RJ10443 (A0904-006) April 14, 2009
Computer Science

IBM Research Report

Interpreting Written How-To Instructions

Tessa Lau, Clemens Drews, Jeffrey Nichols
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099
USA

Interpreting Written How-To Instructions

Tessa Lau, Clemens Drews, and Jeffrey Nichols
IBM Almaden Research Center
650 Harry Rd
San Jose, CA 95120 USA
{tessalau,cdrews,jwnichols}@us.ibm.com

Abstract

Written instructions are a common way of teaching people how to accomplish tasks on the web. However, studies have shown that written instructions are difficult to follow, even for experienced users. A system that understands human-written instructions could guide users through the process of following the directions, improving completion rates and enhancing the user experience. While general natural language understanding is extremely difficult, we believe that in the limited domain of how-to instructions it should be possible to understand enough to provide guided help in a mixed-initiative environment. Based on a qualitative analysis of instructions gathered for 43 web-based tasks, we have formalized the problem of understanding and interpreting how-to instructions. We compare three different approaches to interpreting instructions: a keyword-based interpreter, a grammar-based interpreter, and an interpreter based on machine learning and information extraction. Our empirical results demonstrate the feasibility of automated how-to instruction understanding.

1 Introduction

Written how-to documentation is everywhere, from the instruction manuals that accompany software packages to FAQ documents available online. Like a cooking recipe, these instructions often explain in step-by-step order how to accomplish a task. Today, many instructions are distributed in electronic form. For example, in the enterprise, employees are routinely sent instructions via email on how to update their emergency contact information, classify their IT assets, transfer employees into a different division, or search for job openings. In the consumer web space, users are sharing instructions via forums and newsgroups to help each other accomplish tasks in various applications.

Unfortunately, studies have found that people have difficulty following written instructions [Lau *et al.*, 2004; Prabaker *et al.*, 2006]. Some people report difficulty mapping from the written instruction to the target on the screen, and having to systematically search the on-screen interface for the named button. Others lose track of where they are in the instructions, resulting in following instructions out of order, or missing steps entirely. Such problems can be exacerbated if users are distracted in the middle of performing tasks and must return to them later having lost most of their context.

While guided help systems such as Coachmarks [coachmarks, 1996], Stencils [Kelleher and Pausch, 2005], and CoScripter [Leshed *et al.*, 2008] address this problem, these systems rely on documentation that has been pre-authored for use with their system. They cannot take advantage of the vast array of existing how-to literature already available on the internet that has been written for humans by humans. If we could automatically parse and interpret these instructions, we could enable a new class of guided help systems that recognize instructions in context (e.g., in email, or on a web page) and immediately guide users through performing the task.

At a high level, the problem of understanding hand-written instructions seems to amount to understanding natural language. However, we believe that most instructional material uses a more structured subset of language that may be amenable to automated processing, particularly if we limit the domain of interest to a specific platform. In the web domain, which is the focus of this paper, most documentation uses a common set of terminology to describe operations, using verbs such as "click" and "type" and objects such as "link" and "textbox".

Specifically, this paper makes the following contributions:

- A formalization of the problem of understanding hand-written how-to instructions;
- A qualitative analysis and manually-labeled corpus of hand-written instructions collected from the internet; and
- A comparison of three algorithms for interpreting written instructions automatically.

We begin with a brief overview of related work. We then introduce the data we have collected and present an initial qualitative analysis of the data. Next, we introduce the problem of understanding how-to instructions, and describe our three approaches to the problem. We then report the results of a comparative study of the performance of our three approaches. Finally, we conclude with a discussion of future work.

2 Related work

Previous studies have illustrated some of the limitations of how-to documentation, and proposed various approaches to address these limitations. The Sheepdog study [Lau *et al.*, 2004] found that people have difficulty following directions online, including difficulty mapping from the written text to the on-screen widget and difficulty following instructions in the correct sequence. To address these problems, DocWizards [Prabaker *et al.*, 2006] presents users with a guided walkthrough of demonstrationally-authored documentation, highlighting steps as necessary in the Eclipse interface to visually lead users through the instructions. CoScripter [Leshed *et al.*, 2008; Little *et al.*, 2007] provides a similar guided walkthrough interface for web tasks. All of these systems assume that instructions have previously been recorded in a proprietary format; progress on automatically interpreting hand-written instructions could enable each of these systems to work with a much wider library of existing instructions.

