Collaborative Scripting for the Web

Allen Cypher, Clemens Drews, Eben Haber, Eser Kandogan, James Lin, Tessa Lau, Gilly Leshed, Tara Matthews, Eric Wilcox

INTRODUCTION

Employees in modern enterprises engage in a wide variety of complex, multi-step processes as part of their day-to-day work. Some of these processes are routine and tedious, such as reserving conference rooms or monitoring call queues. Other processes are performed less frequently but involve intricate domain knowledge, such as making travel arrangements, ordering new equipment, or selecting insurance beneficiaries. Knowledge about how to complete these processes is distributed throughout the enterprise; for the more complex or obscure tasks, employees may spend more time discovering how to do process than in actually completing it.

However, what is complex or obscure for one employee may be routine and tedious for another. Someone who books conference rooms regularly should be able to share this knowledge with someone for whom reserving rooms is a rare event. The goal of our CoScripter project is to provide tools that facilitate the capture of how-to knowledge around enterprise processes, share this knowledge as human-readable scripts in a centralized repository, and make it easy for people to run these scripts in order to learn about and automate these processes.

Our "social scripting" approach has been inspired by the growing class of social bookmarking tools such as del.icio.us [4] and dogear [7]. Social bookmarking systems began as personal information management tools to help users manage their bookmarks. However they quickly demonstrated a social side-effect, where the shared repository of bookmarks became a useful resource for others to consult about interesting pages on the web. In a similar vein, we anticipate CoScripter being used initially as a personal task automation tool, for early adopters to automate tasks they find repetitive and tedious. When shared in a central repository, these automation scripts become a shared resource that others can consult to learn how to accomplish common tasks in the enterprise.

RELATED WORK

CoScripter was inspired by Chickenfoot [1], which enabled end users to customize web pages by writing simplified JavaScript commands that used keyword pattern matching to identify web page components. CoScripter uses similar heuristics to label targets on web pages, but CoScripter's natural language representation for scripts requires less programming ability than Chickenfoot's JavaScript-based language.

Keyword Commands [6] allowed users to enter unstructured text which the system would interpret as a command to take on the current web page (by compiling it down to a Chickenfoot statement). While a previous version of CoScripter used a similar approach to interpret script steps, feedback from users indicated that the unstructured approach produced too many false interpretations, leading to the current implementation that requires steps to obey a specific grammar.

Tools for doing automated capture and replay of web tasks include iMacros[1] and Selenium[2]. Both tools function as full-featured macro recorder and playback systems. However CoScripter is different in that scripts are recorded as natural language scripts that can be modified by the user without having to understand a programming language.

CoScripter's playback functionality was inspired by interactive tutorial systems such as Eager [2] and Stencils [3]. Both systems used visual cues to direct user attention to the right place on the screen to complete their task. CoScripter expands upon both of those systems by providing a built-in sharing mechanism for people to create and reuse each others' scripts.

THE COSCRIPTER SYSTEM

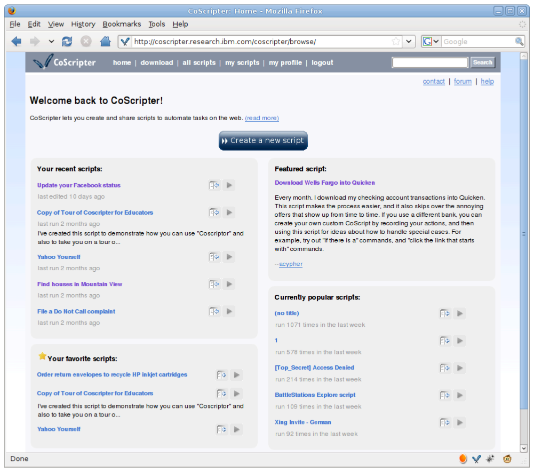
CoScripter interface

CoScripter consists of two main parts: a centralized, online repository of scripts (Figure 1), and a Firefox extension that facilitates creating and running scripts (Figure 2). The two work together to provide a seamless experience. Users start by browsing the repository to find interesting scripts; the site supports full-text search as well as user-contributed tags and various sort mechanisms to help users identify scripts of interest.

Once a script has been found, the user can click a button to load the script into the CoScripter sidebar. The sidebar provides an interface to step through the script, line by line. At

[1]http://www.iopus.com/imacros/firefox/
[2]http://seleniumhq.org/

Under review.


CoScripter: Home - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://coscripter.research.ibm.com/coscripter/browse/
CoScripter home | download | all scripts | my scripts | my profile | logout
Search
contact | forum | help
Create a new script
Done

Welcome back to CoScripter!

CoScripter lets you create and share scripts to automate tasks on the web. (read more)

Your recent scripts:

Update your Facebook status
last edited 10 days ago

Copy of Tour of Coscripter for Educators
last run 2 months ago
I've created this script to demonstrate how you can use "Coscriptor" and also to take you on a tour o...

Yahoo Yourself
last run 2 months ago

Find houses in Mountain View
last run 2 months ago

File a Do Not Call complaint
last run 2 months ago

Your favorite scripts:

Order return envelopes to recycle HP inkjet cartridges

Copy of Tour of Coscripter for Educators
I've created this script to demonstrate how you can use "Coscriptor" and also to take you on a tour o...

