专题：不断变化的Web应用程序

使用客户端重构的个性化Web可访问性

Alejandra Garrido,Sergio Firmenich,Gustavo Rossi,and Julián Grigera
阿根廷拉普拉塔国立大学
Nuria Medina-Medina 西班牙格拉纳达大学
Ivana Harari 阿根廷拉普拉塔国立大学

According to W3C accessibility standards, most Web applications are neither accessible nor usable for people with disabilities. Developers often solve this problem by building separate accessible applications, but these are seldom usable and typically offer less functionality than the original. Another common solution is to maintain a single application, but create an accessible view by applying on-the-fl transformations to each requested page －a solution that rarely suits all audiences. A third solution is described here: let users improve Web accessibility in their client browsers through interface refactorings, which offer many customized, accessible views of a single application.通过W3C可访问性标准，大部分Web应用程序对于残疾人群都是不可访问且不可用的。开发者经常通过创建独立的可访问性应用程序来解决这个问题，但是那很少可用，并且通常提供的功能比原来少。另一个常用的解决方法是保持单一的应用程序，但是通过对每个请求的页面采用FL变换来创建一个可访问的视图，是一个难以适用于所有用户的解决方案。这里描述的是第三种解决方案：让用户通过界面重构提高他们客户端浏览器的Web可访问性，它提供了许多单个应用程序定制的可访问视图。

1 Introduction

Refactoring was originally conceived as a technique to improve software’s internal qualities －-such as understandability and maintainability －while preserving semantics.1 In prior work, we adapted the refactoring approach to improve a Web application’s external attributes, such as usability.2 These Web refactorings consist of small navigation or interface transformations that enhance perceivable aspects of Web applications, such as user interaction and content presentation, while preserving functionality. Refactorings can also solve accessibility and usability problems for disabled users.3 Still, it’s usually impractical to address interface improvements for all audiences because disabilities can vary dramatically in nature (visual, cognitive, or motor), severity (blindness, color blindness, or strabismus) and extent (total or partial). In such different contexts, “one for all”is barely feasible.

重构的初衷是提高软件内部质量的一种技术－－如可理解性和可维护性，同时保留语义1。在以前的工作中，我们采用了重构方法来提高Web应用程序的外部属性，如可用性2。这些Web 重构包括小的导航或界面转换来增强Web应用程序的感知性方面，比如用户互动和内容呈现方式，同时保留功能。重构也可以解决残障用户3 的可访问性和可用性问题。还有，它对于解决所有观众的界面改进问题通常是不切实际的，因为残障用户在性质（视力障碍、认知障碍或运动障碍）、严重性（失明、色盲或斜视）和程度（完全或部分）上显著不同。在如此不同的上下文中，”所有人“是不可行的。

When applying refactoring to improve internal qualities, developers decide which transformations to apply and where, because they’re the ones benefiting from the improvement. As Brian Foote and Joseph Yoder put it, “Who better to resolve the forces impinging upon each design issue as it arises, as the person who is going to have to live with these decisions?”4 Moreover, different developers might prefer alternative solutions for the same “bad smell”(that is, the design problem that motivated the refactoring1). Following on this general philosophy, we believe that end users should be able to tailor a website’s interface for their own benefit.

当应用重构来提高内部质量时，开发者决定应用什么改变，以及在哪儿应用，因为他们受益于改进。正如Brian Foote和Joseph Yoder所说，“当问题出现时，谁能更好地解决冲击每个设计问题的压力？谁又是不得不依靠这些设计的人？4”此外，不同的开发者或许更喜欢可替代的解决方案，对于相同的“坏味道”（即诱发重构1的设计问题）。根据这个普遍的哲理，我们相信那些终端用户能够根据他们自己的利益定制网站的界面。

We propose empowering users (or close representatives) with the ability to select, in their client browser-

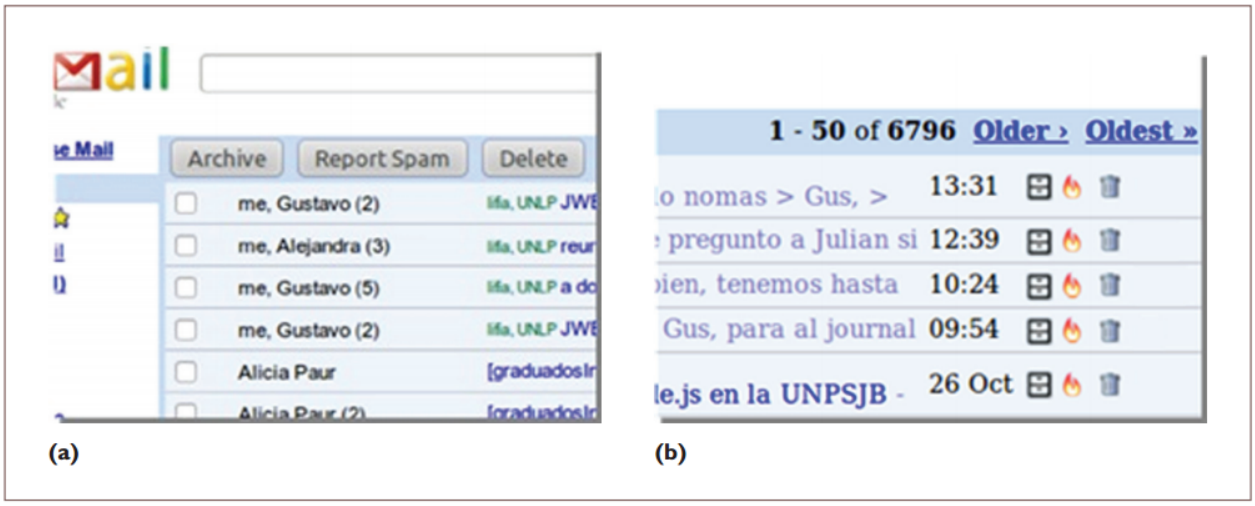

Figure 1: **Applying refactoring to Gmail. The Gmail interface (a) before and (b) after applying Distribute Global Menu to address accessibility issues with the checkbox and operations on top.(Gmail logo reprinted with permission.)对Gmail应用重构。Gmail 界面前者(a)和后者(b) 分别应用分布式全局菜单来解决复选框和顶部操作上的可达性问题。（Gmail图标已得到转载许可）**

s, their own interface refactorings for each site they access. We call our approach Client-Side Web Refactoring. CSWR allows for the automatic creation of different, personalized views of the same application to solve the particular bad smells that each user recognizes. (Developers should continue to focus on addressing general usability problems on the server-side, however,and reach the minimum level of accessibility,or “A”).
我们建议授权用户（或者是类似代表）有选择的能力，在客户端浏览器，对自己可访问的每个站点进行页面重构。我们称我们的方法为“客户端Web重构”（简称CSWR）。CSWR允许自动创建同一个应用程序的不同的、个性化界面来解决每个用户认为的特定的坏味道。（开发者应该继续致力于解决服务器端的一般的可用性问题，而且达到可访问性的最低水平或“A”）。
Here, we describe CSWR and a case study we ran with visually impaired users (though a similar solution could be applied for other disabilities).
在这里，我们描述CSWR和一个视障人士参与的实例研究（尽管类似的解决方案可应用于其他残障人士）。