Automatically parsing textual instructions can be viewed as an information extraction problem. Previous research has successfully parsed hand-written requests in email, e.g. VIO [Tomasic *et al.*, 2006], which automatically extracts commands to fill out a form to effect a change on a web site. While VIO is designed to work with a pre-specified set of possible forms, our system addresses the more difficult challenge of working with arbitrary web pages, where the set of possible actions is not known in advance. The Mr. Web system [Lockerd *et al.*, 2003] found that users utilize a constrained form of English when communicating with a webmaster over email. We believe a similar effect occurs in how-to documentation, where a constrained form of English is used to describe interactions with web applications.

Little and Miller's Keyword Commands system [Little and Miller, 2006] was one of the first to interpret user-generated instructions as actions on web pages; their algorithm inspired the keyword-based interpreter examined in this paper. While keyword-based approaches excel at interpreting underspecified commands, they may produce many false positives when given instructions that do not contain any actionable commands (which occur frequently in our datasets).

The PLOW system [Allen *et al.*, 2007] also uses natural language understanding to interpret human-generated speech in the context of a web task completion system. PLOW's interpretation is performed within the context of a supervised interaction that includes user demonstrations, which are not available to our system. We anticipate leveraging some of the natural language algorithms used by PLOW in the future.

3 Datasets

To evaluate our algorithms, we collected a corpus of instructions written by people describing how to accomplish tasks on the web. We crowdsourced the problem by creating a set of tasks on Amazon's Mechanical Turk marketplace, asking volunteers to write how-to instructional text in exchange for USD$0.25 per set of instructions. We found it necessary to iterate several times on our task description in order to elicit the desired output, as others have found [Kittur *et al.*, 2008].

	scenarios	**# steps**	**# segments**
TRAIN	24	158	274
TEST	19	142	289

Table 1: Summary of datasets

We collected two sets of instructions using Mechanical Turk and formed two datasets, TRAIN and TEST (Table 1). TRAIN contains 24 scenarios, with a total of 158 steps. TEST contains 19 scenarios, with 142 steps.

Our workers wrote instructions for a wide range of websites and tasks. Their scenarios include how to create a gmail account, how to buy a fan on Amazon.com, and how to create a level 70 WarCraft character using a web-based character-editing tool. Most of the scenarios described tasks on publicly-accessible websites, though several required authenticated access to perform, such as logging in to an online banking site, or accessing a university's intranet. In order to evaluate the correctness of an interpretation of an instruction, we needed to be able to access the specified website. The TEST$'$ dataset contains the subset of TEST that operate on websites that are accessible to the general public. TEST$'$ contains 166 instructions.

4 Qualitative analysis

Interpreting human-written instructions is challenging because people do not always use a consistent syntax when describing commands to perform on web pages. The more variation (noise) there is in instruction text, the harder it will be for automated systems to parse those instructions correctly. A qualitative analysis of our dataset revealed a high level of noise that makes automated interpretation challenging.

First, there were a large number of *spelling and grammar mistakes* in the collection. For example, this step was part of a set of instructions on how to use Google search:

I- 1 *Click "I'm Felling Luck' to go direclty to a website that involves your description.*

The three misspellings and pair of mismatched quotes are representative of the mistakes we found in the larger corpus. For the results described in this paper, we have left the data intact and not done any spelling or grammar correction on the data. Any excerpts quoted in this paper preserve the original mistakes made by the authors.

Second, there was a large *variation in complexity* of each individual step description. Some participants described one command per step, such as the example above. Others grouped together many commands into a single step:

I- 2 *After you have filled in the information on this page, click the bottom button that says "Next." Now a form that asks for more information will pop up. [...] Fill in the address if the form doesn't have one already. [...]*

This step includes multiple instructions as well as commentary about those instructions. It also includes *verification steps*, instructions that help the reader keep in sync by advising the reader what to look for or what to expect (e.g., "now a form will pop up"). While our current algorithms do not do anything with this type of instruction, we envision future systems being able to understand these verification steps to ensure the user is on the right path.

