Give the People What They Want: Studying Non-Programmers Describing End-User Web Programming

Tak Yeon Lee tylee@umd.edu
Benjamin B. Bederson bederson@cs.umd.edu

Human-Computer Interaction Lab,
University of Maryland, College Park, MD 20742 USA

ABSTRACT
Understanding end-user's needs is a prerequisite for designing End-User Programming (EUP) environments. This paper reports on two qualitative studies that answer the following questions: 1) what do end-users want to improve on the Web; and 2) how do end-users without programming knowledge describe computational tasks? For the first question we asked 35 Web users about their daily activities and problems on the Web, and how they would improve it. As a result of this, we proposed functional requirements of future WebEUP systems that enable end-users to create, modify, and extend extensions with rich design details and interactivity. The second study focused on non-programmer's mental models about computational tasks. The interviewer asked 13 non-programmers to describe three programs (drawing a histogram, creating a custom filter, and combining information from multiple web pages). We summarized existing challenges and suggest design implications for building an easy, efficient, and expressive WebEUP system.

Author Keywords
End-User Programming; User Study; Mashup

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
Over the decades, the Web has become the most popular and convenient workbench for individuals and businesses in an incredible number of activities. Ironically, people are putting equal or even more time and effort in some activities, because of higher expectations and complexities. For example, online shoppers, who once had been happy with ordering products in a few clicks, now want to compare prices on different malls and even track the daily prices. Readers look for better tools for managing ever-growing channels of information. Fraudulent sites and deceptive opinion spam are huge concerns for consumers [21]. Doing repetitive tasks on poorly designed Web pages can be more painful than paper-based work environments, which are inefficient but malleable.

When the original site could not support the ever-increasing expectations and complexities of end-users, mashups [6, 28, 29, 31], browser extensions and scripts [1, 15, 18, 32] built by third-party programmers have improved the Web. Unfortunately, there are not enough third-party developers to address all 1.4 billion end-user's needs of 175 million websites [27]. Enabling the end-users to address their own needs is the goal of Web end-user programming (WebEUP) [4, 24].

Nevertheless, to build an easy, efficient, and expressive WebEUP system, we need a better understanding about Web users. Zang and Rosson [31] conducted a survey about popular mashups, but did not examine further than top-level categories (e.g. Blocking ADs). Although live collections such as the Chrome Web Store and ProgrammableWeb cover many common tasks, they cannot directly tell what end-users would like to use. Besides, as end-users mostly have no programming knowledge, we need to understand common challenges and opportunities to support them building programs for their specific needs. Artificial Intelligence now offers the potential of making it possible for end users to do much more than they ever could when they had to code every detail manually.

To better understand the potential of supporting end users to computationally customize their own environments, we conducted two qualitative studies that answer the following questions: 1) what do end-users want to improve on the Web; and 2) how do end-users without programming knowledge describe computational tasks?

In the first user study, we interviewed 35 Web users, observing 10 types of Web improvements that showed great potential. These include a wide spectrum of design details including runtime interactivity and extensibility. Based on these findings, we propose 7 functional components of future WebEUP systems. The second study focused on non-programmer's mental models about computational tasks. 13 non-programmers were asked to describe three programs doing simple computational tasks through conversational dialogues. Based on the challenges and opportunities found, we propose design implications for easy, efficient, and expressive WebEUP systems.

This paper's contributions are: 1) a description of the needs of current Web users, and a proposal of features of future WebEUP systems; 2) an examination of how non-programmers naturally describe computational tasks; and 3) a description of design implications.

END-USER PROGRAMMING ON THE WEB

Over the last decade, researchers and companies have developed a large number of WebEUP systems [4]. Those WebEUP tools commonly took a combination of three approaches: *scripting*, *visual programming*, and *programming by demonstration* (PBD). First, *script* languages for WebEUP [1, 15, 32] have simpler, more human-readable syntax than general-purpose languages (e.g. C or Java). However, end-users of script languages still need to learn complex language syntax and specify every detail manually. In order to make programming more accessible to non-programmers, many EUP systems employed *visual programming* techniques such as drag-and-drop for organizing graphical widgets of operations, flowcharts showing program structure [29, 33] and spreadsheets organizing large datasets [28]. While being effective for small educational projects (e.g. simple games and interactive animations), visual programming is often criticized for being less expressive and hard to accommodate large programs, so called Deutsch Limit [1]– "50 visual primitives on the screen at the same time". Due to these limitations, neither scripting nor visual programming approaches have resulted in easy, efficient, and expressive WebEUP systems yet.

Recently, more general EUP systems started using AI-based approaches to automate programming. Mixed-initiative interaction [11, 13], *Programming-by-Demonstration (PBD)* [5], and Programming-by-Example (or example-driven program synthesis) [10, 23, 30] allowed end-users to express program specifications in various ways, and then generate or search programs consistent with the specifications [23]. For instance, researchers have shown that AI can automatically synthesize simple programs such as string manipulation [10], text processing [30], and geometric drawing [3].

WebEUP systems also have begun applying PBE techniques. Tommim *et al* [27] allowed end-users to attach UI enhancements to arbitrary sites by selecting a few elements on the page. Nicholas and Lau [20] enabled end users to re-author a simplified mobile version of web applications by demonstrating the task and directly choosing page elements. Macias and Paterno [16] allowed users to modify the source code of a web page, and then the system creates a generalized modification of similar pages. We believe that similar AI-based techniques will have bigger roles in future EUP systems. Designing such systems would demand careful considerations of how the end-users and the computer work together.