Yahoo Yourself

Featured script:

Download Wells Fargo into Quicken

Every month, I download my checking account transactions into Quicken. This script makes the process easier, and it also skips over the annoying offers that show up from time to time. If you use a different bank, you can create your own custom CoScript by recording your actions, and then using this script for ideas about how to handle special cases. For example, try out "if there is a" commands, and "click the link that starts with" commands.

--acypher

Currently popular scripts:

(no title)
run 1071 times in the last week

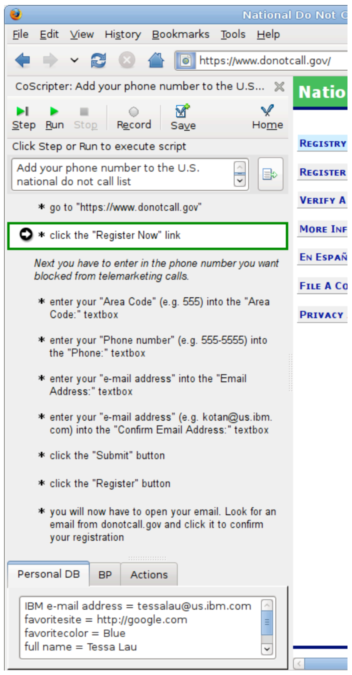
1
run 578 times in the last week

[Top_Secret] Access Denied
run 214 times in the last week

BattleStations Explore script
run 109 times in the last week

Xing Invite - German
run 92 times in the last week

Figure 1. CoScripter script repository



CoScripter: Add your phone number to the U.S...

Click Step or Run to execute script

Add your phone number to the U.S. national do not call list

* go to "https://www.donotcall.gov"
* click the "Register Now" link

Next you have to enter in the phone number you want blocked from telemarketing calls.

* enter your "Area Code" (e.g. 555) into the "Area Code:" textbox
* enter your "Phone number" (e.g. 555-5555) into the "Phone:" textbox
* enter your "e-mail address" into the "Email Address:" textbox
* enter your "e-mail address" (e.g. kotan@us.ibm.com) into the "Confirm Email Address:" textbox
* click the "Submit" button
* click the "Register" button
* you will now have to open your email. Look for an email from donotcall.gov and click it to confirm your registration

IBM e-mail address = tessalau@us.ibm.com
favoritesite = http://google.com
favoritecolor = Blue
full name = Tessa Lau

Figure 2. CoScripter sidebar (Firefox extension)

each step, CoScripter highlights the step to be performed by drawing a green box around the target on the webpage. By clicking CoScripter's "step" button, the user indicates that CoScripter should perform this step automatically and the script advances to the following step.

Scripts can also be run automatically to completion. The system iterates through the lines in the script, automatically performing each action until either a step is found that cannot be completed, or the script reaches the end. Steps that cannot be completed include steps that instruct the user to click a button that does not exist, or to fill in a field without having sufficient information to do so.

CoScripter also provides the ability to record scripts by demonstration. Using the same sidebar interface, users can create a new blank script. The user may then start demonstrating a task by performing it directly in the web browser. As the user performs actions on the web, CoScripter records a script describing the actions being taken. The resulting script can be saved directly to the wiki. Scripts can be either public (visible to all) or private (visible only to the creator).

Scripting language

CoScripter's script representation consists of a structured form of English. Steps in a script are both human-readable and machine-interpretable. Because they are human-readable, scripts double as written instructions for performing a task on the web. Because they are machine-interpretable, we can provide tools that can automatically record and understand scripts in order to execute them on a user's behalf.

Examples of script steps include:

- go to "http://google.com"
- enter "sustainability" into the "Search" textbox
- click the "Google Search" button

Recording works by adding event listeners to clicks and other events on the HTML DOM. When a user action is detected, we generate a step in the script that describes the user's action. Each step is generated by filling out a template corresponding to the type of action being performed. For example, `click` actions always take the form "click the TargetLabel TargetType". The TargetType is derived from the type of DOM node on which the event listener was triggered (e.g., an anchor tag has type "link", while an `<INPUT type="text">` has type "textbox"). The TargetLabel is extracted from the source of the HTML page using heuristics, as described below in the "Human-readable labels" section.

Keyword-based interpreter

The first generation CoScripter (described in [5]) used a keyword-based algorithm to parse and interpret script steps within the context of the current web page. The algorithm begins by enumerating all elements on the page with which users can interact (e.g., links, buttons, text fields). Each element is annotated with a set of keywords that describe its function (e.g., `click` and `link` for a hypertext link; `enter` and `field` for a textbox). It is also annotated with a set of words that describe the element, including its label, caption, accessibility text, or words nearby. A score is then assigned to each element by comparing the bag of words in the input step with the bag of words associated with the element; words that appear in both contribute to a higher score. The highest scoring element is output as the predicted target.

Once an element has been found, its type dictates the action to be performed. For example, if the element is a link, the action will be to `click` on it; if the element is a textbox, the action will be to `enter` something into it. Certain actions, such as `enter`, also require a value. This value is extracted by removing words from the instruction that match the words associated with the element; the remaining words are predicted to be the value.

The design of the keyword-based algorithm causes it to always predict some action, regardless of the input given. While this approach worked surprisingly well at guessing which command to execute on each page, the user study results below demonstrate that this approach often produced unpredictable results.

