PeerJ Computer Science

Give the people what they want: studying end-user needs for enhancing the web

Tak Yeon Lee and Benjamin B. Bederson

Human-Computer Interaction Lab, Computer Science, University of Maryland, College Park, MD, United States

ABSTRACT

End-user programming (EUP) is a common approach for helping ordinary people create small programs for their professional or daily tasks. Since end-users may not have programming skills or strong motivation for learning them, tools should provide what end-users want with minimal costs of learning –i.e., they must decrease the barriers to entry. However, it is often hard to address these needs, especially for fast-evolving domains such as the Web. To better understand these existing and ongoing challenges, we conducted two formative studies with Web users –a semi-structured interview study, and a Wizard-of-Oz study. The interview study identifies challenges that participants have with their daily experiences on the Web. The Wizard-of-Oz study investigates how participants would naturally explain three computational tasks to an interviewer acting as a hypothetical computer agent. These studies demonstrate a disconnect between what end-users want and what existing EUP systems support, and thus open the door for a path towards better support for end user needs. In particular, our findings include: (1) analysis of challenges that end-users experience on the Web with solutions; (2) seven core functionalities of EUP for addressing these challenges; (3) characteristics of non-programmers describing three common computation tasks; (4) design implications for future EUP systems.

Subjects Human–Computer Interaction

Keywords End-user programming, User study, Wizard-of-Oz study, World-Wide Web

Submitted 22 April 2016
Accepted 15 September 2016
Published 14 November 2016

Corresponding author
Tak Yeon Lee, tylee@umd.edu

Academic editor
Harry Hochheiser

Additional Information and Declarations can be found on page 22

DOI **10.7717/peerj-cs.91**

INTRODUCTION

Over the decades, the Web has become the most popular and convenient workbench for individuals and businesses supporting an incredible number of activities. However, developers of Web services cannot completely anticipate future uses and problems at design time, when a service is developed. Thus we can expect users, at use time, will discover misalignment between their needs and the support that an existing system can provide for them (*Fischer & Giaccardi, 2006*). Numerous examples of this misalignment exist. For example, a site designed to support comparison-shopping for online shoppers may not meet the needs of shoppers who want to compare prices across different sites and even track daily prices (http://camelcamelcamel.com). Another is that people often use customizable applications (e.g., RSS feed readers) to manage ever-growing channels instead of visiting individual sites. More broadly, fraudulent sites and deceptive opinion spam are ongoing concerns for consumers (*Ott, Cardie & Hancock, 2012*). When a Web page does not match their needs, people often use mashups

(*Ennals et al., 2007*; *Wong & Hong, 2008*; *Zang, Rosson & Nasser, 2008*; *Zang & Rosson, 2009a*; *Zang & Rosson, 2009b*), browser extensions and scripts (*Bolin et al., 2005*; *Leshed et al., 2008*; *Miller et al., 2008*; https://addons.mozilla.org/en-US/firefox/addon/greasemonkey/) built by third-party programmers. Unfortunately there are not enough third-party solutions to address all 1.4 billion end-user's needs of 175 million websites (*Toomim et al., 2009*), and enabling end users to develop their own solutions is the goal of end-user programming on the Web (WebEUP).

A clear understanding of end-user needs is essential for building successful programming tools (*Rosson, Ballin & Rode, 2005*). In this paper we report two user studies that address the following related research questions: (1) What do end-users want to do; and (2) How can we make programming tools easy for end-users without programming knowledge? Answering these two questions is important to have a clearer understanding of the direction we should take to develop WebEUP systems that will be useful and effective for a broad range of people.

Prior studies (*Zang, Rosson & Nasser, 2008*; *Zang & Rosson, 2008*; *Zang & Rosson, 2009b*) characterize potential end-user programmer's mindset and needs. Researchers also investigated end-user programmer's real world behavior and software artifacts they created with specific WebEUP tools such as CoScripter (*Bogart et al., 2008*). Live collections such as the Chrome Web Store (https://chrome.google.com/webstore/category/apps) and ProgrammableWeb (http://www.programmableweb.com/) are valuable resources that address user needs by community developed scripts and mashups. This paper reports on an interview study with similar motivations—to investigate what challenges end-users experience and how they would improve—but focuses on unmet needs of 35 end-users on the Web with minimal bias of current technology. Through iterative coding we identify the pattern of challenges that end-users experience. We also suggest seven functionalities of EUP for addressing the challenges—*Modify*, *Compute*, *Interactivity*, *Gather*, *Automate*, *Store*, and *Notify*.

There is a wealth of study for the second question—how to make WebEUP tools natural and easy to learn. Researchers have studied the psychology of non-programmers. *Miller (1974)* and *Miller (1981)* examined natural language descriptions by non-programmers and identified a rich set of characteristics such as *contextual referencing*. *Biermann, Ballard & Sigmon (1983)* confirmed that there are numerous regularities in the way non-programmers describe programs. *Pane, Myers & Ratanamahatana (2001)* identified vocabulary and structure in non-programmer's description of programs. We conducted a Wizard-of-Oz study with 13 non-programmers to observe how they naturally explain common computational tasks through conversational dialogue with an intelligent agent. The interviewer acted as a hypothetical computer agent, who understands participant's verbal statements, gestures, and scribbles. This study expands existing work with characteristics of non-programmers mental models.

Findings from the interviews and the Wizard-of-Oz study together demonstrate a disconnect between what end-users need from EUP and what current systems support. In addition to identifying a set of important functionalities that should be included to best support end-users, our findings specifically highlight the needs of social platforms for

solving complex problems, and interactivity of programs created with EUP tools to alleviate end-user's concerns about using third-party programs. The Wizard-of-Oz study also shows that future EUP tools should support multi-modal and mixed-initiative interaction for making programming more natural and easy-to-use.

This paper makes the following contributions: (1) Identifying the current needs of end-user programming on the Web; (2) Proposing features of future EUP systems for addressing the needs of users on the Web; (3) Characterizing non-programmer's mental model of describing programming tasks; and (4) Proposing implications for designing end-user programming interfaces.

RELATED WORK

End-user programming on the web

Over the last decade, researchers and companies have developed a large number of EUP systems for the Web (WebEUP). Those WebEUP tools commonly took a combination of three approaches: scripting, visual programming, and inductive programming including programming-by-demonstration (*Cypher et al., 1993*) and programming-by-example (*Lieberman, 2001*). First, scripting languages for WebEUP (*Bolin et al., 2005*; *Leshed et al., 2008*; https://addons.mozilla.org/en-US/firefox/addon/greasemonkey/) offer natural and domain-specific commands instead of machine-oriented general-purpose syntax of traditional languages such as C or Java. However, end-users of script languages still need to memorize commands and type accurate details. In order to make programming more accessible to users without programming expertise, EUP systems often employ visual programming techniques including drag-and-drop for organizing graphical widgets of operations, and flowcharts showing program structure (*Wong & Hong, 2007*; *Cameron & Churchill, 2009*). While visual programming techniques make programming more intuitive, they are usually less expressive and scalable than scripting languages.

Recent work employs inductive program synthesis techniques—such as programming-by-example and demonstration (*Cypher et al., 1993*; *Gulwani, 2010*; *Rinard, 2012*)—that enable end-users to express their needs via demonstrations and examples of what they are trying to accomplish, and the systems generate programs that are consistent with the examples. Such programs include string manipulation (*Gulwani, 2011*), text processing (*Yessenov et al., 2013*), and geometric drawing (*Cheema, Gulwani & LaViola, 2012*). Especially for the Web, the "Reform" system (*Toomim et al., 2009*) enables end-users to attach UI enhancements to arbitrary sites by selecting a few elements on the page. One approach (*Nichols & Lau, 2008*) enabled end users to re-author a simplified mobile version of web applications by demonstrating the task and directly choosing page elements. Another system (*Macías & Fabio, 2008*) allowed users to modify the source code of a web page, and then created a generalized modification of similar pages. Inductive programming has become a popular approach, since it requires little experience for users to provide examples and demonstration (*Rinard, 2012*). Nevertheless, even a highly sophisticated synthesis algorithm would be useless if end-users cannot naturally express high-quality examples. In the second study, we examine how users naturally express programming tasks, and explore potential issues and opportunities of symbiotic interaction.

Understanding end-user programmers

There is a wealth of existing work that has investigated end-user programmers from various perspectives. Researchers have been interested in what set of functionalities are simple yet effective for a wide range of tasks. *Wong & Hong (2008)* surveyed 22 popular mashups and identified common functionalities including aggregation, personalization, and real-time monitoring. *Zang & Rosson (2008)* and *Zang & Rosson (2009b)* conducted surveys on potential end-user programmers to examine categories of potential mashup applications. In another study, *Zang & Rosson (2008)* investigated end-user's mental models (e.g., where they search information for mashup), and how they use an existing mashup tool to build a simple mashup application. *Rosson, Ballin & Rode (2005)* also examined differences between professional and informal Web developers. Our first interview study has similar motivations—to investigate what challenges end-users experience and how they would improve them. While the aforementioned work mostly focused on how existing tools and techniques had been perceived and used by end-user programmers, our interview examines unmet needs from challenges that ordinary people experience.