STUDY 1: WEB END-USER PROGRAMMING NEEDS

Understanding end-users' needs is crucial when designing EUP systems [25]. This study explored the needs by asking

[1] Peter Deutsch made the comment at a talk on visual programming by Scott Kim and Warrent Robinett.

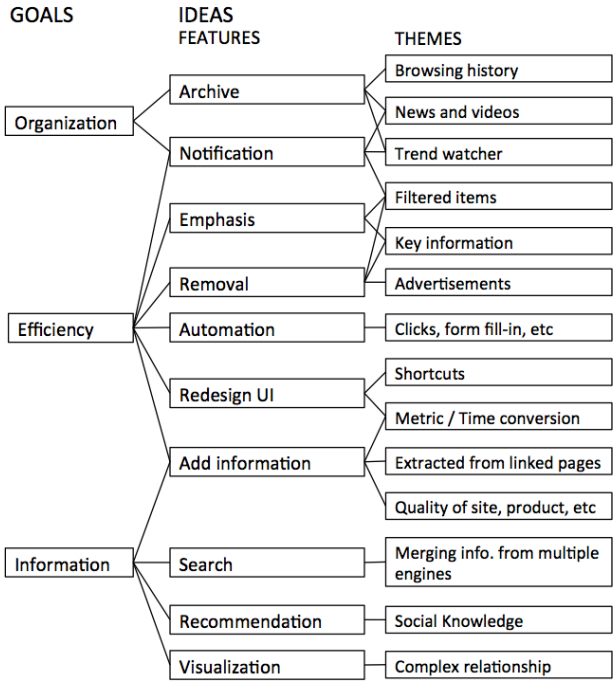

Figure 1. Concept map of the Web improvement ideas suggested by the participants.

Web users about their daily activities and problems they experience on the Web, and how they would improve the Web to create a better experience. As a result, we propose a set of functional requirements that would be beneficial when designing future WebEUP systems.

The data that was gathered for this study consisted of an on-line questionnaire and in-depth interview with 35 active Web users of varying gender (14 men and 21 women), programming expertise (10 out of 35 had programming experience), and age (between 24 and 37, avg. = 30.8). The subjects were recruited through word-of-mouth, university mailing lists, and social networks. We asked them about their daily activities on the Web, and how to enhance their experience by improving those web sites.

In the beginning of the study we tried to gather data using an online questionnaire containing 5 types of common problems and solutions: *mashing up information from linked pages*, *removing unwanted information (e.g. ADs)*, *summarizing web contents*, *auto-filling input forms*, and *converting dates and currency*. Thus the participants answered how often they experience each type of problems and described similar experiences they had. The result was assuring that they want to improve the given situations. However, we soon realized that the answers, which were gathered from the questionnaire, were neither rich enough to tell how they experience the problem or novel beyond the given examples. We thus changed our approach to conduct an in-depth interview without any example first, and then to show those examples only when the participant could not

understand the question. This approach was effective at gathering diverse unexpected input and focusing on the research question.

Method

We analyzed the data using a qualitative method inspired by grounded theory [2, 9]. First, we applied "open coding" to determine the range of ideas suggested by the participants [9]. We did this by having the first author review the questionnaire and the recorded interview. Every time an idea or goal was mentioned, he listed it. This resulted in 99 (average 3 per user) improvement ideas, with 10 shared main features, and 14 detailed themes in Figure 1. Then the first author iteratively constructed hierarchical relationship between them, following "axial coding" process of grounded theory [2, 9]. As result we found three high-level goals - organization, efficiency, and information (Figure 1.)

Finally, selective coding delimits the concepts to those that are relevant to the core theory. The resulting analysis is an integrated set of propositions [9], which describe a set of functional requirements of WebEUP systems.

Transcript with open codes
Subject #18: ... when I have things to buy, I daily check **["Trend watcher": theme]** online shopping malls to find the best deal. First, I go to Amazon.com and search the product **["Search": feature]**. Today I'm looking for Canon G15... There's no deal today. Usually I use an Excel spreadsheet to update daily the price trend **["Archive": feature]** and finding the best deal... I'd like to have a system that collects all the related information about the product **["Filtered Items": theme]**, especially the prices from different sellers and compares them. It will say like 'this is the lowest price for last three weeks' **["Notification": feature]**.
Open Coding memo
The subject regularly visits a shopping mall for watching price trend of a product. He **searches** for deals, and **archives** them in spreadsheet. The improvement should be able to **search** for deals, **archive** the price trend, and **send notifications.**

Table 1. An example of open coding. The bold words ["code": category] are identified from this transcript.

Ideas for Improving the Web

In this section we describe the three common goals of the Web improvement ideas.

Organization: End-users wanted to reduce their efforts of tracking their activities (e.g. browsing history) and timely information on the Web (e.g., price trend, deals, upcoming events, video and news).

> **[Archiving own activity]** *P17: I want to manage all the companies I've already applied.*
>
> **[Trend watcher]** *P22: In Amazon.com, a price goes up and down hourly. CamelCamelCamel.com is very useful, but it only supports Amazon.com. I wish there exists a price trend watcher for popular shopping sites.*

Efficiency: The most common goal was to do their tasks in more efficient ways. The end-users told many causes of inefficiency including advertisements, poor UI designs, verbose terms and conditions, and also lack of essential information. They suggested solutions such as emphasizing important information, removing unwanted elements, automating repetitive operations, redesigning UI, and attaching additional information. A few of them are shown below.