Grammar-based interpreter

The second generation CoScripter used a grammar-based approach to parse script steps. Script steps are parsed using a LR(1) parser, which converts from the textual representation of the step to an object representing a web command, including the parameters necessary to interpret that command on a given web page (such as the label and type of the interaction target).

The verb at the beginning of each step indicates the action to perform. Optionally a value can be specified, followed by a description of the target label and an optional target type. Figure 3 shows an excerpt of the BNF grammar recognized by this parser, for `click` actions.

Human-readable labels

CoScripter's natural-language script representation requires specialized algorithms to be able to record user actions and play them back. These algorithms are based on a set of heuristics for accurately generating a human-readable label for interactive widgets on web pages, and for finding such a widget on a page given the human-readable label.

CoScripter's labelling heuristics take advantage of accessibility metadata, if available, to identify labels for textboxes and other fields on web pages. However, many of today's web pages are not fully accessible. In those situations, CoScripter incorporates a large library of heuristics that use the DOM structure to guess labels for a variety of elements. For example, to find a label for a textbox, we might look for text to the left of the box (represented as children of the textbox's parents that come before it in the DOM hierarchy), or we might use the textbox's name or tooltip.

Our algorithm for interpreting a label relative to a given web page consists of iterating through all candidate widgets of the desired type (e.g., textboxes) and generating the label for the widget. If the label matches the target label, then we have found the target, otherwise we keep searching until we have exhausted all the candidates on the current page.

```
Click ::= click {on} the TargetSpec
TargetSpec ::= TargetLabel {TargetType}
TargetLabel ::= "String"
TargetType ::= link | button | item | area
```

Figure 3. Excerpt from CoScripter's grammar

Variables and the personal database

CoScripter provides a simple mechanism to generalize scripts, using a feature we call the "personal database." Most scripts include some steps that enter data into forms. This data is often user-specific, such as a name or a phone number. A script that always entered "Joe Smith" when prompted for a name would not be useful to many people besides Joe. To make scripts more appealing to a wider variety of users, we wanted to provide an easy mechanism for script authors to generalize their scripts and abstract out variables that should be user-dependent, such as names and phone numbers.

Variabilization is provided through a small extension to the sloppy scripting language. Anywhere a literal text string can appear, we also support the use of the keyword "your" followed by a variable name reference. For example:

- enter your "full name" into the "Search" textbox

These variabilized steps can be recorded at script creation time if the corresponding variable appears in the script author's personal database. The personal database is a text file containing a list of name-value pairs (lower left corner of Figure 2). Each name value pair is interpreted as the name of a variable and its value, so that for example the text "full name = Mary Jones" would be interpreted as a variable named "full name" with value "Mary Jones". During recording, if the user enters a value into a form field that matches the value of one of the personal database variables, the system automatically records a variabilized script step as shown above, rather than recording the user's literal action. Users can also variabilize steps post-hoc by editing the text of the script and changing the literal string to a variabilized reference.

Variables are automatically dereferenced at runtime in order to insert each user's personal inforamtion into the script during execution. When a variable reference is encountered, the personal database is queried for a variable with the specified name. If found, the variable's value will be substituted as the desired value for that script step. If not found, script execution will pause and ask the user to manually enter in the desired information (either directly in to the web form, or into the personal database where it will be picked up during the next execution).

EVALUATING COSCRIPTER'S EFFECTIVENESS

We have conducted several studies to measure CoScripter's effectiveness at facilitating knowledge sharing for web-based tasks across the enterprise. For our first study, we interviewed 18 employees of a large corporation to understand what business processes were important in their jobs, and to identify existing practices and needs for learning and sharing these processes.

In a second study, we deployed CoScripter in a large organization for a period of several months. Based on usage log analysis and interviews with prominent users, we were able to determine how well CoScripter supported the needs discovered in the initial study, and learn how users were adapting the tool to fit their usage patterns. We also identified opportunities for future development.

STUDY 1: UNDERSTANDING USER NEEDS

Participants

We recruited 18 employees of the large corporation in which CoScripter was deployed. Since our goal was to understand business process use and sharing, we contacted employees who were familiar with the organization's business processes and for whom procedures were a substantial part of their everyday work. Our participants had worked at the corporation for an average of 19.4 years (ranging from a few weeks to 31 years, with a median of 24 years). Twelve participants were female. Seven participants served as assistants to managers, either administrative or technical; 6 held technological positions such as engineers and system administrators; 3 were managers; and 2 held human resource positions. The technology inclination ranged from engineers and system administrators on the high end to administrative assistants on the low end. All but two worked at the same site within the organization.

Method

We met our participants in their offices. Our interviews were semi-structured; we asked participants about their daily jobs, directing them to discuss processes they do both frequently and infrequently. We prompted participants to demonstrate how they carry out these procedures, and probed to determine how they obtained the knowledge to perform them and their practices for sharing them. Sessions lasted approximately one hour and were video-recorded with participants' permission (only one participant declined to be recorded).

Results

We analyzed data collected in the study by carefully examining the materials video-recordings, their transcripts, and field notes. We coded the materials, marking points and themes that referred to our exploratory research goals: (1) common, important business processes, and (2) practices and needs for learning and sharing processes.