2 Refactoring Example 重构示例

Figure 1a shows the inbox in Gmail, Google’s email reader. Gmail includes checkboxes on the left that let users select several emails, which is handy for applying an action to all of them.However, for visually impaired people using a screen reader, it’s uncomfortable to have to go back to the checkbox at the line’s beginning to select emails after reading the line, and then go back to the top to apply an operation after selecting the emails; they report this as a “bad accessibility smell”(an accessibility problem that motivates a refactoring).
图1a显示的是Gmail（Google的邮箱阅读器）的收信箱。Gmail包括左侧的复选框，可以让用户选择多个邮件，这很方便对所有邮件进行操作。然而，对于使用屏幕阅读器的视障人士，不幸的是观众读完阅读栏之后不得不返回到阅读栏的顶部复选框去选择邮件，并且选择完邮件之后又不得不返回到顶部去应用此操作；他们报告这种现象为“坏可闻”（诱发重构的一个可访问性问题）。
A refactoring that solves this bad smell is Distribute Global Menu, which distributes a menu of actions affecting a list of elements to each element individually. This eases the local application of an operation because it requires only a single click immediately after the element is read. Figure 1b shows the result of applying this refactoring to the Gmail inbox.The set of actions was removed from the top and attached to each email in the form of icons(each with an alternative text).
诱发这种坏味道的一个重构是分布式全局菜单，它分配一个单独影响每个元素的元素列表菜单。这使一个操作的本地应用程序变得更轻松，因为当一个元素

被阅读后需要仅仅一个简单的立即点击。图1b 显示对Gmai收信箱应用这个重构的结果。这一系列动作是从顶部移走并且链接每个图标形式的邮件（每个对应一个可替代文本）。
However, some users who report the same bad smell are more comfortable using contextual menus, so the set of actions isn’t read with every email. For them, the Contextualize Global Menu refactoring is more appropriate. Also,experienced users prefer to keep the global menu so they can operate on several emails at once; for them, using the Postpone Selection refactoring will move the checkboxes to the last column.
然而，一些报告类似坏味道的用户能够更舒服地使用上下文菜单，所以这一系列动作不能被每个邮件读到。对于他们，上下文全局菜单重构更合适。而且有经验的用户更喜欢保持全局菜单，所以他们能一次操作多封邮件；对于他们，用推迟选择重构将移动复选框到最后一列。

3 Client-Side Web Refactoring 客户端的Web重构

As this example shows, a Web refactoring changes a Web application’s navigation structure or look and feel, preserving its content and operations while removing bad usability or accessibility smells. In previous work, we used Web refactorings to enhance navigation and presentation during the development life cycle. 2,5 We can generate a complete new version of an application with a specific aim －such as a mobile version －by systematically applying and composing refactorings. Here, we propose a similar approach to improve accessibility, where refactoring is applied after deployment and during actual use of the Web application, altering the interface in the browser itself.
正如这个例子所述，Web重构改变Web应用程序的导航结构或外观,去除可用性和可访问性的坏气味的同时保留它的内容和操作。在之前的工作中，我们用Web重构增强了开发生命周期2,5 中的导航和演示。我们能通过系统应用和组合重构，根据特定的目标生成一个应用程序的全新版本–例如一个手机版本。在这里，我们提出一个相似的方法提高可访问性，即Web 应用程序部署和实际应用期间使用重构，改变浏览器本身的界面。
Our CSWR approach has two key benefits:
我们CSWR方法有两个主要好处：
? Simpler maintenance －developers maintain a single core application, applying Web usability refactorings that address a general audience, while different refactored versions can be created by and for different users.
? 维护更简单－开发者维护一个单核心应用程序，其为解决普通观众问题应用Web 可用性重构，当然不同的重构版本被创建且服务于不同的用户。
? Architecture independence －developing a CSWR requires little (if any) knowledge of the target application’s underlying architecture.
? 结构独立－开发一个CSWR需要很少（如果有的话）掌握目标应用程序的底层结构。

The engine behind CSWRs uses a client-side adaptation framework that aims to adapt existing applications by changing their DOM structure. 6 CSWRs are implemented by specializing the class AbstractRefactoring (provided in our framework) and redefining the method adaptDocument() with the refactoring’s mechanics.For example, consider the Split Page refactoring, which solves the problem of a saturated, complex webpage by dividing it into a set of simpler pages or sections. 2 In this case, the method adaptDocument() receives the DOM elements that represent disjoint page sections as parameters and creates a new page for each section, replacing the original page’s contents with an index to the new pages (see the bottom of Figure 2a).
CSMRs背后的引擎使用一个客户端的适应框架，旨在通过改变其DOM结构来适应现用的应用程序6。CSMRs通过实现类AbstractRefactoring（由我们的框架提供）并且用重构机制重定义方法adaptDocument()来实施。例如，考虑拆分页面重构，它通过将其划分成一组简单的页或段解决一个饱和、复杂页面的问题。在这种情况下，方法saturated()接收代表不相关页面部分的DOM 参数作为参数，并且创建为每个部分创建一个新的页面，使用新页面索引代替原页面的内容(参加图2a的底部)。
To apply the Split Page refactoring to a specific page, you must first create an instance of SplitPage (such as the two instances in the middle level of Figure 2a), passing as parameters
应用拆分页面重构到一个特定页面，你必须首先创建一个SplitPage实例(如图2a 中部的两个实例)来传递参数
? a URL identifying the target page or a URI pattern for a set of pages of the same site(such as all Gmail pages in Figure 2a’s code);
? 一个识别目标页的URL或一组相同站点页面的URI模式（如图2a代码中的所有Gmail 页面）and
和
? instances of the class Section , which contain DOM elements specified through xPath expressions. Because DOM elements might not always be identified through xPath queries based on attributes, absolute xPath expressions from the DOM root might be