> **[Attaching additional information]** Details extracted from linked pages, file download, mouse and keyboard operations, and form fill-in
>
> *P16: The web page does not show the important deadlines and information of each event. I want the list shows what I need when printed out.*
>
> **[Automating repetitive operation]** File download, mouse and keyboard operations, and form fill-in
>
> *P7: At Google image search as well as Google scholar results (PDF or PS files for direct downloads, I wish each 'download' link would also provide a check box next to it, so that I can check off multiple entries and download them all.*
>
> **[Emphasizing key information]** Removing / dimming / folding unnecessary elements
>
> *P6: At any T&C page, I want to see the summary of terms and conditions important to me.*
>
> **[Conversion]** Currency, time zone, and metrics
>
> *P7: I visit some websites that involve online contests of some kind (e.g. coding competition), and if the contest start/end times are automatically converted, that will be useful...*
>
> **[Redesign UI]** P23: *When I'd like to buy a cloth, I cannot directly pick it up. But I need to remember the product name, scroll up and find the name in this option list. I'd love if there were a button to each product, so I can click it to buy.*

Information: Web end-users also wanted more information especially when making decision. For example, viewing other people's reviews about files, videos, sites, and products would be extremely beneficial. Cross-validating information from multiple-sources is a common technique for informed decisions.

> **[Social Review]** Quality of sites, page, product, streaming video links, and files
>
> *P12: At Underground TV show sites, I have to try every link until I find the first link working. By working, I mean the show must be high-resolution, not opening any popup, and most of all as little Ad as possible.*
>
> **[Automated Search]** P22: *When buying accessories or bags, I search Google images or YouTube videos to check how they would look in different situations.*

Characteristics of the Improvements

We found three common characteristics of the suggested improvement ideas. Summing them up, we hypothesize that future WebEUP systems must enable end-users to

create, modify, and extend extensions with wide range of design details and interactive UI components.

Wide range of design details

Even when two users wanted the same improvement, they might need entirely different designs in detail. For example, we observed three participants suggested *adding custom shortcuts* ideas having a wide spectrum of design details such as:

- **Triggers:** Define when the app should run. For creating shortcuts, triggers could be every page of the site, front page only, or another site.
- **Placement:** Where the shortcut will be attached in the page. (E.g. top / side of the page)
- **Content:** How the amount of information to be shown. (E.g. graphical button / text link / information widget)
- **Configuration:** Whether a new shortcut is inferred from the browsing history or configured manually, including how the configuration would work.

Other types of improvements such as **adding information** (*placement, contents*), **emphasis** (*filtering logic, style/font/color/size*), **notification** (*via email/message/page, frequency, and contents*), and **removal** (*Remove/Dim/Fold, Automated/Interactive*) showed similar variations. Moreover, many participants told us that the details are not final and might be revised later on. It indicates that end-users want to continuously modify and customize improvements rather than always use off-the-shelf extensions.

Runtime Interactivity

Due to the dynamic nature of the Web, it is very hard to build a robust program that satisfies every one's needs. According to our interview, end-users were concerned with potential errors or malfunctioning behaviors of the extensions they installed. They also had to turn the extensions off if there was no easy way to fix it.

> *P14: Extensions are useful for the first place, but it often hinders as well. For example, 'Popup blocker' often blocks necessary popups, so I turned it off from then.*

A common solution is to enable users to control the extension with interactivity. *Placement* and *Content* mentioned in the previous section are good examples of interactive UI components. For example, a participant explaining the idea of **Metric/Time conversion** wanted to show the converted numbers when the mouse cursor hovered over them. According to this pattern, future WebEUP systems should enable end-users to design interactivity components with event-handlers and sub-procedures.

Extending and Combining Ideas

End-users mostly started with single-feature ideas, however, and would then extend them with additional features. For instance, a participant started with an idea of *emphasizing filtered elements*, and extended the initial idea with *archiving* and *notification* features.

> *P4: I want these conferences are **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.*
>
> *… Instead of visiting this site regularly, I hope the **filter sends me emails** of weekly filtered CFPs on every Friday. The email would contain title, link, deadline, and relevant keywords.*
>
> *… I don't want it to send the same list of conferences again and again. So, there could be a **web page containing all the conferences** the filter has found. The weekly emails highlight newly added calls.*

Functional Requirements of Future WebEUP Systems

We propose functional requirements of future WebEUP systems that can cover the suggested ideas.

(1) Extraction is an essential feature of WebEUP used by most improvement ideas. Whenever an improvement needs to use information or modify elements on Web pages, the improvement must know the extraction query for the elements or information. To find the right query, WebEUP systems usually accept a set of examples from the end-user, and infer a consistent query (i.e. jQuery or XPath) for the examples [6, 7, 8, 15, 27, 29].

(2) Data Transformations including arithmetic operations, filtering and aggregating data / elements, and string manipulation in Table 2 are important features of design details and interactive UI components. However, because of its complex usage, most WebEUP systems support limited set of data transformation features targeted to advanced users. However, Gulwani *et al* [10] showed that complex string manipulation programs could be inferred from input and output examples. We believe future WebEUP systems employing AI-based techniques will be able to support data transformations without overloaded syntaxes or options.