Many instructions also contained *conditionals*, such as steps that instructed users to check a box if they wanted a desired effect (e.g., remembering your username on future visits).

We hoped that writers would use a simple verb-object grammar to describe commands on web pages. However, we often found *out-of-order instructions* such as the following:

I- 3 *Just underneath where is says "Step 1" there is a button saying "browse", click on it.*

Finally, several steps contained *implicit language*:

I- 4 *You can use the "Preview" button to see how it looks.*

People reading this step may or may not choose to click the button, depending on their needs; a guided help system would be most useful if it pointed out the target button and left the decision up to the user whether to invoke it.

The variety with which instructions were expressed illustrate the difficulty of automatically understanding such instructions. However, we would like to point out that complete accuracy is not required for a guided-help system to be useful. Even if the system can only provide automated assistance for some fraction of the steps, having such a system would still be better than not having any automated assistance at all.

4.1 Labeling web commands

Based on our corpus, we identified a number of "web commands" that describe operations on web pages (Table 2). These commands (go to, click, enter, select, check, or no action) collectively cover the vast majority of actions that can be taken on web pages. Clicks were the most commonly-occurring action, followed by entering text. A large fraction of segments did not describe a command; these represent the commentary/advice segments discussed earlier.

In addition, we observed a few different methods for identifying the interaction target on each page. While people occasionally used color or location to identify targets, the most common descriptor seemed to be a textual string representing the target's label or caption followed by the type of the target (e.g., "the Register link", "the Preview button"). Thus, we assume that the combination of action, value, target label, and target type is usually sufficient to completely specify the command to perform.

			# occurrences	
Action	**Value**	**Target type**	**TRAIN**	**TEST'**
Go to	URL	-	17	12
Click	-	link	88	52
Enter	text	textbox	19	18
Select	listitem	listbox	1	0
Turn on/off	-	checkbox/ radiobutton	3	0
No action			146	84

Table 2: Supported web commands and frequency of occurrence

5 Understanding how-to instructions

At a high level, we have formalized the problem of understanding how-to instructions into two parts: segmentation and interpretation. Given a document containing written instructions, we first *segment* the document into individual segments $S_1, S_2, ...S_n$ such that each segment contains at most one command. For a first pass, we can leverage the structure of the document; many how-to documents use bulleted or numbered lists to organize instructions into a sequence of steps.

However, within each step, the text may describe one or more commands to perform. For example, the text "enter your name and click go" contains the two individual commands "enter your name" and "click go". Each command has a different action (enter, click) and is applied to a different target in the application. Segmentation splits each step into the individual commands contained within the step. A solution to the segmentation problem would form the basis for another paper; here, we assume our input has already been segmented and leave the discovery of accurate segmentation algorithms to future work.

Given a segment and the context in which that instruction is to be followed (e.g., the web page to which the instruction applies), we want to *interpret* that instruction relative to the web page and formulate a command. The command should contain enough information for a machine to programmatically perform this command on the given web page.

We represent a command as a tuple $C = (A, V, T)$ containing:

- **A - Action**: the type of action to perform
- **V - Value**: the textual parameter to an action (e.g., URL to visit, text to enter in a field)
- **T - Target**: the target on which to act

As discussed above, in written instructions, targets are typically described using a textual descriptor and a type. We use these two properties to specify the target T:

- $\mathbf{T}_L$ **- Target label**: a textual description of a target, which may be its label, caption, or other descriptive text
- $\mathbf{T}_T$ **- Target type**: the type of a target (e.g., link, textbox, listbox)

In summary, an interpreter I is a function that maps from an instruction segment S and a web page context X to either a command C or null, if the segment does not contain a command: $I(S, X) => \{C, \emptyset\}$.