Note that with our study method, we did not examine the full spectrum of the interviewees' practices, but only those that they chose to talk about and demonstrate. Nonetheless, for some of the findings we present quantified results based on the coded data. In the rest of this chapter, wherever a reference is made to a specific participant, they are identified by a code comprised of two letters and a number.

What Processes Do Participants Do?

We asked our participants to describe business processes they perform both frequently and infrequently. The tasks they described included web-based processes, other online processes (e.g., processes involving company-specific tools, email, and calendars), and non-computer-based processes. Given that CoScripter is limited to a Firefox web browser, we focused on the details of web-based tasks, but it is clear that many business processes take place outside a browser.

We found that there was a significant amount of overlap in the processes participants followed. Seventeen participants discussed processes that were mentioned by at least one other participant. Further, we found that a core set of processes was performed by many participants. For example, 14 participants described their interactions with the online expense reimbursement system. Some other frequently mentioned processes include making travel arrangements in an online travel reservation system, searching for and reserving conference rooms, and purchasing items in the procurement system. *These findings show that there exists a common base of processes that many employees are responsible for performing.*

Despite a common base of processes, we observed considerable personal variation, both within a single process and across the processes participants performed. A common cause for variation within a single process was that the exact input values to online tools were often different for each person or situation. For example, DS1 typically travels to a specific destination, whereas LH2 flies to many different destinations. We observed variations like these for all participants. Secondly, there were a number of processes that were used by only a small number of people. Eleven participants used web-based processes not mentioned by others. For instance, TD1, a human resource professional, was the only participant to mention using an online system for calculating salary base payments. *These findings suggest that any process automation solution would need to enable personalization for each employee's particular needs.*

Our participants referred to their processes using various qualities, including familiarity, complexity, frequency, involvement, etc. Some of these qualities, such as complexity, were dependent on the task. For example, purchasing items on the company's procurement system was a challenging task for most participants. Other qualities, such as familiarity and frequency, varied with the user. For example, when demonstrating the use of the online travel system, we saw CP1, a frequent user, going smoothly through steps which DM1 struggled with and could not remember well. We observed that tasks that were *frequent* or *hard-to-remember* for a user may be particularly amenable to automation.

Frequent processes. Participants talked about 26 different processes they performed frequently, considering them tedious and time-consuming. For example, JC1 said: "[I] pay my stupid networking bill through procurement, and it's the same values every month, nothing ever changes." Automation of frequent processes could benefit users by speeding up the execution of the task.

Hard-to-remember processes. At least 8 participants mentioned processes they found hard to remember. We observed two factors that affected procedure recall: its complexity and its frequency (though this alignment was not absolute). In general, tasks completed infrequently, or that had complex steps, were often considered hard-to-remember. For example, a user of the procurement system said, "It's not so straightforward, so I always have to contact my assistant who contacts someone in finance to tell me these are the right codes that you should be using." Alternatively, although AB1 frequently needed to order business cards for new employees, it involved filling out multiple data fields she found hard to remember. *Automation could benefit users of hard-to-remember tasks by relieving the need to memorize their steps.*

How Do Participants Share Knowledge?

An important goal of our interviews was to develop an understanding of the sharing practices that exist for procedural knowledge. Thus, we asked participants how they learned to perform important processes, how they captured their procedural knowledge, and how and with whom they shared their own procedural knowledge.

Learning. Participants listed a variety of ways by which they learned procedures, most of them listing more than one approach. Figure 1 shows the different ways people learned procedures. Note that the categories are not mutually exclusive. Rather, the boundaries between contacting an expert, a colleague, and a mentor were often blurred. For example, KR1 needed help with a particular system and mentioned contacting her colleague from next door, who was also an expert with the system. Participants often said they would start with one learning approach and move to another if they were unsuccessful. Interestingly, although each participant had developed a network of contacts from which to learn, they still use trial-and-error as a primary way of getting through procedures: 13 out of 18 participants mentioned this approach for obtaining how-to knowledge. *This finding indicates that learning new procedures can be difficult, and people largely rely on trial-and-error.*

For maintaining the acquired knowledge, 15 out of 18 participants kept or consulted private and/or public repositories for maintaining their knowledge. For the private repositories, participants kept bookmarks in their browsers as pointers for procedures, text files with "copy-paste" chunks of instructions on their computer desktops, emails with lists of instructions, as well as physical binders, folders, and corkboards with printouts of how-to instructions. Figure 2 shows sample personal repositories. *Users create their own repositories to remember how to perform tasks.*

One participant noted an important problem with respect to capturing procedural knowledge: "Writing instructions can be pretty tedious. So, if you could automatically record a screen movie or something, that would make it easier to [capture] some of the instructions. It would be easy to get screenshots instead of having to type the stories." *This feedback indicates that an automatic procedure recording mechanism would ease the burden of capturing how-to knowledge (for personal or public use).*

Sharing. Eleven participants reported that they maintain public repositories of processes and "best practices" they considered useful for their community of practice or department. These repositories were commonly shared as databases accessible through the corporate email client. Four participants noted that although their repository was publicly open for posting and reading, they were the sole users of it. DS2 said: "I developed this, and I sent a link to everyone. People still come to me, and so I tell them: well you know it is posted, but let me tell you." Using public repositories suggests that participants sought an open forum for others to find and share their how-to knowledge. However, knowledge was distributed across multiple single-authored repositories, which may have made it harder for learners to find. In fact, seven participants reported that they had resorted to more proactive sharing methods, such as sending colleagues useful procedures via email.