Type	Details
Arithmetic	Add, Subtract, Multiply, Divide E.g. Add([1,2,3], [4,5,6]) → [5,7,9]
String manipulation	Get / Replace substring with regular expression
Filter	Filter by numeric equations (e.g. ==, !=, <, <=) E.g. Filter([15,3,4],divisible,3)→[15,3]
	Filter text elements matching with regular expression E.g. Match(["aa","ab","baa"],/aa/)→["aa","baa"]
Aggregate	Count, Sum, Average, etc. E.g. Sum([1,2,3]) → [6]
	Concatenate multiple lists of strings E.g. Concat(["3","6"],["br"])→["3br","6br"]

Table 2. Types of Data Transformation

(3) Creating new DOM elements such as text, image, and button elements, and even collections of multiple elements is required to cover the wide range of design details. However, similar to Data Transformations, most existing WebEUP systems have little support for creating DOM elements. An alternative way is to leverage existing elements on the Web and AI. For instance, Kumar *et al* [14] have proposed a structured-prediction algorithm that automatically transfers content from one page into the style and layout of another without showing a large number of style options to end-users.

(4) Modifying existing DOM elements is a common feature of web customizers [1, 16, 32], and also was required by our participants for *Custom Filter* and *Redesign* ideas. Modifiable element properties include visibility (hide/show element), size, font, and color, mostly done by editing CSS (Cascade Style Sheet) properties.

(5) Simulating keyboard and mouse interaction is required for automating repetitive tasks [1, 15, 18]. *Form fill-in* apps, for instance, simulate keyboard press events in a specific input box. Programming-by-Demonstration is the best way support this feature.

(6) Data Storage allows a script to remember and share data among users. For example, the *Archive* and *Redesign UI with shortcuts* ideas need private storage to store and retrieve user's behavior and preferences. Social improvements such as *Recommendation* require public storage to share information between users. To our knowledge, no existing WebEUP system allows end-users to set up and use their in-app storage.

(7) Event Handling and Sub-Procedures can improve robustness and usability of any app by adding runtime interactivity to it. Moreover, composing multiple sub-procedures is the best way to handle bigger programs. However, it is challenging for end-users to understand and handle complex structure of non-linear program structures. Most existing WebEUP systems do not fully support sub-procedures (except some script languages [1, 32]).

We believe the seven features above are equally important for expressive WebEUP systems. However, providing all of them in a single WebEUP system would be too complicated to learn for end-users. In the next user study, we look for novel opportunities by investigating non-programmer's mental models.

STUDY 2: NON-PROGRAMMER'S MENTAL MODEL OF COMPUTATIONAL TASKS

Programming languages are difficult to learn because their fundamental structure is not natural to non-programmers [22]. For instance, common programming language syntax such as looping [26], if-then conditional [19], and variable referencing [17] are quite different from spoken languages. Pane and Myers [22] identified the characteristics of non-programmers describing computational tasks in written statements. Our goal is similar, but paying attention to conversational dialogues and multi-modal intents including verbal statements, behavioral signals (e.g. page navigation, mouse click), gestures, and drawing on scratch paper. The data was gathered from 13 participants, who are 5 males and 8 females, average 33.3 years old (STD=5.86) with varying occupations and educational backgrounds. They were recruited by word-of-mouth. None of them had programming experience.

Method

The participants were asked to explain common computational tasks via conversational dialogue with the interviewer who acted like a hypothetical computer agent following the set of rules listed below.

- The computer (acted by the interviewer) can understand the programmer's intent (question, instruction, and statements) expressed in natural language, gestures, and drawings.
- The computer cannot infer semantic meaning of programmer's intent. For example, a rental posting "*4 Bedrooms 3 Lvl Townhome $1650 / 4br*" is merely a line of text to the computer. Thus the programmer has to explain all the semantic meanings.
- The computer executes the programmer's instruction only if it is unambiguous and complete. Otherwise the computer tried to resolve it through conversational dialogue like below:
  Programmer: Delete houses with less than three bedrooms.
  Computer: Please tell me more about 'houses with less 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?

We carefully designed three tasks to cover common situations in the improvement ideas. A sheet of paper containing basic instruction was provided, and the participants could draw or write anything on the paper as shown in Figures 2 and 3.

Task 1. Drawing Histogram: With 10 random numbers between 0 and 12, the participants were asked to complete a histogram having 4 empty bins (0~3, 3~6, 6~9, and 9~12). We wanted to observe how non-programmers would describe: 1) a set of common data-processing operations such as iteration, filter, and count and 2) drawing a bar graph.

Task 2. Custom Filter: We prepared 10 rental postings in Table 3 copied from Craigslist.com. The participants were asked to create a filter that removes houses having fewer than 3 bedrooms. The task consists of three sub-tasks. The first task is to extract sub-strings about bedrooms (e.g. 3br(s), 3bedroom(s), 3 BEDROOMS, 3/2, studio) from every posting. Second, a predicate for selecting sub-strings of fewer than three bedrooms is required. The last sub-task is to hide / remove the selected houses. This task aims to observe how non-programmers would decompose a big task into sub-tasks, specify extraction queries, and refer temporary variables such as sub-strings and selected postings.

"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! $2200 / 3br - 1948ft² - (Clarksburg)
- Lanham 2/1 new deck $1050 / 1818ft² - (Lanham)
- 4 Bedrooms 3 Lvl Townhome $1650 / 4br - (MD)
- 823 Comer Square Bel Air, MD 21014 $1675 / studio

… (6 more)…

Table 3. Housing rental postings collected from Craigslist.com

Task 3. Pulling information from detail pages: At Amazon.com, the product list does not show available colors of each product. Task 3 is to create a mashup script that automatically takes the available colors from the detail page, and attaches them to the product list page. When the mouse cursor is over the color thumbnails, the full-size photo should appear. The task is designed to observe how non-programmers would describe copy operations across multiple pages, and event handling.