End-user programmers face many challenges that professional programmers do, including understanding requirements, making design decisions, reusing and integrating existing code, managing versions, testing, and debugging (*Ko et al., 2011*). *Cao et al. (2010a)* observed how end-user programmers make design decisions while using Popfly, a mashup builder, and suggested implications for effective end-user programming environments. Version control and debugging are inevitable parts of programming. *Kuttal, Sarma & Rothermel (2011)* and *Kuttal, Sarma & Rothermel (2014)* investigated how end-user programmers control different versions of programs they built and what support they require. *Cao et al. (2010b)* studied how end-users test and debug mashups through a process of successive refinement. A few studies recognized a wide spectrum of end-users, and assigned them different roles. For example, the Reform system (*Toomim et al., 2009*) enables (novice) users to apply enhancements developed by professional end-user programmers. Mash Maker (*Ennals et al., 2007*) gives novice and expert end-users different roles: expert users extract semantic structures from raw Web sites that novice end-users will use to create mashups. Both studies in this paper improve our understanding of end-user software engineering. For example, findings from the interview study report the hidden costs of using third-party extensions. Our Wizard-of-Oz study expands our understanding of non-programmers mental model, especially in collaboration with an intelligent agent. Together, the two studies in this paper are complementary to the aforementioned work, providing guidance for the design of future EUP systems.

STUDY 1: END-USER NEEDS ON THE WEB

To better understand end-user needs on the Web, we conducted a semi-structured interview study. The goal is to better understand the challenges that the participants experience, and enhancement ideas that they envision without technical constraints. The approach is to qualitatively analyze the participant responses to identify themes that should be considered in the development of future WebEUP systems.

Table 1 Occupational background of the participants of study 1.

Graduate students		15
	Engineering	8
	Business	4
	Psychology	2
	Education	1
Professionals		12
	IT specialists	8
	Directors and office managers	4
Non-professionals (e.g., homemaker)		8
Total		35

Participants

A total of 35 participants (14 male, 21 female) were recruited via a university campus mailing list, social network, and word-of-mouth. They were on average 30.8 years old ($SD = 5.1$) and have a wide range of occupations as shown in Table 1. Every participant spends at least one hour per day on the Web. A total of 10 out of 35 participants had used at least one programming language, and five participants had created web pages. However, none of them had the experience of end-user programming on the Web. We did not offer any incentive for participation.

Procedure

A total of 18 interviews were conducted via a video chat program with shared screen (Google Hangout: (https://hangouts.google.com/), while the rest were face-to-face interviews at public areas such as libraries and cafes. The interviewer asked participants, "*Show me a couple Web sites that you recently visited, Tell us challenges that you experienced there. If you could hire a team of designers and developers for free, how would you improve the Web sites?*" We recorded (or videotaped for the face-to-face interviews) the participants visiting two to four sites they recently experienced problems. While demonstrating regular tasks on the sites, participants followed the think-aloud protocol. For the challenges they mentioned, we asked them to imagine a team of third-party developers, and to explain to the "team" an enhancement for the Web site. Each interview covered approximately three ($M = 3.028$) sites, and took approximately 20–40 min. The study was found to be exempt from IRB review.

Data and analysis

A total of 35 participants demonstrated the use of 92 sites ($M = 2.63$) that included online shopping (24 sites), academic research (17), streaming video (11), news (10), work-related sites (7), forums (5), search engines (5), social network services (4), travel (4), finance (2), review sites (1), job market (1), and weather (1). Note that these frequencies do not correlate the frequency of regular visits but the challenges that our participants experienced. While visiting the sites the participants explained 106 challenges. Every interview video was transcribed, and coded. As an exploratory work, we pursued an iterative analysis

approach using a mixture of inductive and deductive coding (*Hruschka et al., 2004*; *Braun & Clarke, 2006*). First, we created a codebook derived from the literature (*Zang & Rosson, 2008*; *Cypher et al., 2010*) and an initial post-interview discussion within the research team. The codebook included types of challenges (lack of relevant information, repetitive operations, poorly-organized information, privacy, security, fake information, bugs), and functionalities required for doing a wide range of WebEUP tasks (mashup, redesign, automation, social knowledge, sharing, monitoring). To assure high quality and reliable coding, two researchers independently coded ten randomly selected ideas. Analyzing the Inter-Rater Reliability (IRR) of that analysis with Krippendorff's alpha ($\alpha = 0.391$; *total disagreements* $= 24$ *out of* 255), we revised the codebook. Then the two researchers coded another ten randomly selected ideas, and achieved a high IRR ($\alpha = 0.919$; *total disagreements* $= 6$ *out of* 248). After resolving every disagreement, the first researcher coded the remaining data. Following the guide of thematic analysis (*Braun & Clarke, 2006*), we collated the different codes into potential themes, and drew initial thematic maps that represent the relationship between codes and themes. We then reviewed and refined the thematic maps, to make sure that data within a theme was internally coherent, and that different themes were distinguished as clearly as possible. The two following subsections summarize the two groups of themes: challenges that participants experience on the Web, and functionalities of WebEUP for addressing those challenges.

Result: challenges

Four groups of common challenges and enhancement ideas follow.

Untruthful information

While trust is a key element of success in online environments (*Corritore, Kracher & Wiedenbeck, 2003*), 17 participants reported four kinds of untruthful information on the Internet.

Deceptive ads were reported by three participants. Two of them reported deceptive advertisements that used confusing or untrue promises to mislead their consumers. For example, P31 gave a poignant example that a local business review site posts unavailable items on the Internet:

> *"If you're looking for a contractor to work on your home, and other home stuff, [local business review site] shows them with ratings. A few weeks ago I started paying them again for other information, but they have something very frustrating. They have a several page list of mortgage brokers searchable from [search engine]. But when you pay the fee for their service, they have only a fraction of the information. I complained to them, but they have some stories why it is not... Anyways, I canceled my membership without getting my one month fee refunded." (P31)*

Another participant tried to avoid using an online marketplace because of deceptive ads in it: "*I know there are rental houses with good value on [online marketplace], but I do not use it often. There are too many liars on [online marketplace]. Instead I post on [Social Network Service] to get help or recommendations from people that I trust." (P21)*

Links to low-quality content were reported by seven participants. During the interview, two participants clicked broken links to error pages. Five participants reported that they had to spend significant time and effort to find high-quality video links in underground streaming video sites: *"At [Underground TV show sites], I have to try every link until I find the first 'working' link. By working, I mean the show must be [in] high-resolution, not opening any popup, and most of all as little ads as possible." (P6)* A straightforward solution is to attach quality markers next to the links. However, it is extremely challenging to define a metric of high-quality links that everybody will agree upon.

Virus and Malware was reported by four participants. They were aware of the risks of installing programs downloaded from the Internet, but estimating risks is often inaccurate. For example, two participants stopped using a streaming video site and a third-party plugin worrying about computer viruses, though in fact, those site and plugins were safe.

> *"I used this streaming link site for a while, but not after a friend of mine told me her computer got infected with malwares from this Web site. I wish I could check how trustworthy the site [is] when using [it]." (P24)*

> *"I have [used popup blocker extension], but am not using [it] now. Those apps have viruses, don't they? I also don't use any extensions." (P17)*

This suggests that end-users may have inaccurate knowledge about the risks of their activities on the Internet. Even though third-party programs provide terms and conditions, and permission requests, users are often 'trained' to give permission to popular apps (*Chia, Yamamoto & Asokan, 2012*) as stated by P27: "If the site is important to me, I just press the 'agree' button without reading."

Opinion spam was reported by four participants. While social ratings and consumer reviews are conventional ways to see feedback on products and information, the reliability of the feedback is often questionable (*Ott et al., 2011*). Four participants reported concerns about opinion spam—inappropriate or fraudulent reviews created by hired people. For example, P31 reflected, *"I saw that some sites have certificates, but they were on their own sites. So, who knows what they're gonna do with that information? [. . .] For example, I had a terrible experience with a company that I hired for a kitchen sealing repair, even though they had an A+ rating on [a local business review site]."* P27 also expressed concerns about fake reviews, *"ratings are somewhat helpful. However, I cannot fully trust them especially when they have 5 star ratings—they might have asked their friends and families to give them high ratings."* Similar to deceptive ads, opinion spam is a gateway to serious financial risks such as Nigerian scams (*Corritore, Kracher & Wiedenbeck, 2003*), but there is no simple way to estimate the risk.