Fifteen participants said they serve as sources of information for many others (this number is high due to our focus on recruiting senior members of the organization or people with information sharing as a job function). These participants earned their position as knowledge sharers by serving as mentors for other employees; by assuming this role as part of their position (e.g., administrative assistants, HR professionals, etc.); and by simply being known in their community as experts for various processes.

People in need find experts in a variety of ways. One participant said, "I'm usually one of the people who gets called and [contacted via IM] and emailed because I've been at it for too long." An experienced administrative assistant was listed as an expert for the administrative assistants' community in the company's directory: "They will contact you if they have questions regarding, let's say, archiving. Because I'm on the list, they'll call me."

These results give examples of employees who seek to share their how-to knowledge within the company and ways in which learners find the knowledge. Nonetheless, *our results show that sharing is time-consuming for experts and shared repositories are seldom used by learners, suggesting that sharing could be bolstered by new mechanisms for distributing procedural knowledge.*

Discussion

Our initial study has shown that there is a need for tools that support the automation and sharing of how-to knowledge in the enterprise. We observed a core set of processes used by many participants, as well as less-common processes. Processes that participants used frequently were considered routine and tedious, whereas others were considered hard-to-remember. Automating such procedures could accelerate frequent procedures and overcome the problem of recalling hard-to-remember tasks. Our data also suggest that existing solutions do not adequately support the needs of people learning new processes. Despite rich repositories and social ties with experts, mentors, and colleagues, people habitually apply trial-and-error in learning how to perform their tasks. New mechanisms are needed for collecting procedural knowledge to help people find and learn from it.

We also found that people who were sources of how-to knowledge needed better ways for capturing and sharing their knowledge. These people were overloaded by writing lengthy instructions, maintaining repositories with "best practices," and responding to requests from others. Also, distribution of their knowledge was restricted due to limited use of repositories and bounds on the time they could spend helping others. As such, automating the creation and sharing of instructions could assist experts in providing their knowledge to their community and colleagues.

STUDY 2: REAL-WORLD USAGE

The first interview study explored needs and practices of sharing how-to information. In a second study, we examined how well CoScripter supports these needs and practices through analysis of usage logs and interviews with regular users. The purpose of these studies was threefold: (1) determine how well CoScripter supported the user needs discovered in Study 1, (2) learn how users had adapted CoScripter to their needs, and (3) uncover outstanding problems to guide future CoScripter development.

Log Analysis: Recorded User Activity

CoScripter has been available for use within the IBM corporation starting from November 2006 through the time of this research (September 2007). Usage logs provide a broad overview of how CoScripter has been used in one company, while the interviews in the following section provide more in-depth examples of use. In this section we present an initial analysis of quantitative usage, with a content analysis left to future work. The data reported here excludes the activities of CoScripter developers.

Script Usage Patterns

Users are able to view scripts on the CoScripter wiki anonymously; registration is only required to create, modify, or run scripts. Of the 1200 users who registered, 601 went on to try out the system. A smaller subset of those became regular users, either recently or in the past.

We define active users as people who have run scripts at least five times with CoScripter, used it for a week or more, and used it within the past two months. Fifty-four users (9% of 601 users) are active users. These users, on average, created 2.1 scripts; ran 5.4 distinct scripts; ran scripts 28.6 times total; and ran a script once every 4.5 days. While 9% may seem to be a relatively low fraction, we are impressed by the fact that 54 people have voluntarily adopted CoScripter, and derive enough value from it to make it a part of their work practices.

We define past users as those who were active users in the past, but have not used CoScripter in the past two months. This category consists of 43 users (7%). Finally, we define experimenters as those who tried CoScripter without becoming active users, of which we have 504 users (84%). The logs suggest that individual users are automating frequent tasks, with 23 scripts run 10 or more times by single users at a moderate interval (ranging from every day to twice per month).

Collaborating over Scripts

One of the goals of CoScripter is to support sharing of how-to knowledge. The logs imply that sharing is relatively common: 24% of 307 user-created scripts were run by two or more different users, and 5% were run by six or more users. People often run scripts created by others: 465 (78%) of the user population ran scripts they did not create, running 2.3 scripts created by others on average. There is also evidence that users are sharing knowledge of infrequent processes: we found 16 scripts that automate known business processes within our company (e.g., updating emergency contact info), that were run once or twice each by more than ten different users.

In addition to the ability to run and edit others' scripts, CoScripter supports four other collaborative features: editing others' scripts, end-user rating of scripts, tagging of scripts, and free-form comments added to scripts. We found surprisingly little use of these collaborative features: fewer than 10% of the scripts were edited by others, rated, tagged, or commented on. Further research is needed to determine whether and how these features can be made more valuable in a business context.

