accessed. When a layout is chosen, the real adaptation process (Step 3) starts; it consists in selecting visual components of the non-responsive site and determining where they will be located in the responsive layout chosen. The User who is building the adaptation is able to check, in real time, how the adaptation will look in a mobile device, undo adaptations if they do not look good or apply any correction to the current adaptation configuration (Step 4); several cycles of Step 3 and Step 4 are performed until the User reaches a desirable adaptation result for the page. After this adaptation is completed, a script that runs the adaptation can be generated and published to a repository (Step 5). This script is generated automatically based on the configured adaptation and it basically initializes the responsive layout and applies all the transformations required to adapt the content. It also includes a set of improvements for mobile platforms that will be commented later. This script can be used to test the adaptation directly in the browser and it can be further published in well-known Web Augmentation repositories like Userscripts [25]. Since a Web site comprises a set of Web pages, the aforementioned process should be repeated for every page in the site resulting in an aggregated script.

The lasts steps in the process consist in using the adaptation generated and it implies downloading from the repository and installing it in a Web browser with a script manager enabled (like Tampermonkey [23]) and reloading the site (Step 6 and Step 7 respectively). As a case study to describe the adaptation process we use the Redmine site [22] which has not been optimized for being accessed from mobile devices. Throughout this section we will use this case study to describe deeply every step, including screenshots of the PortAug tool in action.

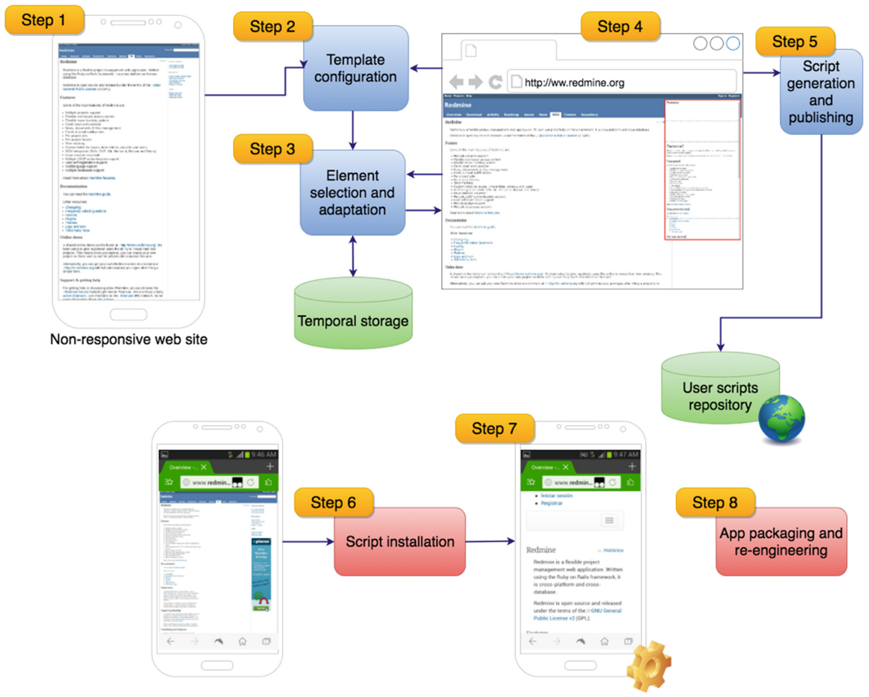


Fig. 1. Application improvement process using redmine example

4.1 Step 1: Select the Web Site to Adapt

As aforementioned, the first step in the PortAug process is to select a site to adapt. The site presents a usability issue (not implementing a Responsive Design), which implies that, when the viewport is reduced, the general layout does not change, widgets are not hidden, and fonts and images are not adapted to improve their look accordingly to the device. In Fig. 1, Step 1, the reader can appreciate how the Redmine's site does not adapt to the viewport of the smartphone from which it is accessed; As it can be seen, no layout or font adjustment are performed in the page and, as a result, all the text in the page becomes hard to read and the top and left menu become inaccessible without zooming.