Summary. In order to deal with untruthful information, participants would look for more trustworthy alternatives. For example, P21 used a social network service instead of online marketplaces. If participants could not find an alternative source, they would assess the risks and benefits of using the untruthful information, and decide either to give up the task or to take the risk, as P31 said: *"I don't believe everything on the Internet. But sometimes I have no other choices than to try it with caution."* The remaining issue is that estimating the risk of untruthful information is often quite difficult.

Cognitive distraction

Most participants reported cognitive distractions that make information on the Web hard to understand. We identified four types of cognitive distractions as listed below.

Abrupt design changes was reported by three participants. Websites are occasionally redesigned—from a minor tune-up to a complete overhaul—for good reason. However, it often undermines the prior knowledge of its users, and makes the sites navigation difficult. For example, P22 could not find her favorite menu item because "*the library recently changed its design, making it much harder to find the menu.*" Since she found a button for switching back to the classic design at the end, she didn't take advantage of new features in the updated design. P24 shared a similar story: "*One day Facebook suddenly changed the timeline to show this double column view. That was very annoying.*"

Annoying advertisements was reported by 30 participants. We found that the degree of cognitive distraction varies across different types of ads. For example, ads with dynamic behavior are much more annoying than static banner ads:

> "*There are popup ads that cover the content and follow your scrolling. Although they usually have very small 'X' or 'Close' buttons, I often miss-click the popup to open the Web page. That's pretty annoying.*" *(P17)*

This finding is consistent with prior research that found display ads with excessive animation impair user's accuracy on cognitive tasks (*Goldstein, McAfee & Suri, 2013*). 16 participants were using browser extensions (e.g., Chrome AdBlock http://goo.gl/rA6sdC) to automatically remove ads. However, one participant had stopped using it for security and usability issues:

> "*I have, but am not using [AdBlock] now. Those apps have viruses, don't they? [...] They would be very useful in the beginning, however they also restrict in many ways. For example, the extension sometimes automatically block crucial popup windows. So I ended up manually pressing 'X' buttons.*" *(P17)*

Unintuitive tasks. Six participants reported that several Websites are hard to use. For example, to create a new album in Facebook, users are required to upload pictures first. This task model clearly did not match a participant's mental model: "*I tried to create a new photo album. But I could not find a way to create a new album without uploading a picture. That was a very annoying experience.*" (P18). Another user reported a similar issue of not being able to create a new contact after searching in a mailing list:

> "*I'm adding a new person to the contact database. I should first search the last name in order not to put duplicate entry. If the name does not exist, it simply shows [0 result found]. Obviously I want to add a new entry, but there's no button for that. That bugs me a lot, because I have to get back to the previous page and type the name again.*" *(P16)*

Websites with unintuitive navigational structures would require users to do many repetitive trial-and-errors.

> "*When preparing to visit a touristic place, I look for entrance fee, direction, and other basic information from their official sites. However, some sites have that information deep in*

their menu structure, so I had to spend much time finding them. I wish those information were summarized and shown in one page. Sometimes it's hard to find useful images for campsites or cabins. For example, I want to see the image of bathroom, but people upload pictures of fish they caught." (P33)

Information overload. Five participants reported that excessive and irrelevant information prevents them from understanding the main things that they care about. For example, P22 was disappointed at blog posts full of irrelevant information: *"I was searching for tips to clean my computer. However, most blog posts have very long explanations of why I should keep computers clean without telling how to clean it till the end." (P22)*

A long list without effective filtering also causes information overload as P2 stated:

"I want these conferences filtered by deadline, for example, showing conferences whose deadlines are at least 1-month from now. Also, if possible, the filter can look at descriptions of each venue and choose ones containing at least three relevant keywords."

A simple enhancement to solve this problem is to remove unnecessary, excessive information, which is often very hard to decide. For example, P27 criticized an online shopping site for having a lot of unnecessary and irrelevant information. However, when evaluating usefulness of individual components, she became more vigilant, and stressed that her opinions are personal and depending on her current situation.

"I would remove these promoted products on the side bar. However, if these promotions were relevant to my current interest, I would keep them. [. . .] Shopping cart and Personal coupon box can be useful later. [. . .] I don't need extra information about secured payment, getting products at the shop, or printing receipts." (P27)

To enhance websites with an over-abundance of information, participants envisioned creative scenarios including interactivity and design details. For example, P2 proposed to add a custom filter for a long list. P26 wanted to have the personalized summary at the top of a long document with a pop-up window for important information:

"I do not read every Terms and Condition agreement. It's too long and mostly irrelevant. However, it would be useful if hidden charges or tricky conditions were highlighted. I think critical information such as hidden charges can be shown in a pop-up window. It would be best the most important summary is shown at the top, because I could just click 'yes' without scrolling it down." (P26)

Repetitive operations

Participants reported tedious and repetitive operations on the Web. Based on them, we identified three common reasons for repetitive operations.

Unsupported tasks. Seven participants wanted to automate repetitive tasks. Efficient repeating of some of the tasks is unsupported by the websites. For example, four participants wanted to automate simple interactions such as downloading multiple files or clicking a range of checkboxes with a single click.

> *"[At an academic library], I click the "Save to Binder" button, then select a binder from the drop-down in a new window. Then I click the "save" button then the "done" button, then close the window. It's really annoying to do it over and over. It would be great to create a "save this!" button." (P4)*

Three participants wanted to automate filling the forms of personal / credit card information.

Information from multiple sources. Reported by 20 participants, integrating information from multiple sources is a common practice on the Web (*Zang & Rosson, 2009b*). End-users switch between browser tabs to compare information repeatedly, but it can be time consuming since it requires short-term memory to compare information on tabs that are not simultaneously visible. A total of 17 participants wanted to save their time and effort by integrating information across multiple sources. For example, P33 told, "*I often search for videos on YouTube for baby diapers or other things to wear because those videos are very helpful to understand usage of products.*" Similarly, four participants wanted to integrate course schedule page with extra information available such as student reviews, lecture slides, and reading lists.

Time-sensitive information. Five participants reported that they regularly check time-sensitive information such as price (3), hot deals (1), second-hand products (1), and other notifications (1). Using price trends as an example, three participants envisioned a complex service that automatically archives price information retrieved from multiple sites, visualizes the price data as timeline graph, and sends email/mobile notifications when the price drops:

> *"I can imagine that program or Web site will be able to grab information, especially prices from various malls, and compare it automatically. It will also say 'this is the lowest price for recent three months.' so that I don't have to visit Amazon and Newegg everyday. [...] I want it to send me email alerts—saying 'Hey, based on your recent search history on the Canon G15, we found these new deals and prices. It's the lowest price in the last month.' (P21)*

> *"[She opened http://CamelCamelCamel.com] If I want to buy a bread machine, I search and choose one model. Here the graph shows the price trend of the model. I can make a decision on whether I should buy or wait. Unfortunately, this site only shows products from Amazon.com." (P33)*

Privacy

Privacy did not come up much, but one participant (P24) expressed strong negative opinions about the way that a social networking service handles her data:

> *"[At a social network service] a friend of mine told me that if I 'like' her photos or put comment on them, others will be able to see it even if the photos are private. [...] Here's another example that I don't like about [the SNS]. One day I uploaded a family photo, and my family-in-law shared those photos. That's totally fine. However, the problem began when friends of my family-in-law started liking and commenting on my family photos. I*

Table 2 Potential functionality of Web enhancements for addressing categories of challenges and enhancements identified from the interview study. (⊙: the functionality is required or related to at least half of challenges and enhancements in the category, ○: required by less than half, ×: No enhancement or challenge in the category required the functionality).

	Modify	Compute	Interactivity	Gather	Automate	Store	Notify
Untruthful information							
Deceptive ads	⊙	⊙	○	○	×	×	×
Links to low-quality content	⊙	⊙	⊙	○	⊙	⊙	×
Virus and Malware	⊙	⊙	⊙	⊙	⊙	⊙	×
Opinion spam	⊙	○	○	⊙	×	⊙	×
Cognitive distraction							
Abrupt design changes	⊙	×	⊙	×	⊙	⊙	×
Annoying advertisements	⊙	⊙	⊙	×	⊙	○	×
Unintuitive tasks	⊙	○	⊙	⊙	⊙	×	×
Information overload	⊙	⊙	⊙	×	○	⊙	×
Repetitive operations							
Unsupported Tasks	⊙	×	⊙	○	⊙	⊙	×
Information from multiple sources	⊙	⊙	⊙	⊙	○	○	○
Time-sensitive information	⊙	○	⊙	⊙	⊙	⊙	⊙
Privacy	○	×	×	×	○	○	○

received a lot of notifications of those activities by people I do not know at all. I felt a little scared." (P24)

As another example of privacy issues, P24 believed that her browser tracks her activity history, and shared it with online advertisement companies without her permission, because banner ads on other Web pages show ads related to her previous activity.