The output of the interpreter will be input to a guided help system that walks the user step-by-step through written instructions. For each segment, if no command is specified, the help system will display the segment to the user. The user can read the segment and optionally take action based on viewing that segment. If the segment contains a command, the help system should highlight the command's target on the current web page and place the focus on the target. For example, if the command is to enter your username into the "login" textbox, the help system could highlight the login textbox on the web page and set the focus to it, so that the user could start typing immediately in the field. After the user has completed the instruction, the guided help system would then move on to the next instructional segment.

6 Interpretation

We have implemented three different approaches to interpreting how-to instructions. The sloppy interpreter (KW) is a keyword-based algorithm that uses the web page context and the words in the instruction to synthesize commands for that web page. The grammar-based interpreter (GR) uses a grammar-based parser to extract the action, value, target label, and target type from the instruction. Both KW and GR have been used in the CoScripter system; KW in the original version [Little *et al.*, 2007] and GR in the current version. The machine learning-based interpreter (ML) uses trained document classifiers and information extractors to predict the action and extract the value, target label, and target type from each instruction.

6.1 Keyword-based Interpreter

Though the KW algorithm has been described in detail previously [Little *et al.*, 2007], we briefly outline it here.

KW interprets a segment of text in the context of a web page. It 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. KW then assigns a score to each element by comparing the bag of words in the input instruction 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 `click`; if the element is a textbox, the action will be `enter`. 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 KW algorithm causes it to always predict some action, regardless of the input given.

6.2 Grammar-based Interpreter

GR uses an $LL(1)$ parser that parses imperative English instructions that are applicable to the web domain. This limits the flexibility of the instructions that can be recognized, but has the advantage that it is simple, fast, and precise. If an instruction can be parsed by the grammar, we can be fairly certain that it is a well formed statement. Statements accepted by the grammar include: *go to www.gmcard.com*; and *click on the "Save and Submit" button*.

The verb at the beginning of each statement describes the action. Optionally a value can be specified, followed by a description of the target label and an optional target type. Figure 1 shows an excerpt of the BNF grammar recognized by this parser, for `click` actions. The result of a successful parse of a human readable instruction is the extraction of the tuple (A, V, T_L, T_T) from the parsed instruction.

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

Figure 1: Excerpt from GR parser's grammar

6.3 Machine learning-based interpreter

Interpreting written instructions can be formulated as a machine learning problem. By using state-of-the-art information extraction techniques, we hypothesized that we could build a parser that is more tolerant of variations in instruction wording (one of the weaknesses of the grammar-based parser) yet achieves a higher precision than the keyword-based sloppy parser.

Using the TRAIN dataset, we trained a multi-class document classifier on the set of segments. For each segment, this classifier predicts a class label representing the action type of the segment (goto, click, enter, select, check, or no action). We used the OneVsAll learner from the MinorThird package [Cohen, 2004], which builds N different classifiers, each of which discriminates between instances of one class and all the remaining classes. The results of the N individual classifiers are combined to form the final prediction. We used the default MinorThird settings, which specify Maximum Entropy as the inner classifier, and used the default features.

Once each segment has been classified with an action, we trained individual extractors to identify the crucial parts of each segment necessary to execute it. For goto actions, we extract the URL. For clicks and checks, we extract a target type and a target label – enough to specify which target to click on. For enter actions, we extract the target type, target label, and (where possible) the string value to enter into the text field. For select actions, we extract the target type, target label, and the name of the item to select from the dropdown.

For each of these 11 extraction tasks, we trained a separate annotator using the MinorThird toolkit. Each annotator was trained on the labelled data in the TRAIN dataset. For example, the goto-value annotator was trained to extract the values from the 17 goto segments in TRAIN. We used the default MinorThird settings to create the annotators, which employ a VPHMM learner [Collins, 2002]. We also enabled the annotator to employ part-of-speech tags in its feature set.

7 Evaluation

We have evaluated the performance of the three interpreters in two phases. Our first evaluation is based on the observation that approximately half of the instructions in our corpus did not include a specific command, but instead included commentary such as general advice on passwords. If a guided help system were to suggest taking action on every one of these non-command steps, it would predict many extraneous commands, causing users to lose faith in the system. Therefore our first experiment characterizes the ability of each of our interpreters to correctly recognize non-command segments. In our second experiment, we consider only the executable commands in the corpus and evaluate the performance of each of our algorithms at correctly interpreting each such instruction.