4.2 Step 2: Template Configuration

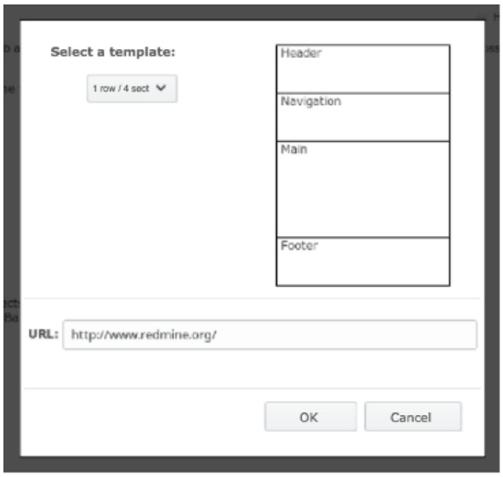
After a non-responsive page has been identified, one of the available responsive layouts must be chosen to reconfigure the page content for a proper mobile rendering. Figure 2.a shows the layout selection page that the PortAug tool provides. Responsive layouts offered include several rows and optionally columns depending on User preferences. It is important to note that, even if a multi-column layout is chosen, it will be adapted automatically (using the aforementioned techniques for Responsive Design implementation) to a single-column layout if the viewport becomes too small, which will ensure that the site adapts to all types of viewport sizes. This behavior is implemented by making use of the well-known responsive framework Bootstrap [24], which is included in the tool and in the final scripts generated by it. In Fig. 2.a it can be seen that a simple column layout has been chosen for adapting the page available at [22].

4.3 Step 3 and 4: Element Selection, Adaptation and Validation

After a layout is chosen, it must be populated with elements of the original, non-responsive Website. This is done iteratively through a set of linear steps: (1) selecting the element from the non-responsive page that has to be placed into a container of the responsive layout, (2) choosing such container and (3) selecting an adaptation strategy. While (1) and (2) are simple and basic tasks, (3) requires some extra knowledge of how the adaptation will work. The adaptation pattern is basically a DOM transformation that will be used when or before inserting the element being adapted in the new layout. Some of the adaptation patterns provided by the tool are:

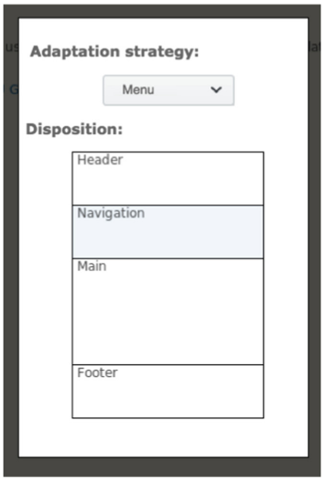
– **Insert**, where the element and all its children are simply cloned into the new layout.
– **Transform to menu**, where all the links are grouped into an expansible menu. For example, a hamburger menu is used instead of showing the links directly.
– **Splash screen**, where the element is shown as a splash screen for the page as it is usual in mobile applications.
– **Form**, in which internal data-entry and related elements like input field, dropdowns and labels are re-located for a better rendering and use in mobile contexts.

The UI used for selecting the container and the adaptation strategy is depicted in Fig. 2.b. The output of this process is a set of 3-uples consisting of the original element, the destination container, and the adaptation pattern. After several iterations of these 3 steps, a first version of the adaptation will be available. The user User can preview how the current adaptation looks like through a specific window that augments the original Website as it can be appreciated in Fig. 1, Step 4.


Select a template: 1 row / 4 sect

URL: http://www.redmine.org/

(a) Tool user interfaces


Adaptation strategy: Menu

Disposition:

(b) Element mapping and adaptation strategy

Fig. 2. Tool user interfaces

4.4 Step 5: Script Publishing and Execution

After a first acceptable iteration of the adaptation is reached, the User can opt to publish an adaptation script. This script[1] (1) imports required libraries, (2) creates the adapted layout and (3) applies all the transformations and operations needed to instantiate the adaptation. From the technical point of view, Step 1 is performed by just adding some script references in the HTML < `head` > element of the page. Step 2 creates the required elements in the HTML < `body` >. Finally, Step 3 iterates over the aforementioned 3-uples (original element, destination container, adaptation pattern), gathers the original element, applies any transformation required if needed depending on the adaptation strategy and inserts the result on the destination container using a specific insertion method that also depends on the adaptation strategy chosen. The default insertion method of the potentially transformed element is to add it at the end of the last element in the container. If one or more of the elements of the original site that are referenced in the 3-uples are not found, then as a protective measure the execution of the whole transformation is cancelled to prevent making the site non-functional.