Potential functionality of web enhancements

This section presents functionalities that we believe future WebEUP systems should consider providing to address the challenges reported in the previous section. We identified seven functionalities: *Modify*, *Compute*, *Interactivity*, *Gather*, *Automate*, *Store* and *Notify*. Table 2 illustrates how closely each functionality is related with the challenges. Among them, *Interactivity*, *Store*, and *Notify* are not generally supported by existing WebEUP systems.

Modify. Modification of existing web pages is the most commonly required functionality for 66 out of 109 enhancements. Examples include attaching new DOM elements to the original pages (31 enhancements), removing or temporarily hiding unnecessary elements (15 enhancements), and highlighting information of interest by changing font size, color, or position (five enhancements). Modification often involves adding new interactive behavior of Web sites (eight enhancements). Existing WebEUP tools support a wide range of modification such as removing unwanted DOM elements (*Bolin et al., 2005*), and attaching new DOM elements or interactive behavior to existing elements (*Toomim et al., 2009*).

Compute. A total of 29 enhancements require a variety of computing data. For examples, 13 enhancement ideas against untruthful information and distracting content on the Web involve filtering elements by certain criteria. Nine ideas of integrating

information from multiple sources require 'compute' functionality to extracting specific information from available documents. Seven enhancement ideas require arithmetic operations. While computation is a fundamental part of programming languages, existing EUP systems support it in varying degrees. For example, scripting languages (https://addons.mozilla.org/en-US/firefox/addon/greasemonkey/) offer an extensive set of language constructs such as general-purpose languages (e.g., JavaScript). Data integration systems (*Tuchinda, Szekely & Knoblock, 2007*; *Wong & Hong, 2007*) focus on handling large amount of semi-structured text input, but provide less support on numerical operations. Systems for automated browsing (*Miller et al., 2008*; *Leshed et al., 2008*) provide few language constructs for computation.

Interactivity. A total of 29 enhancements across all categories (except the *privacy* issue) require interactive components to accommodate the dynamic and uncertain nature of the Web and end-user needs. For example, 13 enhancements include triggering buttons, because users wanted to make use of them *in-situ*. Eight enhancements show previews of changes it will make on the original sites so that users can choose among them. Enhancements often require users to configure options such as search keywords, filtering criteria, specific DOM elements based on their information needs (eight enhancements). WebEUP tools often employ predefined interactive components such as buttons and preview widgets (*Toomim et al., 2009*). However, none of them enable users to create their own interactivity.

Gather. A total of 18 enhancements involve gathering information from either the current domain (9 enhancements) or external sources (five enhancements). For example, to deal with various kinds of untruthful information, enhancements should be able to gather information from trustworthy sources. Gathering is an essential part of integrating information from multiple sources, as P5 stated, "*At various cosmetic malls, I wish the main listing page showed detailed direction on how to use the products.*" In contrast, participants wanted to gather information from external sources that current sites are missing. Information gathering is supported by mashup tools (*Wong & Hong, 2007*; *Ennals et al., 2007*; *Tuchinda, Szekely & Knoblock, 2008*).

Automate. A total of 15 enhancements automate repetitive tasks that include filling in input forms (four enhancements), downloading multiple images and files (four), page navigation (three), clicking a series of buttons, checkboxes, and links (three), and keyword search (one). Existing WebEUP tools such as CoScripter (*Leshed et al., 2008*) and Inky (*Miller et al., 2008*) support automating repetitive tasks.

Store. A total of 14 enhancements store three types of data while being used. The first type relates to user's activities such as filling input forms, page navigation, and job applications found in five enhancements. The second type is temporal information periodically gathered from designated sources such as online shopping malls, or ticketing sites found in five enhancements. The last is bookmarks of online resources such as news articles, blog posts, or streaming videos found in four enhancements. Existing WebEUP systems such as CoScripter (*Bogart et al., 2008*) often provide public repositories for scripts, but none of them allow end-users to create custom storage of usage data.

Notify. Eight enhancements for integrating information from multiple sources ('Information from multiple sources') and gathering time-sensitive information ('Time-sensitive information') involve sending notifications to users via emails (seven) or SMS messages (one), periodically or when user-specified events occur. To our knowledge, no existing WebEUP tool supports notification.

Discussion

In this section, we discuss two design implications for future WebEUP systems and designing Web sites in general.

Social platform beyond technical support

Traditional WebEUP systems focus on lowering the technical barrier of Web programming. For example, mashup tools enable users to integrate information from multiple pages with just a few clicks. Automation tools allow users to create macro scripts through demonstration. Despite the advantage of those technical aids, we noted a few enhancement ideas require domain knowledge of multiple users who have the same information needs. For instance, when end-users want to integrate additional information with original pages, the key question is where the additional information can be found. When users want to focus on an important part of a long text, the key is which part of the text previous visitors found useful (similar to Amazon Kindle's "Popular Highlights" feature.) An example of how a social platform could address the *untruthful information* issue follows. An end-user programmer creates and deploys an enhancement that attaches an interactive component (e.g., button for rating individual hyperlinks) to the original page. Users who have installed the enhancement would use the new component to provide their knowledge (e.g., quality of the linked resources), which will be saved in the enhancement's database. As more data is collected, the enhancement will become more powerful.

To enable end-users to build social platforms in the aforementioned scenario, future WebEUP systems need two functionalities. First, end-user programmers should be able to create and attach interactive components that collect knowledge and feedback from users. Second, end-user programmers should be able to set up centralized servers that communicate with individual enhancements running in each user's browser, and store collected information. To our knowledge, no prior WebEUP system has fully supported these functionalities for social platforms. However, there are certainly custom solutions of this type that are commonly used such as, for example, Turkopticon https://turkopticon.ucsd.edu/ that helps web workers using Amazon Mechanical Turk rate job creators.

Alleviate the risk of using enhancements

According to the *attention investment* framework (*Blackwell & Burnett, 2002*), end-users would decide whether to use an enhancement or not as a function of perceived benefit versus cost. Even though our participants assumed no development costs, we could identify the following concerns about risks of using enhancements.

Uncertain needs. As mentioned earlier, the dynamic and uncertain nature of the Web and end-user needs requires interactive behavior of enhancements. For example, P27

found advertisements on an online shopping site to be annoying, but did not remove the advertisements because of their potential usefulness in the future. WebEUP systems should be able to support interactivity so that users can change configurations or make decisions whenever their needs change. Otherwise end-users will be forced to stop using it, as P26 and P17 did with non-interactive pop-up blockers.

Breaking the original design. Enhancement developers should try to minimize unnecessary change of the original site. Two participants expressed concerns about breaking the original site's design and functionality: "*I think the best part of Craigslist is its simplicity. I might have seen the filters, but did not bother setting them every time I visit this site.*" (P20)

Privacy and security. End-users have significant privacy and security concerns about installing extra software, especially those developed by third-party programmers. Ironically, we observed end-users rarely read legal documents, and are trained to give permissions to popular apps. Future work should confront these practical concerns and design how to communicate potential risks and treatments.

Implications for web site designers

The seven categories of enhancements can be useful to web site designers as they think about what a wide range of users might want. There is another potential of more directly benefiting from end-user modifications to web sites. Actual enhancements made by end-users could provide valuable feedback for designers of the sites if those desires were expressed via use of a WebEUP tool. For example, designers could learn what kind of information users consider to be untruthful by learning about user feedback on specific information. Repetitive operations could be observed by seeing what modifications users make, etc. Nevertheless those feedbacks cannot replace WebEUP, as designers and users often have conflicting interests. For instance, designers may not agree to remove advertisements that end-users find annoying since they provide revenue. Some ideas may be useful for specific user groups but not for everyone, and so are not worth pursuing. Ideally, designers should consider providing hooks or APIs that enable end-users to build robust, high-quality enhancements.

Summary

The interview study explored the space of challenges that end-users regularly experience on the Web, and the functionalities of enhancements that they envisioned. The seven categories of enhancements can provide guidance to website designers in the first place to be aware of the unique needs of many users. Also, WebEUP tools should allow end-user programmers to customize interactivity and design of enhancements they create.

The interview study has a few limitations. First, 35 participants is surely not large and diverse enough to fully represent the needs that all end-users might have. However, we believe that even this modest number has surfaced a significant number of important areas that need support. Second, most participants shared their experience relating to non-professional tasks such as academic searches, shopping, and watching videos, but had little to discuss about professional activities. Thus, the findings should be complemented by future research that focuses on professional activities. Our exploratory study focused

Table 3 Occupational background of the participants of study 2.