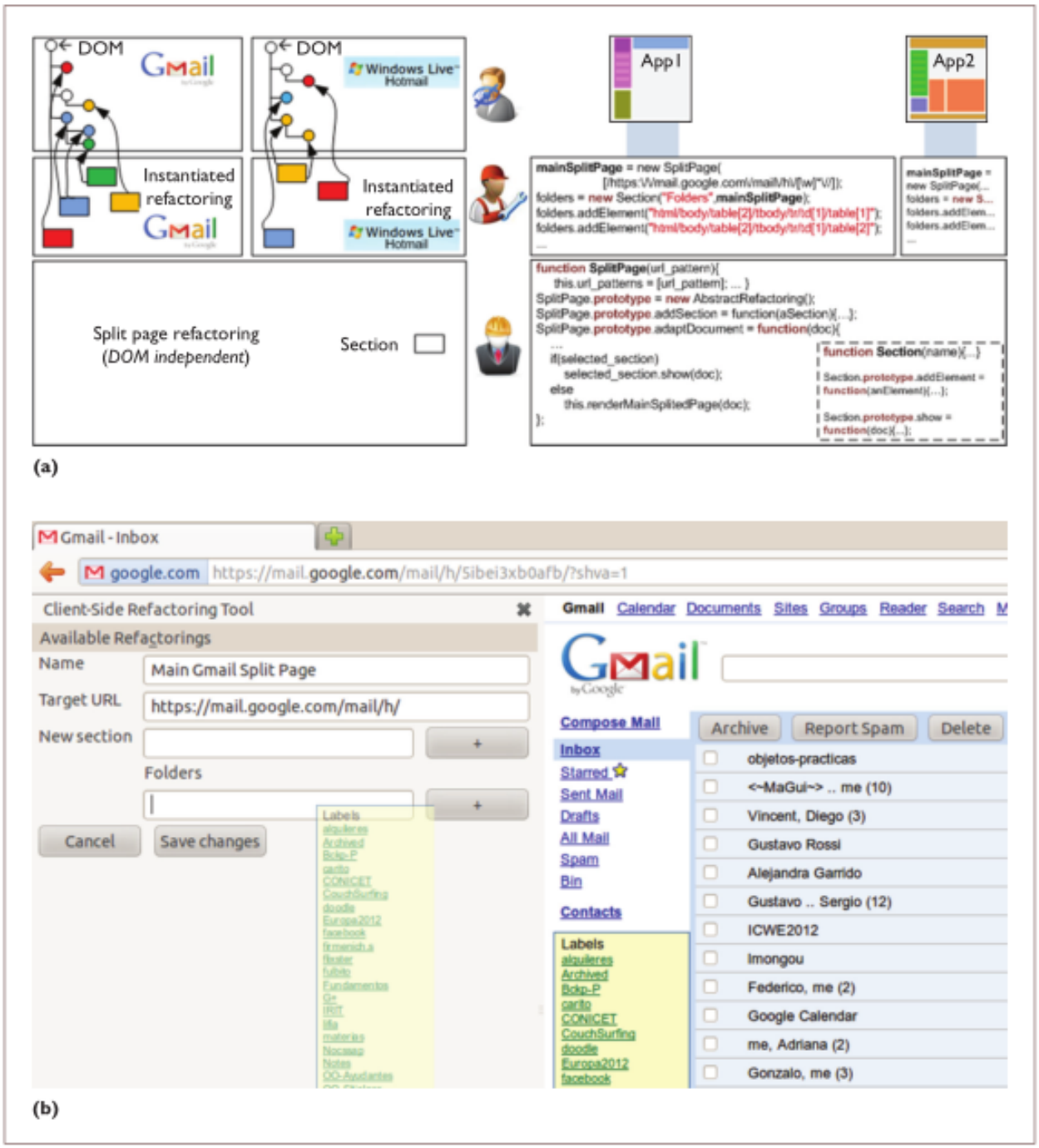

Figure 2: **Client-Side Web Refactoring. a) Split Page refactoring levels: the generic implementation at the bottom, two instances (for Gmail and Hotmail) in the middle, and the result on each DOM at the top. b) Split Page instantiation: the intermediary drags Gmail's Labels to create a new Foders section.客户端网络重构。a) 分页重构层次：底部是类的实现，中间是两个实例（对Gmail和Hotmail），顶部是每个DOM结果。b)分页实例：中介拖动Gmail 标签来创建一个新的目录部分。**