KW interpreter		
Actual \ Predicted	Command	No command
Command	**82**	0
No command	84	**0**

GR interpreter		
Actual \ Predicted	Command	No command
Command	**6**	76
No command	0	**84**

ML interpreter		
Actual \ Predicted	Command	No command
Command	**74**	8
No command	22	**62**

Table 3: Performance results for recognizing command instructions

7.1 Recognizing command instructions

Our first experiment evaluates the accuracy of the three interpreters at correctly differentiating between instructions that contain commands and those that do not. Instructions not containing direct commands will be presented to the user as informational messages during execution of the procedure.

To evaluate the interpreters, we ran each interpreter on each segment in the TEST′ corpus and noted whether it predicted an action (e.g., click, enter, goto) or no action. Results are summarized in Table 3. As expected, the KW interpreter correctly classifies only 49% of the segments (the ones that actually contain a command) because it assumes every instruction contains a command. The GR interpreter recognizes few commands because only a small fraction of the instructions in the corpus conform to its grammar; this result is not surprising given the large variation in the instruction corpus. The ML interpreter performs the best, achieving 82% accuracy on recognizing commands. An analysis of the misclassified instructions reveals that the actual performance of a guided help system would be even better than this percentage implies, because the errors are inconsequential to the operation of the system.

For example, four of the 30 instructions contained implicit references to clickable items but no explicit instruction to click them (e.g., instruction I-4 above). ML incorrectly predicted taking action in this case; however, this would be reasonable behavior for a guided help system.

Another 13 of the 30 instructions are misclassified as enter actions even though the instruction does not provide enough detail to complete the command. For example, one instruction said to `fill out the rest of your information.` ML misclassified these as enter actions even though we expect them to be presented to the user as general advice instead. Note also that these instructions do not provide enough detail to identify the target label or type. In these cases, a guided help system could simply present the instruction as if it did not contain a command, resulting in the correct behavior from a user perspective.

	Act.	**Val.**	**Label**	**Type**	**Overall**
KW	93%	90%	65%	80%	59%
GR	9%	80%	23%	25%	3%
ML	88%	81%	33%	30%	10%

Table 4: Interpretation accuracy on 69 command instructions

7.2 Interpreting command instructions

In addition to recognizing whether or not an instruction contains an command, a guided help system must correctly identify which action to take by extracting the components of the command. Our second experiment evaluates the ability of each of the three interpreters to correctly predict which action to take given the instruction and the current web page.

A total of 82 instructions contained commands. Twelve of these required the user to express an opinion about the preferred link from a set of links on a page (e.g., a search result that looked promising). Performing this type of action is outside the scope of the guided help system that we hope to create. We removed them from consideration; 69 instructions remained.

In order to correctly interpret a command instruction, the interpreter must be able to identify the action type (e.g. click), the target label (e.g., "accept"), and if necessary, the value (e.g., the URL to go to). We ran each of the three interpreters on each instruction segment in the TEST′ corpus. Each interpreter produced an action type, value, target label, and target type for each segment. We then compared these against the true values as defined by a human (one of the authors) manually reading the instruction and interpreting it on the web page. If all of the components match, then we deem the interpreted command to be correct.

Many of the `enter` instructions required human judgement to complete (e.g., entering a username or a search term). In these cases, identifying the target is sufficient for a guided help system; the exact text to enter is not critical. Therefore, for these human-judgement commands, we deemed the interpretation correct as long as the predicted action, target label, and target type are correct.

Table 4 summarizes the results. The Action, Value, Label, and Type columns show each algorithm's performance at predicting each command component. The Overall column shows the accuracy of predicting all components correctly. The KW interpreter is best with 59% accuracy, and GR and ML perform poorly with 3% and 10% accuracy, respectively. Given the noisiness in the corpus, it is not surprising that our interpreters fail to correctly interpret many of the instructions. KW's design goal of handling noisy instructions with no particular syntax explains why it performs relatively well at this task.