Analysis

For qualitative and flexible interpretation of the data, we employed an iterative coding technique inspired by constructive grounded theory [2]. Instead of testing predefined hypotheses with unbiased measuring, the researcher was actively involved in the process of the participants doing the tasks, and iteratively constructed new hypotheses.

Each session was video recorded and transcribed by the first author. The transcript consists of sequential statements that contain conversational dialogue [2] between the participant and the interviewer, page navigation in the browser, mouse or finger pointing gesture, and drawings on the task instruction paper (Figures 2 and 3). Each task consists of 3-8 statements. Following the guidelines of grounded theory [2], we started open coding focusing on *how the participant described an instruction* and *what challenge the participant was experiencing*. While repeating the coding process, a few categories of the codes emerged as *programming styles* (Table 4), *imperative*

[2] If the participant spoke in another language (the native language of the interviewer), we translated it into English.

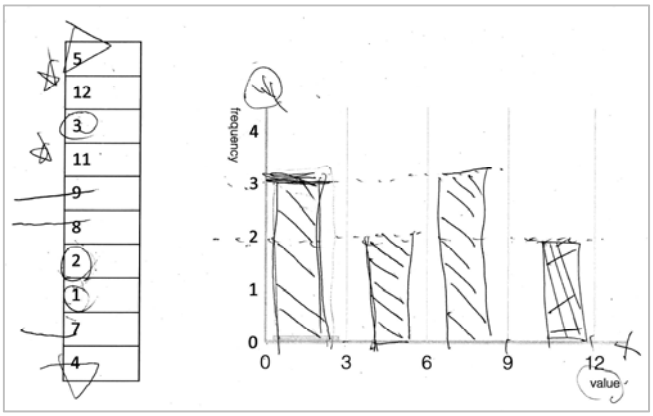

Figure 2. A histogram of Task 1 drawn by a participant.

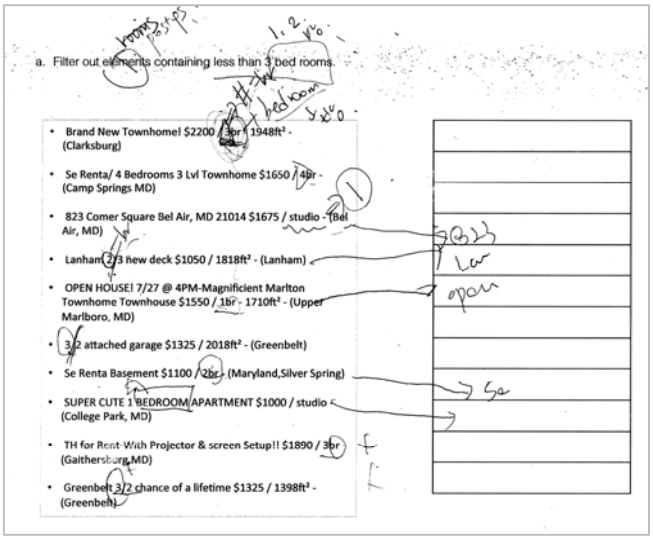

Figure 3. Task 2 sheet drawn by a participant

commands (Table 5), *ambiguities* (Table 6), and *multi-modal intent* (Table 7). In the tables, the left columns of the tables (e.g. Rule-Example in Table 4) contain codes, and the right columns show memos and statements. Due to the nature of qualitative approach we used, we report both findings and our interpretations together in the following section.

Findings and Discussion

General Challenges

At the beginning, the participants had no idea how to communicate with the computer. Most of their initial statements were ambiguous and underspecified as underlined in the examples below.

P5: First, I scan the list with my eyes and exclude them. They clearly stand out.

Computer: Why do they stand out?

P8: I'd order, "Exclude houses with one or two bedrooms."

Computer: How can I know the number of bedrooms?

P11: I'd ask computer to show available colors of this Columbia shirt.

Computer: Where can I get those colors?

To clarify ambiguous instructions, the computer asked further details. Through the conversational dialogues, the participants gradually understood what to explain and how specific they should be. We paid attention to the challenges the participants were experiencing. For example, most participants were clueless (see the following quotes) when explaining how to draw a bar chart in Task 1.

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?) What should I tell? Color?

Describing the right extraction query in Task 2 turned out to be challenging to many participants. Only 30% of the participants found all the rules. Although we did not emphasize correctness of the programs specified, the participants seemed to be more interested in completing the program than the correctness of their filtering logic.

Programming Styles

While traditional programming environments support imperative commands only, our non-programmer participants used five styles in combination as summarized in Table 4.

Style	Memos & Example Statements
Rule-Example *or* Example-Rule	Switching between rules and examples is a common approach to generating sound and complete logic. *P10: For here* (pointing the first column)*, we need 0, 1, and 2. Find numbers including zero, smaller than two. No, three.* *P4: Determine which bin each number is in. If the number is 1 (which is the first item), then count up this bin.*
Iterative Refinement	Instead of following logical order, non-programmers would start from broad, under-specified, or partial statements, and gradually add details. *P1: Attach pictures here. The pictures are taken from the page. The page is loaded from…* *P13: I guess it's easier to say, "Don't do", "Don't do 1 and 2". So, for 3, "not 2b, 2 b, 2/" and "Not 1b, 1 b, and 1/".*
Sequence of Imperative Commands	Using linear sequences of imperative commands is a traditional programming style. *P7: In the detail page, copy all these colors. And paste them after the name or next to the picture. Then go to the next product.* *P9: Looking at the columns here, count up the numbers 0,1,or 2 items. Put them here.*
Stepwise Selection / Filtering	To specify a set of elements or data, non-programmers use multiple steps of selection queries or conditional predicates. *P10: First find numbers '3' and '4'. Then, select the numbers ahead of 'br' or 'bedroom'.* *P6: …below the product title, there are sizes of the product and color text. In the color text, there's a list of images we are looking for.*
Event & Sub-Procedure	Assigning events that trigger sub-procedures. *P4: When we get the signal that the cursor is on each icon image, we replace the main image section with the bigger version of the color.* *P11: There would be a button to open the popup. 'Quick view'. When the button is clicked, it shows color or size information of the product.*