Graduate students		6
	Engineering	3
	Business	2
	Education	1
Professionals		3
	IT specialists	2
	Directors and office managers	1
Non-professionals (e.g., homemaker)		3
Total		13

on end-user needs without considering development costs. Although our decision allowed end-users to express ideas more freely, future work needs to ask participants to both estimated cost and benefits in relation.

STUDY 2: NON-PROGRAMMERS MENTAL MODEL OF COMPUTATIONAL TASKS

Programming is difficult to learn since its fundamental structure (e.g., looping, if-then conditional, and variable referencing) is not familiar or natural for non-programmers (*Pane, Myers & Ratanamahatana, 2001*). Understanding non-programmer's mindset is an important step to develop an easy-to-learn programming environment. The second study builds on the enhancement ideas from the first study by examining how non-programmers naturally describe common computational tasks. Since the study is designed to be open-ended and formative, participants were not provided with a specific tool or language constructs, but expressed intent via verbal statements, examples, and gestures. They also refined their expression through conversational dialogues with the interviewer using the Wizard-of-Oz technique. As result, the findings provide broad and general implications of how non-programmers express computational intent, and suggest open-ended research questions for future EUP systems.

Participants

The study was conducted with 13 participants, including five males and eight females, average 33.3 years old (STD = 5.86) with varying occupations as summarized in Table 3. All of the participants were experienced computer users, but they all said that they had not programmed before. The participants were recruited by the university mailing list that we used in the first study. They received no compensation for participation in this study.

Method

The study aims to characterize how non-programmers naturally describe complex tasks without being biased by specific language constructs or interactive components.[1] We employed the Wizard-of-Oz technique (*Zimmerman et al., 2009*) where the interviewer acted as a hypothetical computer agent that could understand the non-programmer's verbal statements, behavioral signals (e.g., page navigation, mouse click), gestures, and

[1] The study is exempt from IRB review.

drawing on scratch paper, and help them through conversational dialogue. The computer agent (called "computer" from here on) followed the rules listed below.

(1) The computer can understand all the literal meaning of participants' instruction, gestures, and drawings. However, the computer cannot automatically infer any semantic meaning of the task or the material. For example, a rental posting "*4 Bedrooms 3 Lvl Townhome $1,650/4br*" is just a line of text to the computer.

(2) The computer can perceive a pattern from participant's repeated examples and demonstration. For example, if a participant counted numbers within a range 1–3 in a table, the computer asks the participants "*Are you counting numbers that are within a specific range?*"

(3) The computer can execute the participant's instruction only if it is clearly specified without ambiguity. Otherwise the computer asks for additional information to resolve it through conversational dialogue like below:

Programmer: Delete houses with fewer than three bedrooms.
Computer: Please tell me more about 'houses with fewer than three bedrooms'. Which part of the page is relevant?
When the programmer demonstrates a set of examples, the computer will suggest a generalizing statement like below:
Programmer: Delete this one because it contains 3br.
Computer: Do you want me to delete every line that has 3br?

A sheet of paper containing basic instruction was provided, and the participants could draw or write anything on the paper as shown in Figs. 1 and 2.

We chose three tasks that end-user programmers commonly do.

Task 1. Drawing histogram. Given a sheet of paper containing a blank histogram and 10 random numbers between 0 and 12 (see Fig. 1), the participants were asked to explain the computer how to draw a histogram of the numbers. The blank histogram has four bins ($0 \sim 3$, $3 \sim 6$, $6 \sim 9$, and $9 \sim 12$). The purpose of this task was to observe how non-programmers perform: (1) common data-processing operations (e.g., iteration, filtering, and counting), and (2) visualize numeric data by examples and demonstration.

Task 2. Custom filter. We prepared 10 rental postings in Table 4 copied from an online marketplace http://Craigslist.com. The participants were asked to create a program that removes houses having fewer than three bedrooms. The program consists of three components: (1) extracting text that represents the number of bedrooms in each post (e.g., "*3br(s),*" "*3bedroom(s),*" "*3 BEDROOMS,*" "*3/2*"), (2) a conditional logic for filtering posts with less than three bedrooms, and (3) removing/hiding the filtered houses. The purpose of the task is to observe how non-programmers decompose a big task into sub-tasks, specify extraction queries, and refer temporary variables such as sub-strings and selected postings.

Task 3. Mash up. At Amazon.com, each product has different options (e.g., available colors and sizes) that are shown in the product detail page. The participants are asked to create a program that extracts the available colors from detail pages, and attaches to the product listing. The purpose of the task is to understand how non-programmers would describe copy operations across multiple pages, and event handling.

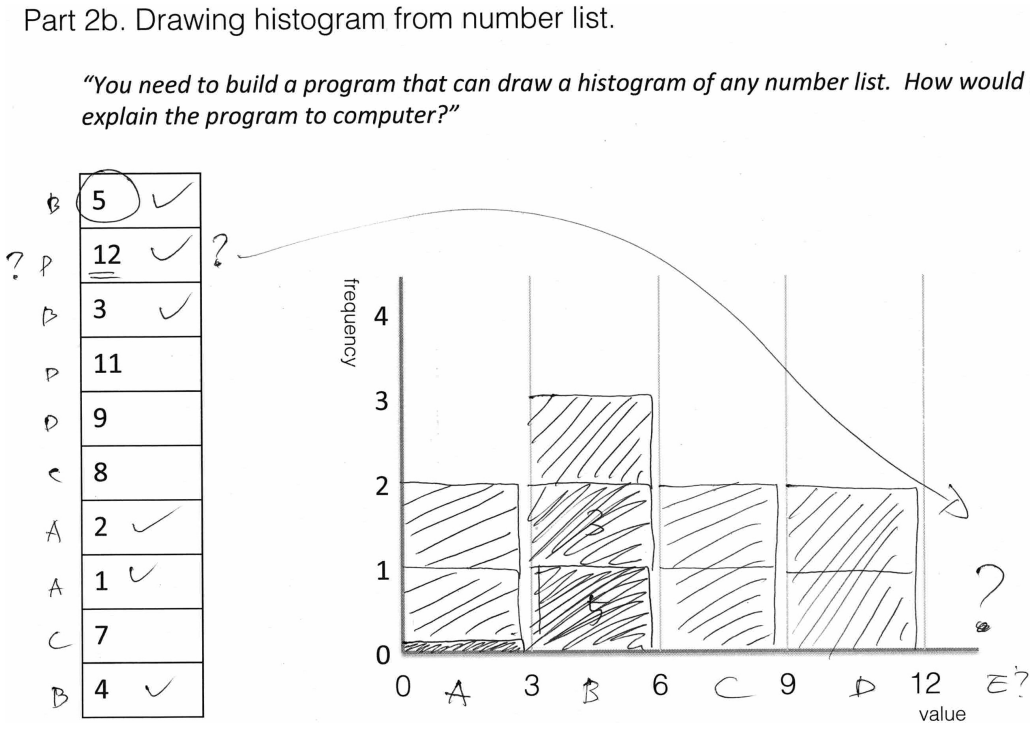

Figure 1 An example drawing of Task 1. Participants were asked to explain how to draw a histogram of the numbers in the table. In this example, the participant gave histogram bins different codes (A–D), and marked each number with the codes. Since the participant could not put 12 into any bin, he marked the number with question mark and a line that points a missing bin.

Table 4 Task 2 material. In Task 2, the participants were asked to create a filter than removes houses with less than three bedrooms among housing rental posts scraped from http://Craigslist.com.

"You want to create a filter that removes houses having less than 3 bedrooms. How would you explain it to the computer?"
• Brand New Townhome! \$2,200/3br—1948ft^2—(Clarksburg)
• Lanham 2/1 new deck \$1,050/1818ft^2—(Lanham)
• 4 Bedrooms 3 Lvl Townhome \$1,650/4br—(MD)
• 823 Comer Square Bel Air, MD 21014 \$ 1,675/studio …(6 more)…

Procedure

Each session began with a brief interview about the participant's programming experience and occupational background. The interviewer introduced the Wizard-of-Oz method, and gave an exercise task—ordering the interviewer (acting the hypothetical computer agent) to move a cup to another corner of the table. After participants said they fully understood the concept of the hypothetical computer agent, we started the actual study by introducing the three scenarios in a randomized order. For each scenario, participants were asked to explain the task to the "computer." Participants were allowed to finish or to give up a task at any point.