Email Survey of Lapsed Users

To learn why employees stopped using CoScripter, we sent a short email survey to all the experimenters and past users, noting their lack of recent use and asking for reasons that CoScripter might not have met their needs. Thirty people replied, and 23 gave one or more reasons related to CoScripter. Of the topics mentioned, ten people described reliability problems where CoScripter did not work consistently, or did not handle particular web page features (e.g., popup windows and image buttons); five people said their tasks required advanced features not supported in CoScripter (most commonly requested were parameters, iteration, and branching); three people reported problems coordinating mixed initiative scripts, where the user and CoScripter alternate actions; and two had privacy concerns (i.e., they did not like scripts being public by default). Finally, seven people reported that their jobs did not involve enough tasks suitable for automation using CoScripter.

INTERVIEWS WITH COSCRIPTER USERS

As part of Study 2, we explored the actual usage of CoScripter by conducting a set of interviews with users who had made CoScripter part of their work practices.

Participants

Based on usage log analysis, we chose people who had used one or more scripts at least 30 times. We also selected a few people who exhibited interesting behavior (e.g., editing other peoples' scripts or sharing a script with others). We contacted 14 people who met these usage criteria; 8 agreed to an interview.

Seven interviewees were active CoScripter users, one had used the tool for five months and stopped 3 1/2 months before the interview. Participants had used CoScripter for an average of roughly 4 months (minimum of 1, maximum of 9). They had discovered the tool either via email from a coworker or on an internal website promoting experimental tools for early adopters. Seven participants were male, and they worked in 8 sites across 4 countries, with an average of 10 years tenure at the company. We interviewed 4 managers, 1 communications manager, 1 IT specialist, 1 administrative services representative, and 1 technical assistant to a manager / software engineer. Overall, our participants were technology savvy, and five of them had created a macro or scripts before CoScripter. However, only two of them claimed software development expertise.

Method

We conducted all but one of our interviews over the phone, due to geographically dispersed participants, using NetMeeting to view the participant's Firefox browser. Interviews lasted between 30 and 60 minutes and were audio-recorded with participants' permission in all cases but one.

We conducted a semi-structured interview based on questions that were designed to gather data about how participants used CoScripter, why they used it, and what problems they had using the tool. In addition to the predetermined questions, we probed additional topics that arose during the interview, such as script privacy and usability. We also asked participants to run and edit their scripts so we could see how they interacted with CoScripter.

Results

Study 1 established user needs for automating frequent or hard-to-remember tasks, and sharing how-to knowledge. Study 2 explored how well CoScripter meets these user needs and areas where it falls short.

Automating Frequent or Hard-to-Remember Tasks

Four participants described CoScripter as very useful for automating one or more frequent, routine tasks. Each person had different tasks to automate, highlighting various benefits of CoScripter as an automation tool. Table 1 lists the most frequent routine tasks that were automated.

Those four subjects described instances where CoScripter saved them time and reduced their effort. For example, CR1 said, "Two benefits: one, save me time — click on a button and it happens — two, I wouldn't have to worry about remembering what the address is of the messaging system here." PV1 also appreciated reduced effort to check voicemail inboxes, "I set up [CoScripter] to with one click get onto the message center." AM1 used his service pack registration script for a similar reason, saying it was "really attractive not to have to [enter the details] for every single service pack I had to register." JL1 runs his script many times during non-business hours. He would not be able to do this without some script (unless, as he said, "I didn't want to sleep").

These participants, none of whom have significant software development experience, also demonstrate an important benefit of CoScripter: it lowered the barrier for automation, since it required no programming skills.

Though most participants interacted with CoScripter via the sidebar or the wiki, JL1 and PV1 invoked CoScripter in unexpected ways. JL1 used the Windows Task Scheduler to automatically run his script periodically in the background. Thus, after creating his script, JL1 had no contact with the CoScripter user interface. PV1 created a bookmark to automatically run each of his two scripts, and added them to his Firefox Bookmarks Toolbar. To run the scripts, he simply clicked on the bookmarks — within a few seconds he could check both voicemail inboxes.

In addition to automating frequent tasks, CoScripter can act as a memory aid for hard-to-remember tasks. For example, DG1 created scripts for two new processes he had to do for his job. Both scripts used an online tool to create and send reports related to customer care. Creating the report involved a number of complicated steps, and CoScripter made them easier to remember:

> [CoScripter] meant I didn't have to remember each step. There were probably six or seven distinct steps when you have to choose drop-downs or check-boxes. It meant I didn't have to remember what to do on each step. Another benefit is I had to choose eight different check-boxes from a list of about forty. To always scan the list and find those eight that I need was a big pain, whereas [CoScripter] finds them in a second. It was very useful for that.

After using CoScripter to execute these scripts repeatedly using the step-by-step play mode (at least 28 times for one script, 9 times for the other) for five months, DG1 stopped using them. He explained: "It got to the point that I memorized the script, so I stopped using it." This highlights an interesting use case for CoScripter: helping a user memorize a process that they want to eventually perform on their own.

Participant LH1 found CoScripter useful in supporting hard-to-remember tasks, by searching the wiki and finding scripts others had already recorded: "I found the voicemail one, it's really useful because I received a long email instruction how to check my voicemail. It was too long so I didn't read it and after a while I had several voicemails and then I found the [CoScripter] script. It's really useful."

Automation Limitations and Issues. Despite generally positive feedback, participants cited two main issues that limited their use of CoScripter as an automation tool, both of which were named by lapsed users who were surveyed by email:

reliability problems and a need for more advanced features.