Table 4. Five Programming Styles of Non-Programmers

Imperative Commands

How would non-programmers refer to basic commands without prior knowledge? First of all, each command has many names as listed in the left column of Table 5. Often a participant referred to a command with different names even in a single task. We also found that those commands are ambiguous and incomplete without contextual information. All the participants relied on the current situation, multi-modal intent, and examples to clarify their intent.

Commands	Example Statements
Find / Search / Pick / Highlight / Look for / Choose within	*P9: In this next column, we need items going 6,7,8. So please find those 6,7,8 items, and draw bar in this column.* *P10: Find numbers including zero, smaller than two.. no, three."* *P13: I want highlight these things. (She selected all the color thumbnails in a detail page) Then I want to say 'copy'.*
Copy & Paste / Bring	*P5: Copy these colors. And Paste them out there.* *P11: Then I would ask computer to open a new detail page, and to bring the color information in the page if colors exist.*
Go / Open	*P7: Go into every product, and show available colors here below the description.*
Draw / Create	*P8: Draw bar graph for each number. Hmm... Create box as many as the number and stack them up.* *P15: Can we create a small window here (next to main image) showing colors?*
Filter out / Hide / Delete	*P13: I want to computer use control-F to search on the page, and do these things. And eliminate those lines found.*
Count / How many	*P14: Can I ask questions to computer? How many dots do we have between 3 and 6?* *P4: Determine which bin each number is in. If the number is 1 (which is the first item), then count up this bin.*

Table 5. Imperative Commands Used by Non-Programmers

Ambiguities in User's Intent

Natural language tends to be underspecified, and ambiguous [22]. Even after figuring out what they need to explain, most participants skipped a few types of crucial information summarized in Table 6. In our opinion, these ambiguities are not necessarily problematic. Provided that the missing information can be automatically inferred, they are good opportunities for more efficient and natural programming. For example, *Implicit Iteration* can be detected by checking whether the operand is the first item of any list.

Missing info.	Memos & Example Statements
Implicit iteration	Specifying process for the first few items, but expect it to be applied to every item. The resolution is repeating the same operation to the items in the list. *P14: Computer, take this and open a new tab. Then take different colors and put here.* *P20: Okay. I want you to display right here (pointing the first product) all the different color options. (Interviewer: "Only for this item?") No for every item.*

Ambiguous keywords of control structure	Control structures are difficult concepts. A few keywords (e.g. each, when, if, then) are used in iterations, conditionals, event-handlers, and set operations. The correct structure can be inferred from user-provided examples. *P19: Show colors when I click the photo.* *P23: If there are two items, draw it like this.*
Reference by value	Instead of variable names, a variable is referred by its value. Actual references can be inferred by the value. *P20: In this next column, we need items going 6,7,8. So please find those 6,7,8 items, and draw bar in this column.* *P21: As we have two, draw two here.*
Contextual Referencing	Instead of unambiguous variables or query, non-programmers use pronouns, data type, pointing gesture, and semantic proximity. *e.g. "Add photos here. (Interviewer: What Photo?) Oh... You know, there's only one set of photos."* *e.g. "Copy these colors. And Paste them out there."*
Skipping attribute key of objects	Without knowing the object structure, non-programmers would not use keys to get attributes. Having an Inspector UI can resolve this ambiguity. *e.g. "Here are '1 bedroom' and '2 bedrooms', they can be crossed-out. Erase 2 bedrooms. Remove '2br' and '1br'. Remove '2/3'. Remove 'studio'. Remove '1bedroom'."* *e.g. "Search these text, and I would say... erase them from the web page. Then it will just erase them (elements containing searched text)."*

Table 6. Common Ambiguities in Non-Programmer's Intent

Multi-modal intent

Every participant effectively used multi-modal intent including *voice*, *gesture*, and *page navigation*. *Voice* usually expressed the main part of their intent such as *command*, *logic* and other *control structures*. When referring to positions, shapes or elements, they commonly used pointing *gestures* with pronouns (e.g. this, here). They also used page *navigation* to change the scope of the operation. We found multi-modal intents are intuitive, natural and effective way for non-programmers to express computational tasks.

Multi-modal intent	Example Statements
Voice(command) + Gesture(position) + Navigation(scope)	*P4: These small icons of products in different colors. I want to copy an area that holds those icons, and link here (pointing below the price) in the list page.* *P12: Then, count the numbers of items here (pointing the number in 0~3) and draw (showing gesture of drawing rectangle).*
Voice(logic) + Gesture(position) + Navigation(scope)	*P6: (*Pointing at the '3br' part of the first item*) "This has three, which is more than two, so this is not the case."* *P13: And then, here's 11. No(pointing 0~3), no(3~6), no(6~9),... (She put one finger on the value in the list, and moves another finger and the pen on different ranges in the graph - similar with data cursor*)

Table 7. Multi-model Intent of Non-Programmers

Rationales and Challenges

We consistently observed that non-programmers expressed rationales (*why they need this statement or program)* and challenges (*why the problem is difficult to solve*). It was not clear that the participants expected the computer to perform any action in response. However, both in real-world conversation between people and mixed-initiative systems, rationales and challenges are effective ways to build mutual understandings about the scope, goals, activity, and constraints [13]. To our knowledge, how to utilize them in EUP systems has not been studied yet.