The resulting script can be imported into any Web augmentation extension for current browsers like GreaseMonkey or GreasyFork and, since it will contain all the required metadata to adapt the destination site, no extra user involvement will be required. When a final-user visits the site (in this case, redmine.org) the script will match the URL and will be run by the augmentation extension, thus applying the generated adaptation.

[1] PortAug adaptation implementation - https://github.com/serranorm/UNLP, last accessed 4-Apr-2018.

4.5 Architectural Application Refactorings

So far, we have introduced one aspect of the approach aimed to provide assisted layout adaptation for legacy Web sites. However, as aforementioned, the approach also includes the implementation, execution, and generation of several meaningful refactorings (to the target application) which allow better tolerance to connectivity problems, improved responsiveness and potential architectural improvements that will ease a re-engineering of the application. For the sake of space, we briefly comment these improvements in this subsection:

Caching and Offline Support. Draft standards like ServiceWorkers[2] and CacheStorage[3] allow Web applications to load even when there is no connectivity at all. Leveraging on these technologies, the generated script is able to (1) cache the layout used across the pages, (2) cache the content of every container in the layout and (3) load the page in no-connectivity contexts. End-users or developers do not have to change or code anything to have this support.

Splash Screen. If the Splash Screen adaptation strategy is used, the script pre-loads an empty screen, shows the content of the splash screen in parallel with the loading of the rest of the contents, and then makes it disappear when the site becomes ready. The Splash screen is also cached.

UI Reengineering. Even though the focus of the approach was centered on avoiding re-engineering the site and using Augmentation techniques to adapt it for mobile environments on the client side, the adaptation defined at the client-side can be used to generate a modern architecture for the legacy Web application (Step 8 in Fig. 1). For every 3-uple extracted from the adaptation process, Component and Service elements can be generated for the Angular[4] Web MVC framework. In this context, all the JavaScript and TypeScript files are generated and added to the original Web Application which, after some minimal deployment changes, will look as in the adapted version. In this case it will not be required to run the adaptation, since the adaptations will be generated and packaged by default and all libraries will be included natively.

Mobile Packaging. Having all the assets indexed as a side effect of marking the DOM that is adapted to a responsive layout, the application can be seamlessly packaged for hybrid platforms using technologies like Apache Cordova and Ionic[5]. Finally, if the engineering team decides to re-implement or adjust the site, they can start from the packaged and refactored site, now with client-side MVC Web support.

[2] W3C - Service Workers - https://goo.gl/yaU3BU, last accessed 21-Jan-2018.

[3] Cache Storage - https://goo.gl/vF9jiQ, last accessed 21-Jan-2018.

[4] Angular - https://angular.io/, last accessed 21-Jan-2018.

[5] Ionic - https://ionicframework.com/, last accessed 21-Jan-2018.

5 Evaluation

5.1 Goals, Hypotheses and Metrics

Following the reporting guidelines of experiments suggested in [19], we defined the goal of the evaluation experiment in the following way:

Analyze *legacy Web applications transformed to Web mobile* ***for the purpose of*** *measuring how the user experience is improved* **with respect to** *their usability* ***from the point of view of*** *researchers* ***in the context of*** *end-users who daily uses a smartphone.*

After defining the Goal, we proceeded to define the different questions that will allow to answer them.