The errors made by the KW interpreter showcase the difficulty of the problem faced by automated instruction interpreters. Three of the mistakes were due to the instruction not containing the same label as the page (e.g., `click on the blue words` "`go to my albums`" when the page has a "click here to go to your album" link). Eight mistakes were due to command types that our system does not recognize, such as clicking in a textbox to give it focus.

Unlike KW, the ML interpreter can only predict target labels and types that actually appear in the instructions. Users rarely provided a complete and correct label for targets (e.g., instruction I-1 above). Also, the brevity of some instructions (e.g., `press go`) made it difficult for ML's information extraction algorithms to correctly extract target labels. It is likely that domain-specific extraction algorithms will be required for a learning-based approach to achieve higher accuracy.

8 Conclusion

In summary, we have presented an analysis of and solutions to the problem of automatically interpreting human-authored how-to instructions. We reported on the collection of a dataset of written instructions for 43 web tasks, and presented a qualitative analysis of these instructions. We presented and compared three different approaches to instruction interpretation, showing that the machine learning-based algorithm outperforms the keyword-based and grammar-based algorithms at differentiating between executable instructions and advice, with 82% accuracy. The keyword-based algorithm performs better at identifying the command described in executable instructions, with 59% overall accuracy. Our results indicate that automatic how-to instruction interpretation shows promise, and may soon enable the creation of systems that assist users in following hand-written instructions.

Many directions remain for future work. We would like to implement our interpreters in a mixed-initiative guided help system and study user reactions to such a system, to verify that the current level of accuracy is acceptable for general use. We plan to investigate whether the ML approach can be redeemed through the use of domain-specific information extraction techniques, perhaps incorporating features from the more-successful keyword-based parser. Finally, we will validate our results on a larger corpus of scenarios written by a wider variety of technical users.

References

[Allen *et al.*, 2007] James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift, and William Taysom. PLOW: A Collaborative Task Learning Agent. In *AAAI 07: Proceedings of the Twenty-Second Conference on Artificial Intelligence*, 2007.

[coachmarks, 1996] Designing Coachmarks. `http://developer.apple.com/documentation/mac/AppleGuide/AppleGuide-24.html`, 1996.

[Cohen, 2004] William W. Cohen. Minorthird: Methods for Identifying Names and Ontological Relations in Text using Heuristics for Inducing Regularities from Data. `http://minorthird.sourceforge.net`, 2004.

[Collins, 2002] Michael Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *EMNLP '02: Proceedings of the 2002 conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

[Kelleher and Pausch, 2005] Caitlin Kelleher and Randy Pausch. Stencils-based tutorials: design and evaluation. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 541–550, New York, NY, USA, 2005. ACM.

[Kittur *et al.*, 2008] Aniket Kittur, Ed H. Chi, and Bongwon Suh. Crowdsourcing user studies with Mechanical Turk. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 453–456, New York, NY, USA, 2008. ACM.

[Lau *et al.*, 2004] Tessa Lau, Lawrence Bergman, Vittorio Castelli, and Daniel Oblinger. Sheepdog: learning procedures for technical support. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces*, pages 109–116, New York, NY, USA, 2004. ACM.

[Leshed *et al.*, 2008] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. CoScripter: automating & sharing how-to knowledge in the enterprise. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1719–1728, New York, NY, USA, 2008. ACM.

[Little and Miller, 2006] Greg Little and Robert 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.

[Little *et al.*, 2007] Greg Little, Tessa A. Lau, Allen Cypher, James Lin, Eben M. Haber, and Eser 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.

[Lockerd *et al.*, 2003] Andrea Lockerd, Huy Pham, Taly Sharon, and Ted Selker. Mr.Web: an automated interactive webmaster. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 812–813, New York, NY, USA, 2003. ACM.

[Prabaker *et al.*, 2006] Madhu Prabaker, Lawrence Bergman, and Vittorio Castelli. An evaluation of using programming by demonstration and guided walkthrough techniques for authoring and utilizing documentation. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 241–250, New York, NY, USA, 2006. ACM.

[Tomasic *et al.*, 2006] Anthony Tomasic, John Zimmerman, and Isaac Simmons. Linking messages and form requests. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, pages 78–85, New York, NY, USA, 2006. ACM.