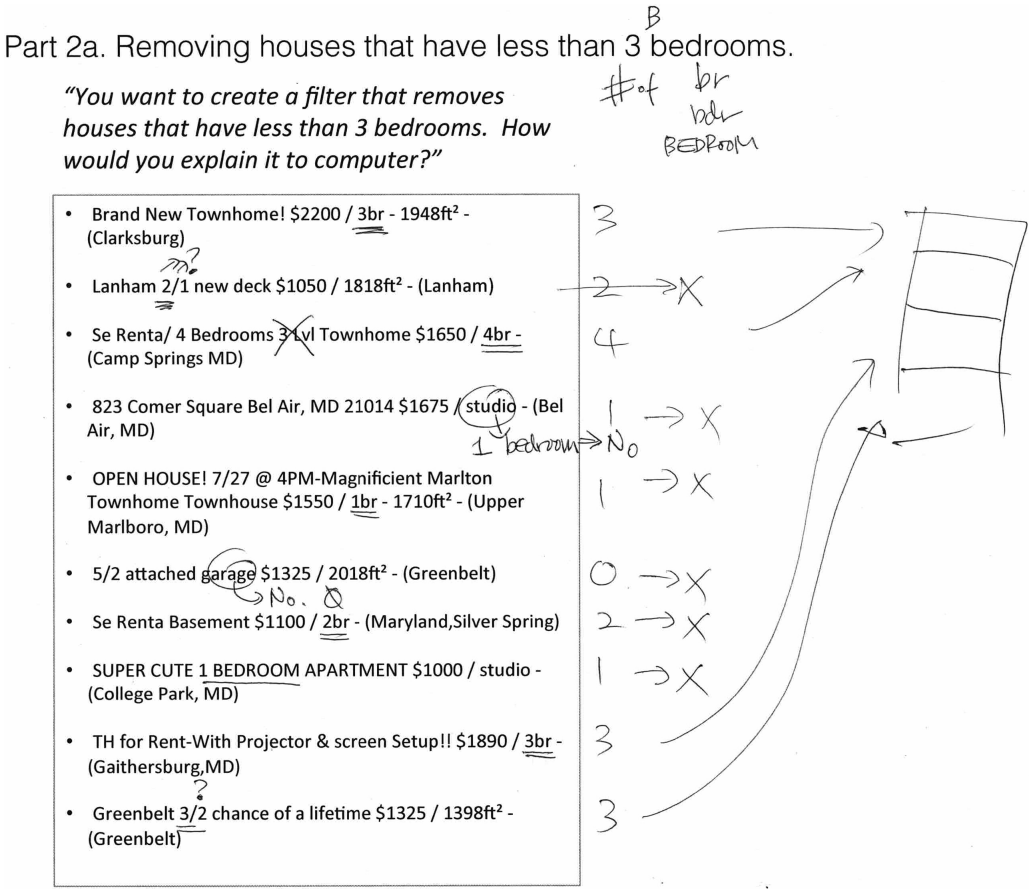

Figure 2 **An example drawing of Task 2.** In this example of Task 2, the participant used scribbles along with verbal statements. For example, the participant wrote variations of keywords that indicate "bedroom" used in the list. He/she also circled and underlined the number of rooms in each title to demonstrate the text extraction logic, crossed out titles that did not meet the criteria, and drew arrows from houses to empty slots in the list.

Data and analysis

The entire session was video recorded, and transcribed for qualitative analysis. The transcript of each task consists of a sequence of conversational dialogue between the participant and the interviewer, finger and mouse pointing gestures, scribbles on the paper (Figs. 2 and 3; only for Task1 and T2), and page scroll and mouse events in the browser (only for Task 3). To analyze the transcript, the first author created the initial codebook derived from the literature (*Pane, Myers & Ratanamahatana, 2001*) and an initial post-interview discussion within the research team. The codebook included how the participants described and what challenges they experienced. While repeating the coding process, a few categories emerged: *programming styles*, *imperative commands*, *ambiguities*, and *multi-modal intent*.

Findings

In this section we characterize how non-programmers describe computational tasks. Participants were allowed to stop at any moment, but all of them could eventually complete tasks with the computer's help. Each task took an average of 415.3 s ($SD = 217.4$). We did

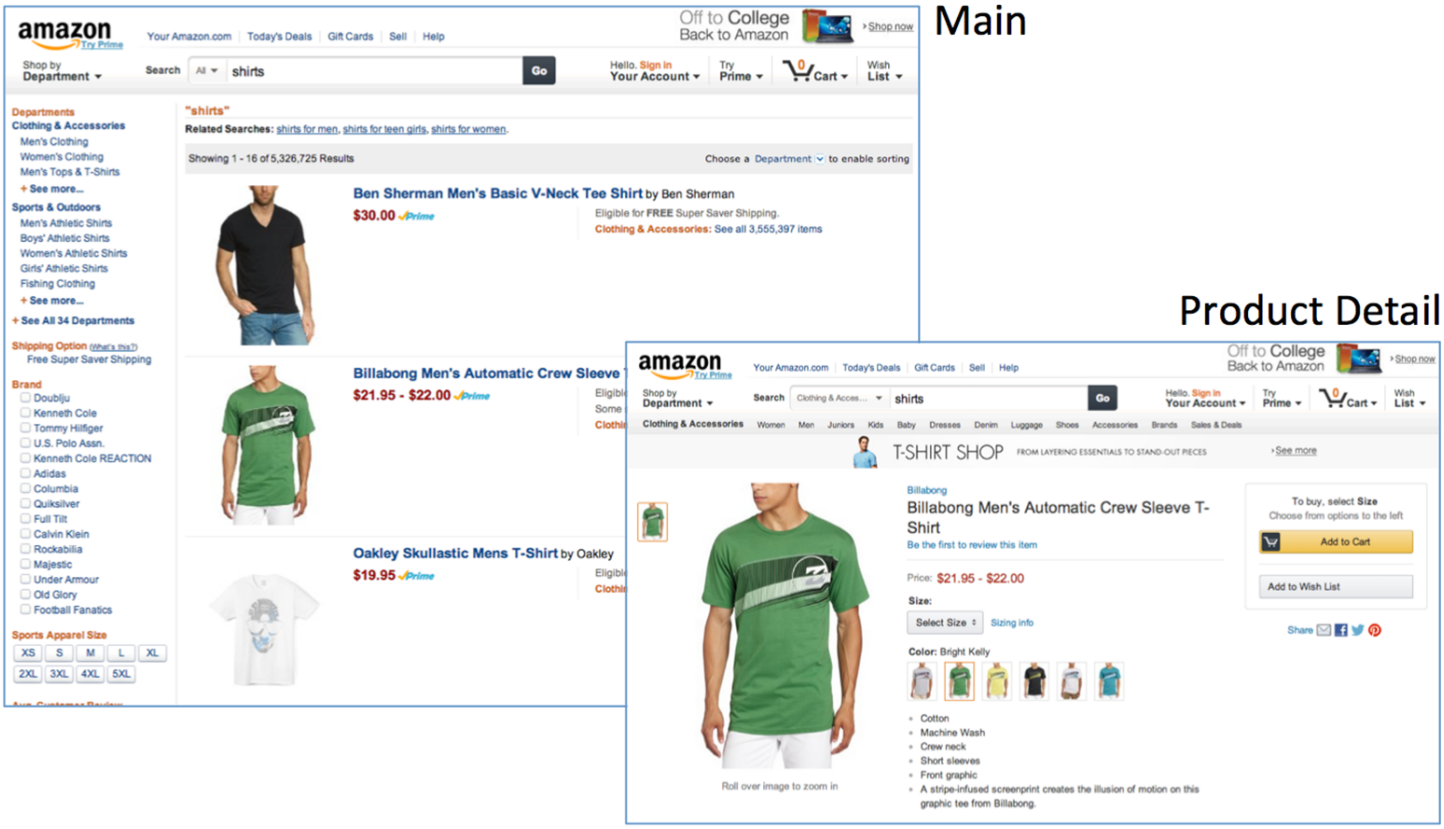

Figure 3 **Task 3 material.** In Task 3, participants were asked to describe a simple Mashup program that shows available colors of each individual product in the Main page extracted from the Product Detail page.

not observe any fatigue effect. Since participants had very limited understanding of the computer at the beginning, most of their initial explanations were not very informative. Thus the computer asked for further information as the examples below.

(Task 1. Histogram)
P12: Wouldn't computers draw graph when numbers are assigned? I'm asking because I have no idea.
P11: Find the numbers, and draw them at the first bin. Computer: How can I draw them?
P11: What should I tell? Color?
(Task 2. Custom filter)
P5: First, I scan the list with my eyes and exclude them. They clearly stand out Computer: How do they stand out?
P8: I'd order, "Exclude houses with one or two bedrooms. "Computer: How can I know the number of bedrooms?
(Task 3. Mash Up)
P11: I'd ask computer to show available colors of this Columbia shirt. Computer: Where can I get available colors?