Our main Research Question (RQ) is: *is the application's usability improved after the UI transformation?*

Whereas the Null Hypothesis (NH) in the experiment means no improvement between tested samples, we presented the Alternative Hypothesis (AH) to each metric which will be studied later additionally. The μ_{Original} denotes the mean of the metric gathered from the original UI with no adaptation while the $\mu_{\text{Augmented}}$ denotes the mean of the metric measured from the enhanced UI. We considered the well-known usability metrics [6] to answer the question:

- Binary task completion. It refers to measures whether the users complete tasks or not. The researches marks if the task is completely performed after the experiment. HA: $\mu_{\text{Original}} \leq \mu_{\text{Augmented}}$.
- Task completion time. The time users take to complete the tasks. The researched performed the corresponding time-keeping during the experiment recording the spent seconds. HA: $\mu_{\text{Original}} \geq \mu_{\text{Augmented.}}$
- Event rate. The display events (zooming, clicking, and scrolling) introduced by de user. Based on the recorded user activity during the experiment, the researched counted the performed events. HA: $\mu_{\text{Original}} \geq \mu_{\text{Augmented.}}$
- Standard satisfaction. Measure satisfaction level by using a question for an interval data from 1 (poor) to 5 (excellent). HA: $\mu_{\text{Original}} \leq \mu_{\text{Augmented.}}$

For a sake of space, some metrics were not considered in this study such as Recall, Learnability, etc.

5.2 Experiment Design

In order to answer these questions, we designed a between-subject design experiment where subjects were asked to use two different applications (experiment factors) which each one has two versions (alternatives of the factor) of their UI: original UI (control) and enhanced UI (treatment). Each subject was asked to complete a task in each application where the application's version was randomly assigned.

In this experiment, we focused on measuring how the user experience is improved with UIs enhancements in order to transform the application into a mobile friendly one. Therefore, we did not consider evaluating the user experience related to another approach.

5.3 Experimental Unit, Subjects, Instrumentation, and Data Collection

The Web sites of Redmine [22] and Pidgin [21] are the experimental units which were enhanced with augmentations that improved their UI for a mobile device' s viewport. The former is a well-known open-source issue tracker and the latter is an open-source instant messaging application. For each application, we designed a set of transformations for the UI so as to be mobile ready.

To evaluate the subjects' behavior, we asked the them to fill out a form where they should provide demographic information and their satisfaction level about the application. The tasks were simple requirements for browsing the site to reach some content:

In Redmine's site:

- Task 1 (RT1): Locate information about the Redmine's book in the main page;
- Task 2 (RT2): Browse the application for accessing a recorded issue.

In Pidgin's site:

- Task 1 (PT1): Check if the application is available in Italian language.
- Task 2 (PT2): Sign on in the development section of the site.

The participants were 34 end-users that included professionals, scientists, students, and government employees. The participants had a mix of gender (Male 68%, Female 32%), age (0–19yo 6%, 20–39yo 50%, 40–59yo 32%, and 60–79yo 12%), education (Primary education 15%, Secondary education 12%, Technician 9%, Undergraduate 29%, Bachelor or superior 35%), and Information Technology employee (yes 56%, no 44%).

The experiment protocol was executed in the same way with all the subjects. First of all, they filled out a demographic questionnaire, then they were asked to perform the tasks mentioned in previous subsection. Finally, we asked them some questions about their perception of the usability.

5.4 Analysis and Implication

All samples were processed using Mann–Whitney U test [16], which is a non-parametric statistical hypothesis test technique, considering a standard confidence level (α) of 0.05. This technique can be used with unequal sample size which is the case of this experiment. Additionally, we computed the effect size using Cliff's Delta technique.

To achieve the task of processing the collected results, we first processed and digitalized responses. Then, we used different scripts based on Python language (version 3.6.1) and Scipy library (version 0.19.0) to compute nonparametric hypothesis tests.

In order to answer the RQ, we will discuss the result of metrics introduced in Subsect. 5.1 which is shown in Table 1. For each one, we report Cliff's delta value for the effect size, and the p-value resulting from the hypothesis testing that compares the means of both approaches.

The experiment was accomplished by all the participants; that is, there is no difference in the *Binary task completion* metric. However, the *satisfaction*'s metric excels in performance in all tasks. In all cases, the p-value was lower than the alpha level 0.05 which allows to reject the null hypothesis and support the alternative one meaning a better performance of the metric.