Four out of eight participants noted experiencing problems with the reliability and robustness of CoScripter's automation capability. All of these participants reported problems related to CoScripter misinterpreting instructions. Misinterpretation was a result of CoScripter's original human-readable scripting system [5], which did not enforce any particular syntax, but instead did a best-effort interpretation of arbitrary text. For example, one user reported running a script that incorrectly clicked the wrong links and navigated to unpredictable pages (e.g., an instruction to "click the Health link," when interpreted on a page that does not have a link labeled "Health," would instead click a random link on the page). Another user reported that his script could not find a textbox named in one of the instructions and paused execution. Misinterpretation problems were exacerbated by the dynamic and unpredictable nature of the web, where links and interface elements could be changed or removed.

Five users reported wanting more advanced programming language features, to properly automate their tasks. In particular, participants expressed a need for automatic script restart after mixed-initiative user input, iteration, script debugging help, and conditionals.

Sharing How-To Knowledge

Our interviews revealed some of the different ways CoScripter has been used to share how-to knowledge. These included teaching tasks to others, promoting information sources and teaching people how to access them, and learning how to use CoScripter itself by examining or repurposing other peoples' scripts.

Participants used CoScripter to teach other people how to complete tasks, but in very different ways. The first person, LH1, recorded scripts to teach her manager how to do several tasks (e.g., creating a blog). LH1 then emailed her manager a link to the CoScripter script, and the manager used CoScripter to complete the task. The second person, DG1, managed support representatives and frequently sent them how-to information about various topics, from generating reports online to using an online vacation planner. DG1 used CoScripter to record how to do the tasks, and then copied and pasted CoScripter's textual step-by-step instructions into emails to his colleagues:

> When I want to give people instructions on how to do something on the Internet, it was great for doing that so I didn't have to write them out. I found it very useful for that, creating clear instructions on what to do. I sent the text, but I never sent the link so that they could just run it. I never actually sent a script to someone because I didn't know anyone who used the tool. I could have asked people to install it, but I never did. I tend to work with others who are afraid of new technology. I used that a number of times. For that it did exactly what I needed.

DG1's use of CoScripter highlights a benefit of its human-readable approach to scripting. The scripts recorded by CoScripter are clearly readable, since DG1's colleagues were able to use them as textual instructions.

Another participant, MW1, a communications manager, used CoScripter to promote information that is relevant to employees in a large department and to teach them how to access that information. He created one script for adding a news feed to employees' custom intranet profile and a second script to make that feed accessible from their intranet homepage. He promoted these scripts to 7000 employees by posting a notice on a departmental homepage and he plans to send links to the scripts in an upcoming email. Before using CoScripter, MW1 said he did not have a good method for sharing this how-to information with a wide audience. With such a large audience, however, correctly generalizing the script was a challenge; MW1 said he spent two to three hours getting the scripts to work for others. Still, MW1 was so pleased that he evangelized the tool to one of his colleagues, who has since used CoScripter to share similar how-to knowledge with a department of 5000 employees.

Participants used CoScripter for a third sharing purpose: three people talked about using other peoples' scripts to learn how to create their own scripts. For example, PV1 told us that his first bug-free script was created by duplicating and then editing someone else's script.

Sharing Limitations and Issues. Though some participants found CoScripter a valuable tool for sharing, these participants noted limitations of critical mass and others raised issues about the sharing model, generalizability, and privacy, which were barriers to sharing.

Sharing has been limited by the narrow user base (so far) within the enterprise. As CR1 said, critical mass affected his sharing ("I haven't shared [this script] with anyone else. But there are other people I would share it with if they were CoScripter users."), and his learning ("I think I would get a lot more value out of it if other people were using it. More sharing, more ideas. Yet most of the people I work with are not early adopters. These people wouldn't recognize the difference between Firefox and Internet Explorer."). DG1, who sent CoScripter-generated instructions to his coworkers via email, could have more easily shared the scripts if his coworkers had been CoScripter users.

We also saw problems when users misunderstood the wiki-style sharing model. For example, PV1 edited and modified another person's script to use a different online tool. He did not realize that his edits would affect the other person's script until later: "I started out by editing someone else's script and messing them up. So I had to modify them so they were back to what they were before they were messed up, and then I made copies." A second participant, PC1, had deleted all the contents of a script and did not realize this until the script was discussed in the interview: "That was an accident. I didn't know that [I deleted it]. When I look at those scripts, I don't realize that they are public and that I can blow them away. They come up [on the sidebar] and they look like examples."

Easy and effective ways to generalize scripts so that many people can use them is essential to sharing. Though users are able to generalize CoScripter scripts, participants told us it is not yet an easy process. For example, MW1 created a script for 7000 people, but spent a few hours getting it to work correctly for others. Also, while CoScripter's personal database feature allows users to write scripts that substitute user-specified values at runtime, not all processes can be generalized using this mechanism. For example, PV1 copied another person's script for logging into voicemail in one country and modified it to login to voicemail in his country, since each country uses a different web application.