required.
? 类部分的实例，其包括通过xPath表达式指定的DOM元素。因为DOM元素可能并不总是通过基于属性识别的xPath查询指定的，来自DOM根的绝对xPath表达式或许被需要。
The three levels in Figure 2a correspond to the three steps involved in the refactoring process, which can involve three user roles (pictured in the middle column):
图2a中的三个等级对应重构过程中的三个步骤，其包括三个用户角色（图中中间列）：
? The JavaScript (JS) programmer creates new refactorings by writing a parameterized script that reuses components offered by our framework's refactoring engine.
JavaScript(JS)程序员通过写一个参数化脚本创建新的重构，该脚本重用被我们框架的重构引擎提供的组件。
? An intermediary instantiates refactorings for a specific website. (The JS programmer can play this role as well). The refactorings are instantiated either by writing code (as in Figure 2a) or using our refactoring tool (as in Figure 2b). This graphical tool lets the intermediary point-and-click on the target page to select the components that act as values for each refactoring parameter. Figure 2b,for example, shows an instantiation of SplitPage, where the group of email labels is dragged to a Folders section that will go into a new page.
? 一个为特定网站的中介实例化重构。（JS程序员也能扮演这个角色。）重构可通过写代码（如图2a）或用我们的重构工具（如图2b）实现实例化。这个图形化工具让目标页上的中介点击来选择作为每个重构参数值的组件。例如，图2b显示SplitPage的一个实例，这里邮件标签组被拖到进入新一页的目录部分。
? End users install instantiated refactorings by choosing them from an accessible menu in their Web browsers; from then on, they can access the refactored website configured to their needs. Unlike traditional refactoring, we added this new step (separating it from instantiation) so that handicapped users unable to code a script or use a graphical refactoring tool can still be part of the process by choosing their own refactorings.
? 最终用户通过从他们Web浏览器中的可访问菜单选择来安装实例化重构；从此，他们可以访问按他们需要配置的重构网站。不同于传统的重构，我们添加新步骤（从实例中分离）使不能写脚本或用图形化重构工具的残疾用户也能通过选择他们自己的重构来成为过程的一部分。
CSWR's power comes not only from letting end users choose specific refactorings, but also from letting intermediaries compose refactorings in different ways. CSWR composition is done at instantiation time, and requires special handling because refactorings can interfere with each other. Similar to code refactorings, an applied refactoring might invalidate the next refactoring's preconditions. In this case, the preconditions of CSWRs are the existence of the DOM elements specified as parameters (xPath expressions). Thus, an applied refactoring might invalidate subsequent refactorings if it changes the xPaths that identify their target elements. For example, if Split Page and Distribute Global Menu are both instantiated on the original DOM's elements, and Split Page is applied first,the Distributed Global Menu won't find its target elements in their original xPath location.
CSWR的动力不仅仅来自于让最终用户选择特定重构，也来自于让中介用不同的方式组合重构。CSWR在实例化时候完成组合，并且需要特殊处理，因为重构能互相干扰。类似于代码重构，一个应用重构或许在下一个重构的前提下无效。在这种情况下，CSWR的前提是被指定为参数（xPath表达式）的DOM元素实体。所以，如果应用重构改变识别目标元素的xPath,它或许会使后续重构无效。例如，如果拆分页面和分布式全局菜单都在原来的DOM元素上实例化，并且拆分页面先被应用，分布式全局菜单不会在原来的xPath定位式中找到目标元素。
Our refactoring tool helps intermediary users correctly compose CSWRs so that they can create and distribute a complete, accessible version of an application as a composition of CSWR instances. When an intermediary user selects several refactorings to compose, the tool creates and suggests a possible sequence, first by placing structural refactorings (such as Split Page),then refactorings that adapt specific DOM elements (such as Distribute Global Menu), and finally by placing DOM-independent refactorings (such as Replace Image with their Alt Text).CSWRs are thus instantiated in order, and users can specify certain noninterfering CSWRs to be independent. With this information, the tool creates the menu of optional CSWRs, so that when end users install a composed set of CSWRs, they can still choose to activate or deactivate independent CSWRs individually.
我们的重构工具帮助中介用户正确地构成CSWR以至于他们能创建和分配一个应用程序完整的、可访问的版本作为CSWR实例组合。当一个中介用户选择若干重构去组合的时候，工具创建并推荐一个可能的序列，首先实行结构重构（如拆分页面），然后重构以适应特定的DOM 元素（如分布式全局菜单），最后实行DOM独立重构（如用他们的Alt文本替换图像）。CSWR 因此被按顺序实例化，用户能指定特定互不干扰的CSWR进行独立化。有了这些信息，该工具创建可选的CSWR 菜单，这样当最终用户安装一套CSWR组合是，他们仍然可以单独地选择启动或

顶用独立的CSWR。

4 Case Study:Accessible Gmail实例研究:可访问性Gmail

We carried out a case study on the HTML version of Gmail with visually impaired users to validate our claim that refactorings offer a better experience when they’re customized for and by end users to suit their own expertise, screen-reader of choice, personal preferences, and so on. We played the roles of JS programmers and intermediaries.
我们开展了对视障用户的Gmail的HTML版本为例来验证我们的要求，重构提供了一个更好的体验，当他们与最终用户定制设和自己的专长、屏幕阅读器的选择、个人喜好等等。我们扮演JS 程序员和中介机构的角色。

Preliminary Study初步研究

We first conducted a study with seven potential Gmail users to test our preliminary hypothesis:our refactored Gmail version is more accessible and usable than the original. Of the seven users, six were blind, and one had a severe sight deficiency. The test users had various computer and Internet skill levels; most used email clients and Web browsers, although only one had previously used webmail tools.
我们首先开展一项有七个潜在Gmail用户参与的研究来测试我们的初步假设：我们重构的Gmail 版本比原来的更方便和实用。在七个用户中，六个是失明的，一个有严重的视力缺陷。测试用户有各种电脑和Internet技能水平；最常用的邮件客户端和Web 浏览器，尽管只有一个以前使用过邮件工具。
We gave users the following tasks:
我们给用户以下任务：
1. Read and reply to an email.
1. 阅读和回复一封邮件。
2. Compose an email and send it to several people.
2. 撰写一封邮件并将它发给几个人。
3. Search and delete a specific sent message.
3. 搜索并删除一封特定的发送信息。

Users had to complete the tasks first in the original Gmail, so we could check for ignored bad smells, and then in a refactored version,to detect unsolved bad smells and determine how this first refactoring attempt improved or spoiled the user experience.
用户必须首先在原来的Gmail上完成任务，所以我们能检查忽略的坏气味，然后在一个重构版本中，检测解决的坏气味并决定第一次重构尝试如何改进或破坏用户体验。
The selected refactorings were Split Page (to partition the email list, tags list, and the main Google menu), Distribute Global Menu (for email operations), Reorganize into a List (for sets of unnumbered actions), Remove Redundant Operation (to remove the global menu of email operations at the top and leave the one at the bottom of the email list), and Postpone Selection (to move the checkbox column to the end of each email’s row).
选定的重构是拆分页面（为分区的邮件列表，标签列表和主要的Google菜单）,分布式全局菜单（为邮件操作），整理成列表（为无编号的操作组），去除多余的操作（删除顶部邮件操作的全局菜单及保留在邮件列表底部）和推迟选择（移动复选框列到每个邮件行尾）。
During the study, we used observation and questionnaires to gather various feedback. These led to several findings:
在研究期间，我们用观察和问卷调查收集各种反馈。这产生了一些发现：
? Different skill levels in screen reader use led to different complications; for example,refactorings aimed at simplifying the structure were useful mostly for novice users, but were burdensome for experienced ones.
? 在屏幕阅读器使用方面不同的技能水平会导致不同的混乱；例如，旨在简化结构的重构对于新手用户非常有用，但是对于有经验的用户却是繁重的。
? Users suggested new refactorings, including the use of context menus as an alternative to distributed menus.
? 用户建议新的重构，要包含使用上下文菜单来代替分布式菜单。
? Users new to webmail clients couldn’t complete the tasks in the original Gmail, but could complete them with external aid in the refactored version (which proves our hypothesis with new users).
? 使用网页邮箱客户端的新用户不能在原来的Gmail完成任务，但是能在重构版本中外部援助情况下完成（这证明了我们在新用户方面的假设）。
? Users had positive feedback on the content’s organization and functionality. Using a ratings scale of very good, good, average, and poor, the feedback included five good qualifications and one average for organization/functionality, but ease of use scored only two goods and four averages, which helped us gather other bad smells and improve the tools for intermediaries and final users.
? 用户在内容的组织和功能方面有积极的反馈。用非常好、好、一般、差的评级量表，在组织/功能方面得到五个‘好’和一个‘一般’的反馈，但是在易用性方面得分仅有两个‘好’和四个‘一般’，这帮助我们收集其他坏味道，并且提高为中介和最终用户的工具。