The *Event rate* was split into three sub-metrics: Zoom, Scroll, and Clicks. Zoom UI events present a significant improvement for all the tasks with a p-value lower than 0.05. The *scroll* events reported a better performance when using augmented UIs but the PT1 shows no difference (p = 0.986). A post hoc study showed the end-users start using scrolling events in place of the zoom in/out because the content was rendered fitting the viewport's width in a longer page to check whether or not the Italian language was supported. And the *Click* events show some but not a statistically significant improvement in RT1 (p = 1.73) and Pidgin T2 (p = 1.73) and no difference in RT2 (p = 1) and PT1 (p = 1). After studying the samples, we realized that UI transformations did not introduced a refactoring that reduces the required clicks events for accomplishing the task. This fact supports why the metric's measures.

Finally, regarding the *time completion* metric, a half of tasks show an improvement in the time required for performing the task. The PT1 shows some but not statistically significant improvement (p = 1.175) and RT2 shows no preference (p = 0.640). A posteriori study points out that the introduction of a 'hamburger' menu required an additional click event to open it and consumed time animating the menu's expansion.

To calculate the effect size, we used cliff's delta technique which has as a result value in (−1,1) range. In this case, the values are bigger than 0.778 [15] in all evaluation where the NH is rejected depicting a high meaningfulness.

Table 1. Sample results

	Usability		Time		Zoom		Scroll		Clicks	
	Effect size	p-value	Effect size	p-value	Effect size	p-value	Effect size	p-value	Effect size	p-value
Redmine T1	1,520	0,000	−0,778	0,024	−2,805	0,000	−1,264	0,001	−0,343	0,173
Redmine T2	1,688	0,000	0,151	0,640	−2,358	0,000	−0.928	0,002	1,086	1,000
Pidgin T1	1,641	0,000	−0,195	0,172	−3,684	0,000	0,919	0,986	–	1,000
Pidgin T2	1,735	0,000	−1,075	0,003	−1,879	0,000	−2,450	0,000	−0,343	0,173

Summarizing, participants that were assigned to augmented UIs were more time-efficient and precise in the task execution than those that worked with the original one. We claim these results are preliminary evidence of augmentation approach benefits but a throughout analysis including a post-hoc study is required but for the sake of space these were not included.

5.5 Threats to Validity

There are several threats to validity that were considered during the experiment design.

Construct Validity. The experiment was designed to measure how the use of augmented UIs with a mobile-friendly layout improves the application usability. In order to reduce the experiment's complexity and bias introduction possibility, we defined the method (with or without augmentation) as the only independent variable.

Internal Validity. The subjects were selected randomly. The provided material was the same to all subject. We also checked that all the users had basic knowledge of smartphone usage and had not usage experience in any application of this kind.

External Validity. The subjects were mixed end-users. A broader experiment considering different subject of different cultures who have worked on different business domains will improve the generality of our claims.

Conclusion Validity. The experiment was based on objective metrics evaluated with all gathered data without any exclusion to guaranty that the outcome of the experiment analysis will be the same and avoiding hypothesis fishing. Finally, in order to avoid the impact of random irrelevancies on the experiment, we used a large number set of samples that helped the irrelevancies to become diluted.

6 Conclusions, Discussion and Future Work

In this paper, we described PortAug, an approach with its associated tooling that assists in creating adapted Responsive versions for non-Responsive Web sites in an assisted, quick and cost-effective manner using Web Augmentation techniques. We described the usability problems present on that area and the high costs involved to solve them, and then introduced our approach, described step by step with a real-world example and showing examples of the supporting tooling at work. It is noteworthy that this work contributes an emerging technology that brings to legacy systems mobile features in a feasible and economic which improve the user experience. Finally, we described the results of a controlled experiment which showed an improvement in the usability of augmented applications as well as a reduction in the scrolling and zoom in/out events.

While the approach showed to be productive through the use of augmentation techniques, it inherits some limitations from them. One of the most important is related to site changes. Since augmentation strategies work transforming the original DOM model of a web application, they use more or less concrete ways of identifying elements to be transformed within it. If the original website changed in some way in which that it prevents to apply the adaptation, all adaptation is cancelled as a preventive measure.