Natural language tends to be underspecified and ambiguous (*Pane, Myers & Ratanamahatana, 2001*). We frequently observed that our participants skipped mentioning essential information. For example, most participants did not specify how to iterate multiple elements in list. They instead demonstrated handling the first item, and expected the computer to automatically repeat the same process for the rest of the items. They did not refer to objects by names as programmers use variables. However, they referred to previously mentioned objects by their actual values (underlined in the following example), as P20 said, "*In this next column, we need items going 6, 7, and 8. So please find those 6, 7, 8, and draw bar in this column.*" They also used pronouns (e.g., "*Remove them*"), data type (e.g., "*Attach colors"*), and gestures (e.g., "*Paste them here.*"). While loops and variable referencing are core concepts of programming languages, our findings suggest that non-programmers would find them unnecessary or even unnatural. We will discuss the issue further with design implications for future EUP systems in the discussion section.

Through conversational dialogues, participants figured out what information the computer requires and how to explain. We found a few characteristics of how non-programmers explain computational tasks as listed below.

Explaining with rules and examples was used by 9 of 13 participants. When participants explained rules first, the following examples usually provided information that the rules were potentially missing. For example, while drawing a histogram for Task 1, P4 stated a rule, "*Determine which bin each number is in,*" followed by an example, "*If the number is one (pointing the first item in the table), then count up this bin (pointing the first bin in the histogram).*" Participants also provided examples first, and then explained the rules. P10 doing Task 1 gave all the numbers (0, 1, and 2) for the first bin, "*For here (pointing the first column) we need 0, 1, and 2*", and then explained the range of those numbers, "*Find numbers including zero, smaller than two.*" Traditional programming languages rarely allow example-based programming. Although EUP systems often support Programming-by-Example (PBE) techniques, they do not allow this pattern—combining rules and examples to describe individual functional elements.

Elaborating general statement through iteration was observed for every participant. Initial explanations of tasks were usually top-level actions (e.g., draw bars, remove houses, attach pictures) that contained a variety of ambiguities; but participants then iteratively elaborated the statements by adding more details. For example, P1 doing T3 described the top-level action, "*Attach pictures here.*" Then he elaborated where the pictures were taken from, "*Attach pictures from the pages.*" He kept on elaborating what the pages are and how to extract pictures from the pages. For T 1, as another example, P14 told the computer, "*Draw a graph.*" She then rephrased the statement with more details, "Draw a graph to number 2." This pattern is far from traditional programming languages that support users to create statements in the order of their execution.

Multi-modal expressions including gestures and scribbles were frequently used by all participants. While verbal statements were still the central part of explanation, they used gestures along with pronouns (e.g., "*Count these*", "*Put them here*"), and

scribbles to supplement verbal statements like an example in Fig. 2. While multi-modal expressions seem to be natural and effective for non-programmers, traditional programming environments rarely support them.

Rationales are not direct instructions for the computer. However, we consistently observed participants explaining rationales. For example, P6 doing T3 explained why she chose to attach small color chips rather than larger images, "*While we can show images, which would be quite complex, I'd want you to do use color boxes.*" P13 also explained rationale of her scribbles on the sheet of T1, "*We can also secretly write number here (center of each cell) to remember, so track for afterward so we didn't make any mistake.*"

Discussion

This user study provides characteristics of non-programmers explaining how they would solve computational tasks. Given that traditional programming environments do not fully support the way these participants conceptualized their solutions, we discuss the implications for the design of multi-modal and mixed-initiative approaches for making end-user programming more natural and easy-to-use for these users. Our recommendations are to:

Allow end-users to express ideas with combinations of rules, examples, gesture, and rationales. A common pattern of multi-modal intent expression is to generate rules (i.e., program modules) from examples or demonstrations, and to allow users to review and modify those rules (*Kandel et al., 2011*; *Le, Gulwani & Su, 2013*; *Yessenov et al., 2013*). In such systems, different modalities have separated roles. Our user study, in contrast, suggests that rules, example, and rationales can be highly effective when used in combination. Future EUP systems should give end-users more flexibility to express their intent via multi-modality.

Support iterative refinement of programs. It is well known that end-users may not be able to provide complete information of the programs they want (*Lau, 2008*). We observed that users would start with quick and brief description of task outlines, goals, or solutions that handle only a subset of the potential scenarios, and then iteratively refine it by adding more rules and examples. Many PBE systems (*Gulwani, 2011*) allow users to provide additional examples for disambiguation, and even suggest critical examples (*Mayer et al., 2015*).

Support mixed-initiative interaction to disambiguate user intent. To guide non-programmers to explain essential information such as loops and variable referencing, our study employed conversational dialogue (as explained in 'Method') between participants and the computer. For example, when participants gave incomplete statements (e.g., demonstration for the first item), the computer asked them for additional information ("*What would you like to do for the rest items?*") or confirmation (e.g., "*Do you want to do the same for the rest items?*"). Likewise, future EUP tools should incorporate mixed-initiative interaction to help end-users express unambiguous statements; although it is an open-ended research question how the computer and end-users have mutual understanding.

We made several simplifying assumptions that limit the scope of our findings. First, the computer followed three informal rules, which may not be specific enough to design

working a system. First, the computer did not follow a formal definition for its behavior and abilities. A formal set of rules would make the Wizard-of-Oz study stronger. Second, participants could not see or test programs they were building, which is uncommon for any programming environment. The next step is to study similar questions with a fully interactive system that provides generated programs and test results. Third, the three tasks do not represent a full spectrum of computational tasks. However, as with the first study, we believe that even this narrow analysis provides useful insights that could guide the design of a new WebEUP system. Finally, future work should include a functioning system that presents and tests solutions that address the challenges and the implications of this study.

CONCLUSION

This paper reports results from two user studies that help to better understand the needs and capabilities of Web end-users. The first study, a semi-structured interview study, explores challenges that end-users daily experience, and identifies seven categories of enhancements that we believe would be helpful to be included in future EUP tools. The second, a Wizard-of-Oz study, demonstrates how non-programmers explain common computational tasks and provides design ideas for more natural programming environments. Finally future work should build an interactive research prototype based on the findings, and test with real end-users.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

The authors received no funding for this work.

Competing Interests

Benjamin Bederson is an Academic Editor for PeerJ.

Author Contributions

- Tak Yeon Lee conceived and designed the experiments, performed the experiments, analyzed the data, contributed reagents/materials/analysis tools, wrote the paper, prepared figures and/or tables, performed the computation work.
- Benjamin B. Bederson reviewed drafts of the paper.

Ethics

The following information was supplied relating to ethical approvals (i.e., approving body and any reference numbers):

IRBNet @ University of Maryland College Park (UMCP), Project ID and Title: [443008-1] Survey About Web Inefficiencies. Project was declared exempt from IRB review.

Data Availability

The following information was supplied regarding data availability:

The raw data has been supplied Supplemental Information 1.

Supplemental Information

Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj-cs.91#supplemental-information.

REFERENCES

Biermann AW, Ballard BW, Sigmon AH. 1983. An experimental study of natural language programming. *International Journal of Man-Machine Studies* **18**:71–87 DOI 10.1016/S0020-7373(83)80005-4.

Blackwell A, Burnett M. 2002. Applying attention investment to end-user programming. In: *Proceedings of the IEEE 2002 symposia on human centric computing languages and environments, 2002*. Piscataway: IEEE, 28–30 DOI 10.1109/HCC.2002.1046337.

Bogart C, Burnett M, Cypher A, Scaffidi C. 2008. End-user programming in the wild: a field study of Co Scripter scripts. In: *IEEE symposium on visual languages and human-centric computing, 2008, VL/HCC 2008*. Piscataway: IEEE, 39–46 DOI 10.1109/VLHCC.2008.4639056.

Bolin M, Webber M, Rha P, Wilson T, Miller RC. 2005. Automation and customization of rendered web pages. In: *UIST'05*. New York: ACM, 163–172 DOI 10.1145/1095034.1095062.

Braun V, Clarke V. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* **3**:77–101 DOI 10.1191/1478088706qp063oa.

Cameron JM, Churchill EF. 2009. Conversations in developer communities: a preliminary analysis of the Yahoo! Pipes community. In: *Proceedings of the fourth international conference on communities and technologies*. New York: ACM, 195–204.

Cao J, Rector K, Park TH, Fleming SD, Burnett M, Wiedenbeck S. 2010b. A debugging perspective on end-user mashup programming. In: *2010 IEEE symposium on visual languages and human-centric computing (VL/HCC)*. Piscataway: IEEE, 149–156 DOI 10.1109/VLHCC.2010.29.

Cao J, Riche Y, Wiedenbeck S, Burnett M, Grigoreanu V. 2010a. End-user mashup programming: through the design lens. In: *CHI '10*. New York: ACM, 1009–1018 DOI 10.1145/1753326.1753477.

Cheema S, Gulwani S, LaViola J. 2012. QuickDraw: improving drawing experience for geometric diagrams. In: *CHI '12*. New York: ACM, 1037–1064 DOI 10.1145/2207676.2208550.