Actual Experiment实际实验

Following our preliminary study, we conducted an

experiment with new users. Of the 10 blind users in this study, three were experienced in operating Gmail, and the other seven were experienced in Web browsing, but not with Gmail.This time, our hypothesis was that a personalized version of Gmail is better than our completely refactored version. We conducted the test with one user at a time, going through simplified tasks that covered the same basic ground as in our previous study:
根据我们的初步研究，我们进行了一个新用户的实验。本研究中的10个盲人用户，其中3个在操作Gmail上有经验，其他7个在Web浏览器上有经验，但对于Gmail没有经验。这次，我们的假设是一个个性化的Gmail版本比我们完全重构的版本好。我们一次一个用户地进行测试，覆盖以前研究中的同样基础测试组的简化任务：
1. Delete all emails from a specific sender.
1. 删除特定发送者的所有邮件。
2. Find a specific deleted email in the Trash and put it back in the Inbox.
2. 在垃圾箱中找到一个特定的已删除邮件，并把它放回收信箱。
3. Answer the email recovered in task 2.
3. 回复任务2中恢复的邮件。
Before the actual experiment, we asked users to answer a specific email (as in task 3) on the original Gmail. This had a twofold purpose:it gave us a sense of their expertise with the tools (browser and screen reader) and reduced the bias that might occur for users without previous Gmail experience.
在实际实验之前，我们让用户在原来的Gmail中回复一个特定的邮件（如在任务3 中）。这有双重目的：它让我们了解他们使用工具（浏览器和屏幕阅读器）的技能，并且减少没有Gmail经验的用户可能产生的偏差。
For the main part of the experiment, we devised an optimal set of four refactorings based on the experience from the previous study and applied them to Gmail:
对于实验的主要部分，我们设计了一个基于以前研究经验的四种重构的最优组合，并将其应用在Gmail上：
? Split Page, to reduce each page’s contents and thus ease content access;
? 分页，来减少每个页面的内容，从而缓解内容访问;
? Distribute Menu, to simplify the tasks applied to each item on a list (such as emails);
? 分布式菜单，来简化应用在列表（如邮件）中每个列表项的任务;
? Contextualize Menu, to present actions over an item as a contextual menu; and
? 上下文适应菜单，将一个列表项的当前行为作为一个上下文菜单；并且
? Postpone Selection, to let users read the email subjects and check them immediately afterward to apply an action to several selected emails.
? 推迟选择，让用户阅读邮件主题并立即检查它们之后对一些选定的邮件申请动作。
Before we asked the subjects to complete the tasks, we explained each refactoring. Once the users were finished with the tasks in the completely refactored version, we had them arrange their own set of refactorings using the menu options in their browsers. They then performed the affected tasks —those related to the selected refactorings —again for further comparison.
在我们要求测试者完成任务之前，我们解释了每个重构。一旦用户在完全重构版本完成任务，我们让它们安装它们自己的重构组合，其应用在它们浏览器上的菜单选项。然后他们再次执行受影响的任务- 那些与选择的重构相关的任务- 再做进一步比较。

Results结果

The main measurement we gathered was task completion time, comparing the times from the completely refactored site with those repeated in the personalized version (see Table 1). Out of 10 users, five preferred Contextualize Menu to Distribute Menu, two discarded Split Page, and the rest preferred the refactored site as it was.
我们收集的主要测试法是任务完成时间，比较完全重构版本的时间与那些在个性化版本重复花的时间（如表1）。在这10个用户中，5个比分布式菜单更喜欢上下文适应菜单，2个丢弃拆分页面，而剩余的更喜欢它本身的重构网站。
The overall completion times decreased by an average of 33.44 percent, with 32.03 percent for the subjects who chose Contextualize Menu and 36.94 percent for the group with no Split Page refactoring.
整体完成时间平均减少33.44%，选择上下文适应菜单科目的减少32.03%，没有拆分页面重构的组减少36.94%。
From this second study, we gathered new feedback, which led to the following findings:
从这第二次研究，我们收集新回馈，其产生以下的发现：
? Novice users found it easier to navigate using Split Page, but this wasn’t the case for experienced users because the split structure requires additional navigation steps for some tasks (such as when folders were moved to a folder index in a separate page).
? 新手用户发现使用拆分页面导航更容易，但是这不适用于有经验的用户，因为拆分结构下一些任务需要额外的导航步骤（例如当目录被移动到一个单独的页面目录索引时）。
? Some novice users suggested splitting the pages in a new way, so that some of the features were always

Table 1. Results comparing completely refactored vs. personalized versions of Gmail.

User	Selected refactorings	Completely refactored (secs)	Personalized (secs)	Drop rate (%)
1	SplitPage, ContextualizeMenu, Postpone Selection	180	160	11.11
2	SplitPage, ContextualizeMenu, Postpone Selection	300	200	33.33
3	SplitPage, ContextualizeMenu, Postpone Selection	143	43	69.93
4	SplitPage, ContextualizeMenu, Postpone Selection	91	52	42.86
5	SplitPage, ContextualizeMenu, Postpone Selection	68	66	2.94
			Partial	32.03
6	DistributeMenu, Postpone Selection	90	65	27.78
7	DistributeMenu, Postpone Selection	180	97	46.11
			Partial	36.94
			Overall	**33.44**

present; others wanted to easily hide the main menu when desired.
? 一些新手用户建议用一种新的方式拆分页面，这样一些功能总是存在的；其他人想根据所需更容易地隐藏主菜单。
? Users’ habits directly interfere with the results.For example, experienced users didn’t appreciate the benefits of Distribute Menu until after they tried it and used it for a while, because they were used to dealing with the global menu.
? 用户的习惯直接干扰结果。例如，有经验的用户直到他们尝试并使用它一段时间以后才能领会分布式菜单的益处，因为他们习惯处理全局菜单。
These results clearly show the importance of personalization: because experienced users can move quickly through a page with keyboard combinations, they prefer loaded pages and shorter navigation paths －a solution that frustrates inexperienced users because it demands going through lots of content every time the page reloads.
这些结果清楚地显示出个性化的重要性：因为有经验的用户能使用键盘组合快速地浏览页面，他们更喜欢加载页面和更简短的导航路径－一个阻碍有经验用户的解决方案，因为每当页面重加载都需要很多内容。