There are several threads and sub-areas for future work. Extending the tooling for supporting more adaptation strategies and layouts is clearly one need, since the current catalog is limited. Though the tooling provides assistance in building the adaptation, it still has to be iteratively and manually which, although saving a notable amount of work, it requires a noticeable time of manual intervention. Using page analysis and processing algorithms, a first version of an adaptation, including the selection of the best-matching template can be offered thus saving part of the manual work involved, represents another interesting area for future work. Also, we plan to include principles of Progressive Web Applications which transcend layout including features like responsiveness, load speed, tolerance to network failures etc. using a similar Web

Augmentation approach and assisting in performing backend changes if required. Also, providing an alert system to notify users if the site changes and facilitating a quick adaptation re-builder system is another interesting work path that we are pursuing. Finally, as an additional work path we plan to conduct an enterprise-wide acceptance test of the tool and methodology in local startup companies to validate its usefulness and provided added value.

References

1. Armenise, R. et al.: A tool for automatic adaptation of web pages to different screen size. In: ICEIS 2010 - Proceedings of the 12th International Conference on Enterprise Information Systems. pp. 91–98 (2010)
2. Barboutov, K., et al.: Ericsson Mobility Report. https://goo.gl/FMNgUQ
3. Bosetti, G.A., et al.: An approach for building mobile web applications through web augmentation. J. Web Eng. **16**(2), 75–102 (2017)
4. Díaz, O.: Understanding web augmentation. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE 2012. LNCS, vol. 7703, pp. 79–80. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35623-0_8
5. Gaouar, L., et al.: Model driven approaches to cross platform mobile development. In: Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication - IPAC 2015, pp. 1–5 (2015)
6. Hornbæk, K.: Current practice in measuring usability: challenges to usability studies and research. Int. J. Hum Comput Stud. **64**(2), 79–102 (2006)
7. Leeladevi, B., et al.: Transforming a website from desktop to mobile a cross platform viewpoint. In: Proceedings of the 2015 International Conference on Green Computing and Internet of Things, ICGCIoT 2015 (2016)
8. Liu, Q., et al.: A web page design method for multi-terminal devices. In: 2015 International Symposium on Educational Technology, vol. 3(1), pp. 3–7 (2015)
9. Marcotte, E.: Responsive Web Design. A Book Apart (2011)
10. Martin, R.C.: Agile Software Development: Principles, Patterns, and Practices. Prentice Hall PTR, Upper Saddle River (2003)
11. Miján, J.L., Garrigós, I., Firmenich, S.: Supporting personalization in legacy web sites through client-side adaptation. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds.) ICWE 2016. LNCS, vol. 9671, pp. 588–592. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38791-8_54
12. Natda, K.: Responsive web design. Eduvantage **306**, 1–22 (2013)
13. Nebeling, M., et al.: CrowdAdapt: enabling crowdsourced web page adaptation for individual viewing conditions and preferences. In: Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 23–32. ACM, New York (2013)
14. Nebeling, M.: XDBrowser 2.0: semi-automatic generation of cross-device interfaces. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, pp. 4574–4584. ACM, New York (2017)
15. Romano, J., et al.: Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and other surveys? In: Florida Association of Institutional Research Annual Meeting, pp. 1–33 (2006)
16. Ross, S.M.: Introduction to Probability and Statistics for Engineers and Scientists. Academic Press, Cambridge (2004)

17. Song, R., et al.: Design and implementation of the web content adaptation for intelligent tourism cloud platform. In: Proceedings - 2012 International Conference on Control Engineering and Communication Technology, ICCECT 2012 (2012)
18. Torrecilla-Salinas, C.J., et al.: Estimating, planning and managing Agile web development projects under a value-based perspective. Inf. Softw. Technol. **61**, 124–144 (2015)
19. Wohlin, C., et al.: Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers Norwell, Netherlands (2000)
20. Wu, K.C., et al.: Development model and environment for dynamic mobile cloud services. In: Proceedings - 2012 IEEE Asia Pacific Cloud Computing Congress, APCloudCC 2012 (2012)
21. Pidgin, the universal chat client. https://www.pidgin.im/
22. Redmine. http://www.redmine.org/
23. Tampermonkey. https://tampermonkey.net/
24. Twitter Bootstrap. http://getbootstrap.com/
25. Userscripts. http://userscripts-mirror.org/
26. World Internet Users Statistics and 2017 World Population Stats. http://www.internetworldstats.com/stats.htm