> *P6: Let's say I want to buy a polo shirt in pink color. I click this shirt, then it shows all the colors available. But people want to check the colors in listing as well. So, put some buttons here to click and check what colors are available.*
>
> *P6: While we can show images, which would be quite complex, I'd want you to do use color boxes.*
>
> *P12: By the way, because of 'at least three bedrooms', pick things larger than 3.*
>
> *P13: We can also secretly write number here (*center of each cell*) to remember, so track for afterward so we didn't make any mistake.*

DESIGN IMPLICATIONS

This section presents design implications in response to the findings in our studies for building easy, efficient, and expressive WebEUP systems. Although we focused on WebEUP, many of these implications are also applicable to general programming environments.

Unified Support For Five Programming Styles

The usability of programming environments could be improved by supporting a programming style (e.g. imperative, rule-based, and event-driven) suited for the task [22]. We also consistently observed that end-users employed combinations of the five programming styles we found such as an example scenario based on the first user study:

> Tim wants to create a custom notification on an online marketplace. He wants to receive emails when there is a new posting that contains the configured keywords. First, he read through the list and selects a few interesting postings. The system automatically infers the correct conditional logic (*Rule-Example*). Then he assigns a mouse-over (*Event)* to the top-most selected call, which will show detailed information about the post (*Sub-Procedure*). The system will automatically assign same event handler to all the items (*Rule-Example*). He tests each item, checking whether the email alert program he created will support him better.

To our knowledge, there is no WebEUP system equipped with unified support for the five programming styles. Scripting languages and visual programming environments [7, 15, 18, 29, 32] accommodate *imperative commands* and *event handling*, but they do not support *Rule-Example* and *iterative selection / refinements*. PBD / PBE systems [3, 10, 30] focus on *Rule-Example* and *Iterative refinement / selection* in specific task domains, but building bigger programs that contain event handlers and sequential commands would be challenging.

Mixed-Initiative Approach

In the second study we observed that the participants and the computer could converge to the solutions through conversational dialogues, which was analogous to mixed-initiative interaction model [12]. For instance, the *Rule-Example* and *Iterative refinements* of Table 5 and resolving *implicit iteration*, *control structure*, and *ambiguous keyword* in Table 6 involve a sort of human-computer collaboration where the end-users express their intent and then the computer infers and suggests corresponding solutions. With the mixed-initiative interaction, the learning curve of its users becomes much lower than menu-driven tools or traditional programming languages [11]. Also mixed-initiative systems can avoid the risk of a fully automated system by leveraging human skills. Nevertheless, the core challenge of mixed-initiative interaction [13], *grounding* to mutual understanding between end-users and the computer, still remains. The following sections will address ideas for creating common grounding.

Enhancing Iterative Refinement

When initially creating a new program, non-programmers often did not specify statements in the order that those statements should run. Instead, they started with quick and brief statements such as task outlines, goals, or partial solutions (e.g. the first step of a query, or a single case of filtering). All the non-programmers made a lot of mistakes in the first trials. Then they iteratively added details while trying it on examples. When the program looked good enough, they wrapped up the program by cleaning up unnecessary details.

Here we propose three potential ways to support iterative refinement. First, EUP systems should allow users to sketch programs with missing details, and recommend possible ways to fill in those holes. For example, if a user selected a button and modified its text, the computer could recommend 1) repeating the same operation on similar buttons in the page, 2) candidate logic of getting new text for selected buttons, and 3) adding next steps of operation on those buttons.

The second type of support is to automatically group redundant statements with refactoring techniques in software engineering. Imagine a scenario below (based on a story from the second study).

> Alice wants to remove houses on Craigslist that have one or two bedrooms. She first selects postings containing '1 br' and '2 br', and deletes them. She keeps on deleting houses with '1 bedrooms', '2 bedrooms', '1/', '2/', and so on. When she deletes all the houses, the computer refactors the series of delete commands into one such as - Delete postings containing '1' or '2' followed by 'bedroom', 'br' or '/'.

The last type of support is the creation of semi-automated unit tests. In the second task (Custom Filter task), we observed that 70% of participants had underspecified filters that were missing at least one string pattern. In order to prevent this, the EUP system should enable users to see the result of the current program, and quickly verify it.

FUTURE WORK

Instead of giving a conclusive answer, this paper proposed many ideas and open questions. In fact, the authors are currently developing a novel WebEUP system based on the ideas. For instance, tapping the channel of user's behavioral intent to mixed-initiative systems is an interesting direction that requires further exploration. Also, how to support mutual understanding between computer and human requires iterative design process.

CONCLUSION

Understanding end-users' needs is a crucial part of the EUP system design process. The two qualitative user studies in this paper investigated 1) what end-user's want to build with WebEUP, and 2) how non-programmers would express computational tasks. From the first study we suggested 7 functional requirements of future WebEUP systems that enable end-users to create, modify, and extend extensions with design details and interactive UI components. The second study result consists of challenges and opportunities of non-programmers showing their intent. Although we set WebEUP as our target domain, we believe that the insights are informative to improve usability of general programming environments.