Discussion论述

In previous work, 6 we developed tools that let users create conceptual models and then define adaptations in terms of these models, based on the idea of ModdingInterface. 7 We’re now planning to adapt this conceptual layer specifically for use with CSWR. This new abstraction level would let developers define concepts (and their properties) over DOM elements and define the adaptations in terms of concepts, instead of manipulating DOM elements directly with xPath expressions. Thus, if two Web applications manage the same concepts －and thereby form an application family that shares the same abstract model －the CSWRs defined in terms of abstract concepts can be applied to both applications.
在以前的工作中6，我们开发工具让用户创建概念模型，然后基于ModdingInterface 理念7，定义在这些模型的适应性。我们现在正在规划为使用CSWR的用户专门适应这一概念。这个新的抽象层次会让开发人员在DOM元素上定义概念（及其性质），并且定义这一概念的适应性，而不是直接使用xPath表达式操纵DOM元素。所以，如果连个Web应用程序管理同一概念－从而形成共享相同抽象模型的一个应用程序家庭组－定义在这个抽象概念的CSWR能应用在这两个应用程序上。
For example, webmail applications that share the same abstract model (Inbox, Folder,Email, and so on) can use the same set of CSWRs defined in terms of these concepts. This approach not only allows more CSWR reusability, but it might improve script resilience. Such resilience is one of the most important drawbacks for client-side scripting, and our approach isn’t exempt: when webpage DOMs change,scripts might stop working. If the development uses an agile process, server-side refactorings might update the DOM often. Another common constraint in this type of technology is that it’s not applicable to all websites; for example,sites developed using technologies such as Flash might present problems.
例如，共享同一抽象模型的邮件应用程序（收件箱、文件夹、电子邮件等等）能用同一组定义在这个抽象概念的CSWRs，但是它或许提高脚本弹性。这样的

弹性是客户端脚本语言最重要的劣势之一，我们的方法也没有豁免：当网页的DOM发生变化时，脚本或许停止工作。如果开发者使用一个敏捷过程，服务器端重构会经常更新DOM。这种技术的另一个常见限制是他不适用于所有网站；例如，使用诸如Flash技术开发的网站可能产生问题。

Although we propose CSWR to improve accessibility for unsighted users, developers can easily apply the same approach to create different views of a Web application targeted to improve other external qualities or to create, for example, a mobile version. Note that W3C guidelines for both accessibility (Web Content Accessibility Guidelines; www.w3.org/TR/WCAG10) and mobile (Mobile Web Best Practices; www.w3.org/TR/mobile-bp) have several similarities, which can be implemented as CSWR if they aren't contemplated originally by Web applications.

尽管我们提出CSWR来提高视障用户的可访问性，开发者可以很容易采用同样的方法来创建Web应用程序的不同视图，其目的是提高其他外部品质或创建诸如手机版本之类的。注意W3C指南在可访问性（Web内容可访问性指南；www.w3.org/TR/WCAG10）和移动（移动互联网最佳实践；www.w3.org/TR/mobile-bp）上有很多相似之处，如果他们起初没考虑在Web应用程序上，这也能实现为CSWR。

Refactoring is a powerful and essential tool that lets developers improve running applications based on feedback. This feedback might come from bad smells in the code identified by developers, or from bad smells in usability and accessibility experienced by users. However, for developers to correct bad smells based on user feedback typically takes a long time —especially for bad accessibility smells, which generally aren't a priority. We thus put refactoring in the hands of users, who know better what they actually need. This not only lets users customize specific interaction improvements, but also removes such improvements from the main development cycle of the applications themselves, which reduces cost and effort.

重构是一个强大的、必不可少的工具，其让开发人员基于基于反馈提高运行的应用程序。这种反馈可能来自开发人员确定的代码中的坏味道，也可能来自于用户体验中发现的易用性和可访问性的坏味道。然而开发人员根据用户反馈纠正坏味道，特别是可访问性的坏味道，通常要花费很长时间，这通常不是一个优先的方法。所以我们让用户掌握重构，他们更清楚他们真正想要的东西。这不仅仅让用户定制特定的相互改善方案，而且消除了来自应用程序本身的主要开发周期的改善方案，从而降低了成本和精力。

Web refactorings are a technically compelling way to dynamically improve users' experience,as they are composable and let users create different application versions without any knowledge of the internal design. This is a huge benefit because it also allows a crowdsourcing approach to making CSWRs and their compositions available. Indeed, our future work includes building a crowdsourcing tool for volunteers to upload new generic refactorings or instantiate existing refactorings for a particular website, which could come as a package of composed refactorings to create a completely new version of a site. We propose hosting crowdsourced CSWRs to spread their adoption with the least possible burden for end users. Moreover, to overcome the existence of different versions of refactorings in response to webpages' DOM evolution, our crowdsourcing tool's architecture would automatically select the latest versions of a given CSWR or CSWR set.

Web重构是动态改善用户体验的一个技术上不可抗拒的方法，因为它们是可重组的，并且让用户在不知道内部设计的情况下创建不同的应用程序版本。这有很大益处，因为它也允许一个使CSWR及其组合体可用的众包方法。当然，我们将来的工作会包括为志愿者创建一个众包工具，为一个特定网站加载新的公共重构或包含重构的实例，其包括一个组合重构包来创建一个网站的全新版本。我们建议拥有众包CSWRs以便为最终用户以尽可能小的负担来扩展使用。而且，为了克服因应对网页DOM演变造成存在不同重构版本的问题，我们的众包工具的结构能自动选择给定的CSWR或CSWR组的最新版本。

Related Work in Improving Web Accessibility
引：提高Web可访问性的相关工作

Ideally, accessibility should be contemplated early, during Web application design, and webpages should follow existing standards or guidelines such as Web Content Accessibility Guidelines (WCAG). Such guidelines could, for example, 1,2 be incorporated in the Web engineering life cycle. Other key approaches to ensure or enforce accessibility include systematically assessing compliance with guidelines 3 and automatically detecting accessibility problems in webpages. 4,5 Despite these research efforts, however, most Web applications aren't yet fully accessible, and the problem must be tackled with more dynamic approaches.