Chia PH, Yamamoto Y, Asokan N. 2012. Is this app safe? A large scale study on application permissions and risk signals. In: *Proceedings of the 21st international conference on world wide web. WWW '12*. New York: ACM, 311–320 DOI 10.1145/2187836.2187879.

Corritore CL, Kracher B, Wiedenbeck S. 2003. On-line trust: concepts, evolving themes, a model. *International Journal of Human-Computer Studies* **58**:737–758 DOI 10.1016/S1071-5819(03)00041-7.

Cypher A, Dontcheva M, Lau T, Nichols J. 2010. *No code required: giving users tools to transform the web*. San Francisco: Morgan Kaufmann Publishers Inc.

Cypher A, Halbert DC, Kurlander D, Lieberman H, Maulsby D, Myers BA, Turransky A (eds.) 1993. *Watch what I do: programming by demonstration*. Cambridge: MIT Press.

Ennals R, Brewer E, Garofalakis M, Shadle M, Gandhi P. 2007. Intel Mash Maker: join the web. *SIGMOD Record* **36**:27–33 DOI 10.1145/1361348.1361355.

Fischer G, Giaccardi E. 2006. Meta-design: a framework for the future of end-user development. In: Lieberman H, Paternó F, Wulf V, eds. *End user development*. Netherlands: Springer, 427–457.

Goldstein DG, McAfee RP, Suri S. 2013. The cost of annoying ads. In: *Proceedings of the 22nd international conference on world wide web. WWW '13*. Republic and Canton of Geneva: International World Wide Web Conferences Steering Committee, 459–470.

Gulwani S. 2010. Dimensions in program synthesis. In: *PPDP'10*. New York: ACM, 13–24 DOI 10.1145/1836089.1836091.

Gulwani S. 2011. Automating string processing in spreadsheets using input–output examples. *SIGPLAN Notices* **46**:317–330 DOI 10.1145/1925844.1926423.

Hruschka DJ, Schwartz D, St John DC, Picone-Decaro E, Jenkins RA, Carey JW. 2004. Reliability in coding open-ended data: lessons learned from HIV behavioral research. *Field Methods* **16**:307–331 DOI 10.1177/1525822X04266540.

Kandel S, Paepcke A, Hellerstein J, Heer J. 2011. Wrangler: interactive visual specification of data transformation scripts. In: *CHI '11*. New York: ACM, 3363–3372 DOI 10.1145/1978942.1979444.

Ko AJ, Abraham R, Beckwith L, Blackwell A, Burnett M, Erwig M, Scaffidi C, Lawrance J, Lieberman H, Myers B, Rosson MB, Rothermel G, Shaw M, Wiedenbeck S. 2011. The state of the art in end-user software engineering. *ACM Computing Surveys* **43**:21–1–21:44 DOI 10.1145/1922649.1922658.

Kuttal SK, Sarma A, Rothermel G. 2011. History repeats itself more easily when you log it: Versioning for mashups. In: *2011 IEEE symposium on visual languages and human-centric computing (VL/HCC)*. Piscataway: IEEE, 69–72 DOI 10.1109/VLHCC.2011.6070381.

Kuttal SK, Sarma A, Rothermel G. 2014. On the benefits of providing versioning support for end users: an empirical study. *ACM Transactions on Computer-Human Interaction* **21**:9:1–9:43 DOI 10.1145/2560016.

Lau T. 2008. Why PBD systems fail: lessons learned for usable AI. In: *CHI 2008, April 5–April 10, 2008, Florence, Italy*. New York: ACM,.

Le V, Gulwani S, Su Z. 2013. SmartSynth: synthesizing smartphone automation scripts from natural language. In: *MobiSys '13*. New York: ACM, 193–206 DOI 10.1145/2462456.2464443.

Leshed G, Haber EM, Matthews T, Lau T. 2008. CoScripter: automating & sharing how-to knowledge in the enterprise. In: *CHI '08*. New York: ACM, 1719–1728 DOI 10.1145/1357054.1357323.

Lieberman H. 2001. *Your wish is my command: programming by example*. San Francisco: Morgan Kaufmann.

Macías JA, Fabio P. 2008. Customization of Web applications through an intelligent environment exploiting logical interface descriptions. *Interacting with Computers* **20**:29–47 DOI 10.1016/j.intcom.2007.07.007.

Mayer M, Soares G, Grechkin M, Le V, Marron M, Polozov A, Singh R, Zorn B, Gulwani S. 2015. User interaction models for disambiguation in programming by example. In: *28th ACM user interface software and technology symposium*. New York: ACM.

Miller LA. 1974. Programming by non-programmers. *International Journal of Man-Machine Studies* **6**:237–260 DOI 10.1016/S0020-7373(74)80004-0.

Miller LA. 1981. Natural language programming: styles, strategies, and contrasts. *IBM Systems Journal* **20**:184–215 DOI 10.1147/sj.202.0184.

Miller RC, Chou VH, Bernstein M, Little G, Van Kleek M, Karger D, Schraefel M. 2008. Inky: a sloppy command line for the web with rich visual feedback. In: *Proceedings of the 21st annual ACM symposium on User interface software and technology*. New York: ACM, 131–140.

Nichols J, Lau T. 2008. Mobilization by demonstration: using traces to re-author existing web sites. In: *IUI '08*. New York: ACM, 149–158 DOI 10.1145/1378773.1378793.

Ott M, Cardie C, Hancock J. 2012. Estimating the prevalence of deception in online review communities. In: *WWW '12*. New York: ACM, 201–210 DOI 10.1145/2187836.2187864.

Ott M, Choi Y, Cardie C, Hancock JT. 2011. Finding deceptive opinion spam by any stretch of the imagination. In: *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies - volume 1. HLT '11*. Stroudsburg: Association for Computational Linguistics, 309–319.

Pane JF, Myers BA, Ratanamahatana CA. 2001. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies* **54**:237–264 DOI 10.1006/ijhc.2000.0410.

Rinard MC. 2012. Example-driven program synthesis for end-user programming: technical perspective. *Communications of the ACM* **55**:96–96 DOI 10.1145/2240236.2240259.

Rosson MB, Ballin J, Rode J. 2005. Who, what, and how: a survey of informal and professional web developers. In: *2005 IEEE symposium on visual languages and human-centric computing*. Piscataway: IEEE, 199–206 DOI 10.1109/VLHCC.2005.73.

Toomim M, Drucker SM, Dontcheva M, Rahimi A, Thomson B, Landay JA. 2009. Attaching UI enhancements to websites with end users. In: *CHI '09*. New York: ACM, 1859–1868 DOI 10.1145/1518701.1518987.

Tuchinda R, Szekely P, Knoblock CA. 2007. Building data integration queries by demonstration. In: *IUI '07*. New York: ACM, 170–179 DOI 10.1145/1216295.1216328.

Tuchinda R, Szekely P, Knoblock CA. 2008. Building Mashups by example. In: *IUI '08*. New York: ACM, 139–148 DOI 10.1145/1378773.1378792.

Wong J, Hong JI. 2007. Making mashups with marmite: towards end-user programming for the web. In: *CHI '07*. New York: ACM, 1435–1444 DOI 10.1145/1240624.1240842.

Wong J, Hong J. 2008. What Do We "Mashup" when We Make Mashups? In: *Proceedings of the 4th international workshop on end-user software engineering. WEUSE '08*. New York: ACM, 35–39 DOI 10.1145/1370847.1370855.

Yessenov K, Tulsiani S, Menon A, Miller RC, Gulwani S, Lampson B, Kalai A. 2013. A colorful approach to text processing by example. In: *UIST'13*. New York: ACM, 495–504 DOI 10.1145/2501988.2502040.

Zang N, Rosson MB. 2008. What's in a mashup? And why? Studying the perceptions of web-active end users. In: *IEEE symposium on visual languages and human-centric computing, 2008, VL/HCC 2008*. Piscataway: IEEE, 31–38 DOI 10.1109/VLHCC.2008.4639055.

Zang N, Rosson MB. 2009a. Web-active Users Working with Data. In: *CHI '09 extended abstracts on human factors in computing systems. CHI EA'09*. New York: ACM, 4687–4692 DOI 10.1145/1520340.1520721.

Zang N, Rosson MB. 2009b. Playing with information: how end users think about and integrate dynamic data. In: *IEEE symposium on visual languages and human-centric computing, 2009, VL/HCC 2009*. Piscataway: IEEE, 85–92 DOI 10.1109/VLHCC.2009.5295293.

Zang N, Rosson MB, Nasser V. 2008. Mashups: who? what? why? In: *CHI EA'08*. New York: ACM, 3171–3176 DOI 10.1145/1358628.1358826.

Zimmerman J, Rivard K, Hargraves I, Tomasic A, Mohnkern K. 2009. User-created forms as an effective method of human-agent communication. In: *CHI '09*. New York: ACM, 1869–1878 DOI 10.1145/1518701.1518988.