ACKNOWLEDGMENTS

<Removed for blind review>

REFERENCES

1. Bolin, M., Webber, M., Rha, P., Wilson, T., and Miller, R.C. Automation and customization of rendered web pages. *Proceedings of the 18th annual ACM symposium on User interface software and technology*, ACM (2005), 163–172.
2. Charmaz, K. Grounded Theory: Objectivist and Constructivist Methods. (2000).
3. Cheema, S., Gulwani, S., and LaViola, J. QuickDraw: improving drawing experience for geometric diagrams. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2012), 1037–1064.
4. Cypher, A., Dontcheva, M., Lau, T., and Nichols, J. *No Code Required: Giving Users Tools to Transform the Web*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.
5. Cypher, A., Halbert, D.C., Kurlander, D., et al., eds. *Watch what I do: programming by demonstration*. MIT Press, Cambridge, MA, USA, 1993.
6. Ennals, R., Brewer, E., Garofalakis, M., Shadle, M., and Gandhi, P. Intel Mash Maker: join the web. *SIGMOD Rec. 36*, 4 (2007), 27–33.
7. Fujima, J., Lunzer, A., Hornbæk, K., and Tanaka, Y. Clip, connect, clone: combining application elements to build custom interfaces for information access. *Proceedings of the 17th annual ACM symposium on*

User interface software and technology, ACM (2004), 175–184.
8. Gardiner, S., Tomasic, A., Zimmerman, J., Aziz, R., and Rivard, K. Mixer: mixed-initiative data retrieval and integration by example. *Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction - Volume Part I*, Springer-Verlag (2011), 426–443.
9. Glaser, B.G. and Strauss, A.L. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine, 1967.
10. Gulwani, S. Automating string processing in spreadsheets using input-output examples. *SIGPLAN Not. 46*, 1 (2011), 317–330.
11. Guo, P.J., Kandel, S., Hellerstein, J.M., and Heer, J. Proactive wrangling: mixed-initiative end-user programming of data transformation scripts. *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM (2011), 65–74.
12. Horvitz, E. Principles of mixed-initiative user interfaces. *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, ACM (1999), 159–166.
13. Horvitz, E.J. Reflections on Challenges and Promises of Mixed-Initiative Interaction. *AI Magazine 28*, 2 (2007), 3.
14. Kumar, R., Talton, J.O., Ahmad, S., and Klemmer, S.R. Bricolage: example-based retargeting for web design. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2011), 2197–2206.
15. Leshed, G., Haber, E.M., Matthews, T., and Lau, T. CoScripter: automating & sharing how-to knowledge in the enterprise. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2008), 1719–1728.
16. Macías, J.A. and Paternò, F. Customization of Web applications through an intelligent environment exploiting logical interface descriptions. *Interacting with Computers 20*, 1 (2008), 29–47.
17. Miller, L.A.. Natural language programming: styles, strategies, and contrasts. *IBM Syst. J. 20*, 2 (1981), 184–215.
18. Miller, R.C., Chou, V.H., Bernstein, M., et al. *Inky: A Sloppy Command Line for the Web with Rich Visual Feedback*. .
19. Myers, B.A., Pane, J.F., and Ko, A. Natural programming languages and environments. *Commun. ACM 47*, 9 (2004), 47–52.
20. Nichols, J. and Lau, T. Mobilization by demonstration: using traces to re-author existing web sites. *Proceedings of the 13th international conference on Intelligent user interfaces*, ACM (2008), 149–158.
21. Ott, M., Cardie, C., and Hancock, J. Estimating the Prevalence of Deception in Online Review Communities. *Proceedings of the 21st International Conference on World Wide Web*, ACM (2012), 201–210.
22. Pane, J.F., Myers, B.A., and Ratanamahatana, C.A. Studying the language and structure in non-programmers' solutions to programming problems. *Int. J. Hum.-Comput. Stud. 54*, 2 (2001), 237–264.
23. Rinard, M.C. Example-driven program synthesis for end-user programming: technical perspective. *Commun. ACM 55*, 8 (2012), 96–96.
24. Rode, J., Rosson, M.B., and Qui˜nones, M.A.P. End User Development of Web Applications. In H. Lieberman, F. Paternò and V. Wulf, eds., *End User Development*. Springer Netherlands, 2006, 161–182.
25. Rosson, M.B., Ballin, J., and Rode, J. Who, what, and how: a survey of informal and professional Web developers. *2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, (2005), 199–206.
26. Soloway, E., Bonar, J., and Ehrlich, K. Cognitive strategies and looping constructs: an empirical study. *Commun. ACM 26*, 11 (1983), 853–860.
27. Toomim, M., Drucker, S.M., Dontcheva, M., Rahimi, A., Thomson, B., and Landay, J.A. Attaching UI enhancements to websites with end users. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2009), 1859–1868.
28. Tuchinda, R., Szekely, P., and Knoblock, C.A. Building Mashups by example. *Proceedings of the 13th international conference on Intelligent user interfaces*, ACM (2008), 139–148.
29. Wong, J. and Hong, J.I. Making mashups with marmite: towards end-user programming for the web. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2007), 1435–1444.
30. Yessenov, K., Tulsiani, S., Menon, A., et al. A Colorful Approach to Text Processing by Example. 2013.
31. Zang, N., Rosson, M.B., and Nasser, V. Mashups: who? what? why? *CHI '08 Extended Abstracts on Human Factors in Computing Systems*, ACM (2008), 3171–3176.
32. Greasemonkey. https://addons.mozilla.org/en-US/firefox/addon/greasemonkey/.
33. Yahoo! Pipes. http://pipes.yahoo.com/pipes/.