理想情况下，可访问性应早期考虑，在Web应用程序设计中，并且网页应该遵循已有的标准和规范，诸如Web内容可访问性指南（WCAG）。例如1,2这样的指南能奶乳Web 工程生命周期中。其他其他保证或强制执行可访问性的关键途径包括系统性评估指南规范3和自动检测网页的可访问性问题4,5。尽管存在

这些研究方法，但大多数Web应用程序还没有完全可访问，而这个问题必须用更加动态的方法来处理。

A well-known technique to transform existing webpages to be accessible is transcoding, 6 which applies transformations on the fly, based on semantic annotations manually added by developers or automatically derived from Web design models. Transcodings can be applied on the server, the client, or a proxy. 6 Our approach shares the philosophy behind transcoding, but most of the existing transcoding systems for accessibility lack extensibility and personalization:

一个改造现有网页可访问性的著名技术是转码，其适用于动态转换，基于由开发者手动添加或自动从Web设计模型中获取的语义注释。转码可以应用于服务器、客户端或代理6。我们的方法分享转码背后的哲学，但是大多数现有的用于可访问性的转码系统缺乏可扩展性和个性化。

? All transcoding methods (including Text Magnification and Content Reorder) 6 are predefined by their developers, and it isn't possible to add new transcoding methods. Most such systems are only extensible – in terms of which webpages will be transcoded – if volunteers are allowed to annotate a website and apply the transcodings for future visitors.Our Client-Side Web Refactoring (CSWR) approach allows for a new type of volunteer (JavaScript programmer) who can add new refactorings in response to new bad smells or tackle the same bad smell in a different way.

? 所有的转码方法（包括文字放大和内容排序6）都是由开发者预先定义的，而且它不可能添加新的转码方法。大多数这样的系统仅仅是可扩展的 - 根据什么样的网页将被转码 - 如果志愿者被允许注释网站和为未来参观者引用转码的话。我们的客户端Web重构（CSWR）方法允许一个新类型的志愿者（JavaScript程序员），其可以添加新的重构来应对新的坏味道或者用不同的方法处理同样的坏味道。

? Transcoding-aware annotations have the same impact for all users regardless of their special capacities. Because transcodings are considered transparent from the users' viewpoint, it isn't possible to fine tune them for a specific webpage according to each user. With CSWR, each user can select a different set of refactorings for each website.

? 转码意识的注释对所有用户有相同的影响，不管他们有什么特殊技能。因为转码从用户角度是显而易见的，它不可能根据每个用户对一个特定的网页适当微调。有了CSWR，每个用户能为每个网站选择一个不同的重构组。

? Transcodings don't necessarily preserve behavior; they can remove some operations, such as when they aim to simplify content. In contrast, refactorings were conceived as behavior-preserving transformations, 7 which, in the case of Web applications, means preserving content and functionality. 8

? 转码不一定保留行为；我们不能删除一些操作，例如当他们旨在简化内容的时候。相比之下，重构被视为行为保持的转换7，其在Web应用程序的情况下，意味着保存内容和功能。

? Transcodings don't necessarily compose, and they might even interfere with each other. 6 We propose CSWR composition as an additional way of customizing a website, letting users apply a sequence of refactorings incrementally.

? 转码不一定会组合，甚至会相互干扰6。我们提出CSWR组合作为另一种定制网站的方法，让用户增量地应用一系列重构。

Interest is growing in client-side scripting 9 for customizing existing pages, as proven by large communities using GreaseMonkey (`www.greasespot.net`), a popular tool for client-side scripting that allows any kind of change over a webpage's DOM.Specific tools such as WebAdapt2Me (`http://www-03.ibm.com/able/accessibility_services/WebAdapt2Me.html`) and AccessMonkey10 focus on accessibility. However, both tools only let users make basic changes to style, such as font size or color,and basic changes to content order.

兴趣在定制现有网页的客户端脚本上不断增加，其被使用GreaseMonkey(`www.greasespot.net`)的大型机构证实，其是允许任何网页DOM变化的客户端脚本的一种流行工具。特定的工具诸如WebAdapt2Me（`http://www-03.ibm.com/able/accessibility_services/WebAdapt2Me.html`）和专注于可访问性的AccessMonkey10。然而，这两个工具仅仅让用户做出风格上的基本改变，如字体大小或颜色和内容顺序上的基本改变。

When it comes to accessibility improvements, current client-side tools are too primitive. Generic tools such as GreaseMonkey hardly provide mechanisms for script compatibility;when different scripts are applied over the same pages, the execution of one script can spoil the previous one's changes or invalidate the execution of the script that follows. Besides,while GreaseMonkey lets users adapt a specific page, it doesn't let them generalize – that is, they can't apply the same change on different pages if changes depend on the DOM's structure. Although GreaseMonkey is excellent as a weaver, it doesn't offer facilities for accessibility. In contrast, our tool is a weaver that further provides mechanisms for refactoring definition, composition, and installation. Tools designed specifically for accessibility based on client-side scripting, such as AccessMonkey, also have several limitations, mainly because they're focused on basic style changes, which are usually insufficient to solve problems such as user disorientation or long navigation chains.

当涉及到可访问性的改进时，目前的客户端工具太落后。诸如GreaseMonkey的通用工具不提供脚本兼容机制；当不同的脚本被用在同一网页时，一个脚本的执行可以破坏接下来的脚本。而且，当GreaseMonkey让用户适应一个特定页面时，它不它们推广 - 也就是说，如果改变取决于DOM结构时，它们不能在不同的页面上应用同样的改变。尽管GreaseMonkey像织工一样出色，它也不能提供无障碍设施。相比之下，我们的工具是进一步提供重构定义、组成和安装机制的织工。基于客户端脚本、专为可访问性设计的工具，如AccessMonkey，也有一些局限性，主要因为它专注于简单风格变化，这通常不足以解决诸如用户导航迷失或长导航链的问题。