Finally, sharing was further limited when participants' were concerned about revealing private information. Three participants created private scripts to protect private information they had embedded in the scripts. AC1 said: "There is some private information here — my team liaison's telephone number, things like that. I don't know, I just felt like making it private. It's not really private, but I just didn't feel like making it public." Others used the personal database variables to store private information and made their scripts public. This privacy mechanism was important to them: "Without the personal variables, I would not be able to use the product [CoScripter]. I have all this confidential information like PIN numbers. It wouldn't be very wise to put them in the scripts." However, one participant was wary of the personal variables, "The issue I have with that is that I don't know where that is stored ... If I knew the data was encrypted, yeah."

Summary

These findings show that, while not perfect, CoScripter is beginning to overcome some of the barriers to sharing procedural knowledge uncovered in Study 1. First, CoScripter provides a single public repository of procedural knowledge that our interviewees used (e.g., several participants used scripts created by other people). Second, CoScripter eliminates the tedious task of writing instructions (e.g., DG1 used it to create textual instructions for non-CoScripter users). Third, CoScripter provides mechanisms to generalize instructions to a broad audience so that experts can record their knowledge once for use by many learners (e.g., MW1 generalized his scripts to work for 7000 employees).

GENERAL ISSUES AND FUTURE WORK

By helping users automate and share how-to knowledge in a company, CoScripter is a good starting point for supporting the issues uncovered in Study 1. User feedback, however, highlights several opportunities for improvement. While some of the feedback pointed out usability flaws in our particular implementation, a significant number of comments addressed more general issues related to PBD-based knowledge sharing systems. Participants raised two issues that highlight general automation challenges — reliability challenges and the need for advanced features — and four collaboration issues that will need to be addressed by any knowledge sharing system — the sharing model, script generalization, privacy, and critical mass.

One of the most common complaints concerned the need for improved reliability and robustness of the system's automation capability. These errors affected both users relying on the system to automate repetitive tasks, and those relying on the system to teach them how to complete infrequent tasks. Without correct and consistent execution, users fail to gain trust in the system and adoption is limited.

Users also reported wanting more advanced programming language features, such as iteration, conditionals, and script debugging help, to properly automate their tasks. These requests illustrate a tradeoff between simplicity — allowing novice users to learn the system easily — and a more complex set of features to support the needs of advanced users. For example, running a script that has iterations or conditionals might be akin to using a debugger, which requires significant programming expertise. A challenge for CoScripter or any similar system will be to support the needs of advanced users while enabling simple script creation for the broader user base.

Our studies also raise several collaboration issues that must be addressed by any PBD-based knowledge sharing tool: the sharing model, privacy, script generalization, and critical mass. The wiki-style sharing model was confusing to some users. Users should be able to easily tell who can see a script and who will be affected by their edits, especially given the common base of processes being performed by many employees. A more understandable sharing model could also help address privacy concerns, as would a more fine-grained access control mechanism that enabled users to share scripts with an explicit list of authorized persons. Finer-grained privacy controls might encourage more users to share scripts with those who have a business need to view them. For generalization, we learned that personal database variables were a good start, but we are uncertain as to what degree this solution appropriately supports users with no familiarity of variables and other programming concepts. One way to better support generalization and personalization could be to enable users to record different versions of a script for use in different contexts, and automatically redirect potential users to the version targeted for their particular situation. Finally, a small user base limited further use of the system. We hope that solving all the issues above will lower the barriers to adoption and improve our chances of reaching critical mass.

Finally, one important area for future work is to study the use of CoScripter outside the enterprise. While we conducted the studies in a very large organization with diverse employees and we believe their tasks are representative of knowledge workers as a whole, the results we have obtained may not be generalized to users of CoScripter outside the enterprise. CoScripter was made available to the public in August 2007 (see http://coscripter.research.ibm.com/), and more research on its use is needed to examine automation and sharing practices in this larger setting.

CONCLUSION

In summary, we have presented the CoScripter system, a platform for capturing, sharing, and automating how-to knowledge for web-based tasks. We have described CoScripter's user interface and some of its key technical features. The empirical studies we have presented show that CoScripter has made it easier for people to share how-to knowledge inside the enterprise.

ACKNOWLEDGEMENTS
We thank Greg Little for the initial inspiration for the CoScripter project and Jeffrey Nichols for invaluable discussions and architectural design.

REFERENCES

1. M. Bolin, M. Webber, P. Rha, T. Wilson, and R. C. Miller. Automation and customization of rendered web pages. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 163–172, New York, NY, USA, 2005. ACM.
2. A. Cypher. Eager: programming repetitive tasks by example. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 33–39, New York, NY, USA, 1991. ACM.
3. C. Kelleher and R. Pausch. Stencils-based tutorials: design and evaluation. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '05)*, pages 541–550, Portland, Oregon, USA, 2005.
4. K. J. Lee. What goes around comes around: an analysis of del.icio.us as social space. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 191–194, New York, NY, USA, 2006. ACM.
5. G. Little, T. A. Lau, A. Cypher, J. Lin, E. M. Haber, and E. Kandogan. Koala: capture, share, automate, personalize business processes on the web. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 943–946, New York, NY, USA, 2007. ACM.
6. G. Little and R. C. Miller. Translating keyword commands into executable code. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 135–144, New York, NY, USA, 2006. ACM.
7. D. R. Millen, J. Feinberg, and B. Kerr. Dogear: Social bookmarking in the enterprise. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 111–120, New York, NY, USA, 2006. ACM.