References参考文献 1. V. Luque Centeno et al., “Web Composition with WCAG in Mind,” Proc. Int’l Cross-Disciplinary Workshop on Web Accessibility (W4A), ACM, 2005, pp. 38 - 45.
2. P. Plessers et al., “Accessibility: A Web Engineering Approach,” Proc. 14th Int’l Conf. World Wide Web, ACM, 2005, pp. 353 - 362.
3. J. Vanderdonckt, A. Beirekdar, and M. Noirhomme-Fraiture, “Automated Evaluation of Web Usability and Accessibility by Guideline Review,” Proc. 4th Int’l Conf. Web Engineering, LNCS 3140, Springer, 2004, pp. 17 - 30.
4. C. Benavídez et al., “Semi-Automatic Evaluation of Web Accessibility with HERA 2.0,” Proc. Int’l Conf. Computers Helping People with Special Needs, LNCS 4061, Springer, 2006, pp. 199 - 206.
5. TAW3: Tool for the Analysis of Websites, Fundación CTIC, Spanish Ministry of Employment and Social Affairs (IMSERSO) Online Web Accessibility Test; www.tawdis.net.
6. C. Asakawa and H. Takagi, “Transcoding,” Web Accessibility: A Foundation for Research, S. Harper and Y. Yesilada, eds., Springer, 2008, pp. 231 - 261.
7. M. Fowler, Refactoring: Improving the Design of Existing Code, Addison Wesley, 1999.
8. A. Garrido, G. Rossi, and D. Distante, “Refactoring for Usability in Web Applications,” IEEE Software, vol. 3, no. 28, 2011, pp. 60 - 67. 9. O. Diaz, C. Arellano, and J. Iturrioz, “Layman Tuning of Websites: Facing Change Resilience,” Proc. 17th Int’l Conf. World Wide Web (WWW), 2008, ACM, pp. 127 - 128.
10. J. Bigham and R. Ladner, “Accessmonkey: A Collaborative Scripting Framework for Web Users and Developers,” Proc. Int’l Cross-Disciplinary Conf. Web Accessibility (W4A), ACM, 2007, pp. 25 - 34.

References参考文献 1. M. Fowler, Refactoring: Improving the Design of Existing Code, Addison Wesley, 1999.
2. A. Garrido, G. Rossi, and D. Distante, “Refactoring for Usability in Web Applications,” IEEE Software, vol. 3, no. 28, 2011, pp. 60 - 67.
3. N. Medina-Medina et al., “Refactoring for Accessibility in Web Applications,” Proc. 11th Int’l Conf. Interacción Persona-Ordenador, Assoc. Interacción Persona-Ordenador, 2012, pp. 427 - 430; www.aipo.es/ items.php?id=364.
4. B. Foote and J. Yoder, “Big Balls of Mud,” Pattern Languages of Program Design 4, N. Harrison, B. Foote, and H. Rohnert, eds., Addison Wesley, 2000, pp. 653-692.
5. N. Medina-Medina et al., “An Incremental Approach for Building Accessible and Usable Web Applications,” Proc. 11th Int’l Conf. Web Information System Eng.(WISE), Springer, 2010, pp. 564 - 577.
6. S. Firmenich et al., “A Crowdsourced Approach for Concern-Sensitive Integration of Information across the Web,” J. Web Engineering, vol. 10, no. 4, 2011, pp. 289 - 315.
7. O. Diaz, C. Arellano, and J. Iturrioz, “Layman Tuning of Websites: Facing Change Resilience,” Proc.17th Int’l Conf. World Wide Web (WWW), ACM, 2008,pp. 127 - 128.

Alejandra Garrido is an assistant professor at Facultad de Informática, Universidad Nacional de La Plata, Argentina, where she’s a member of the Research and Development in Advanced IT Lab (LIFIA). She is also a researcher at Argentina’s National Scientific and Technical Research Council (CONICET). Her research interests include refactoring and Web engineering, focusing on design patterns, frameworks, refactoring for the C language, and refactoring for usability. Garrido has a PhD in computer science from the University of Illinois at Urbana-Champaign. She’s a member of the Hillside Group. Contact her at garrido@lifia.info.unlp.edu.ar.

Sergio Firmenich is a teaching assistant at Facultad de Informática, Universidad Nacional de La Plata, Argentina, and a member of the Research and Development in Advanced IT Lab (LIFIA). His research interests focus on Web application adaptability —specifically, on engineering the adaptation of existing applications. Firmenich has a PhD in computer science from Universidad Nacional de La Plata. Contact him at firmenich@lifia.info.unlp.edu.ar.

Gustavo Rossi is a professor at Facultad de Infor-


Software
BRIDGING
SOFTWARE
COMMUN
THROUGH
SOCI
Software
TWIN PEAKS
OF REQUIREMENTS
AND ARCHITECTURE
Software
SAFETY-CRITICAL
SOFTWARE
IEEE
computer society
IEEE Software offers pioneering ideas, expert analyses, and thoughtful insights for software professionals who need to keep up with rapid technology change. It's the authority on translating software theory into practice.
www.computer.org/software/subscribe
SUBSCRIBE TODAY

mática,Universidad Nacional de La Plata, Argentina, and the director of the Research and Development in Advanced IT Lab (LIFIA). He is also a researcher at CONICET. His research interests include Web application design and agile approaches. Rossi has a PhD in informatics from the Pontifical Catholic University of Rio de Janeiro, Brazil. He' s one of the developers of the Object-Oriented Hypermedia Design Method (OOHDM) and is a member of IEEE and ACM. Contact him at gustavo@lifia.info.unlp.edu.ar.

Julián Grigera is a PhD student at Facultad de Informática, Universidad Nacional de La Plata, Argentina. His research interests are in Web development and agile methodologies, and he' s previously worked on context-aware systems architecture and sensing mechanisms,and usability and accessibility for Web applications and mobile devices. Grigera has a Licentiate degree in informatics from the University of La Plata. Contact him at juliang@lifia.info.unlp.edu.ar.

Nuria Medina-Medina is an associate professor and researcher in the Department of Computer Languages and Systems at the University of Granada, where she' s a member of the Group on Specification, Development, and Evolution of Software (GEDES). Her research interests include hypermedia systems, user modeling, user adaptation, and software evolution, as well as Web browsing, refactoring for the visually impaired, and bioinformatics. Medina-Medina has a PhD in computer science from the University of Granada. Contact her at nmedina@ugr.es.

Ivana Harari is an assistant professor and Director of Web Accessibility at the Facultad de Informática, Universidad Nacional de La Plata, Argentina. Her research interests include human - computer interaction, mobile user interface design, and Web accessibility, as well as usability engineering and testing, user-centered design, free and open source software (FOSS) tools for disabled people, and adaptive and accessible mobile interfaces. Harari has an education specialist degree in university teaching from the University of La Plata.Contact her at iharari@ada.info.unlp.ed u.ar.

cn Selected CS articles and columns are also available for free at `http://ComputingNow.computer.org`.