Mining Online Software Tutorials: Challenges and Open Problems

Adam Fourney
University of Waterloo
Waterloo, ON, Canada
afourney@cs.uwaterloo.ca

Michael Terry
University of Waterloo
Waterloo, ON, Canada
mterry@cs.uwaterloo.ca

CHI 2014, April 26 - May 01, 2014, Toronto, ON, Canada.

ACM 978-1-4503-2474-8/14/04…$15.00.
http://dx.doi.org/10.1145/2559206.2578862

Abstract

Web-based software tutorials contain a wealth of information describing software tasks and workflows. There is growing interest in mining these resources for task modeling, automation, machine-guided help, interface search, and other applications. As a first step, past work has shown success in extracting individual commands from textual instructions. In this paper, we ask: How much further do we have to go to more fully interpret or automate a tutorial? We take a bottom-up approach, asking what it would take to: (1) interpret individual steps, (2) follow sequences of steps, and (3) locate procedural content in larger texts.

Author Keywords

Tutorial mining; Natural language processing

ACM Classification Keywords

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

Introduction

The Internet contains a vast and rich repository of tutorials and other procedural information describing how to accomplish a wide range of tasks with almost any publicly available interactive system. For nearly any task, it is likely that there exists online instructional content that will assist users in accomplishing their goals.

Underspecified steps:
"Create a new, rather large document."

Anti-patterns:
"Whatever you do, don't create a hanging indent by pressing the space key to create spaces, or even by tabbing across the page"

Theory or background:
"Unsharp mask works by increasing the contrast between edges in the photos."

Figure 1: Samples of tutorial steps that demonstrate challenges posed by: **underspecified steps** or parameters (top), **anti-patterns** or warnings of what *not* to do (middle), and text that provides **background or theoretical details** that need not be executed by the user (bottom).

Recognizing the wealth of data afforded by these resources, researchers in recent years have turned to the problem of extracting useful data from online tutorials. This past research has explored applications of these data including task modeling [2], software automation [1,10,14], machine-guided help [11], and interface search [5,6]. Beyond this existing research, there are many compelling ways this data could be utilized. For example, a system could infer the time required to perform a tutorial, the target audience of the tutorial (e.g., novice vs. expert), or the amount of creativity or input expected of users. Tutorials could also be used to infer attributes related to the design of the software, such as missing features, or features that frequently lead to breakdowns in use of the software.

The primary challenges in extracting data from tutorials lie in the fact that the information is represented using natural language and, frequently, images and video content, requiring systems that can transform this free-form content into forms that systems can readily reason about. Much of the prior work in this space has focused on extracting information from text-based tutorials, which is also the focus of this paper. This past work has demonstrated [5,10,11,14] that mentions of user interface widgets (tools, menus, etc.) can be detected in instructional material with accuracies of 95-97%. But, the information available in tutorials is much richer and more nuanced than simple lists of widgets or commands. Despite clear progress, greater, and broader, machine understanding of instructional materials remains a significant research challenge.

As an example, consider the following excerpt from a real-world photo manipulation tutorial:

> *"Place it underneath the original text, as if it were a reflection."*

The correct interpretation of this instruction requires: (a) *coreference resolution* [8], to determine to which object the pronoun "it" refers; (b) *spatial reasoning*, to determine approximately where the item is to be placed; and (c) an understanding of the *purpose clause* [4] "as if it were a reflection" to further constrain the final placement. In this case, three challenges arise from a single tutorial sentence. When examining full tutorials, these and other challenges quickly accumulate, compounding the problem (e.g., See Figure 1).

The primary contribution of this paper is to present a roadmap for the research challenges that must be tackled for more complete machine understanding of instructional materials for interactive systems, with a focus on text-based materials. The paper serves to consolidate and organize the failure cases and limitations mentioned in past work, including some of our own papers [5,6]. It also presents challenges we have encountered while working in this space, many of which have not been explicitly identified in past work, but are nonetheless critical to the correct interpretation of written instructions. Throughout the paper, we contextualize each challenge with numerous real-world examples across a range of applications and tutorials, and discuss partial or potential solutions when such mitigations are possible.

Background

Much of the prior work in this research area has focused on extracting commands and parameter values from text-based sources, typically using supervised learning methods.

Motivated by the desire to improve guided help systems, Lau et al. explored possible strategies for extracting operations from handwritten how-to instructions generated by workers on Mechanical Turk [11]. In this work, the authors manually segmented the instructions so that each segment contained an action (e.g., click), a target (e.g., the “OK” button), and any associated values (e.g., parameters). In comparing approaches for extracting data, they found a keyword (template-based) strategy outperformed both a handcrafted grammar and a set of maximum entropy classifiers, achieving accuracies of 93%, 90% and 65% for recovering actions, values, and targets, respectively. However, their techniques assume all text segments describe operations to perform, which the authors found to be true only ~50% of the time.

Lau et al.'s work was later expanded upon [14], with the goal of transforming professionally written test cases into automated software testing procedures. In this later work, support vector machines (SVMs) and conditional random fields (CRFs) were utilized to segment the text into steps and extract actions, targets, and values from the resultant segments. The authors achieved F1-scores consistently over 0.95 for segmentation, and similar scores for each of the aforementioned entity types. Even with these high scores, errors and ambiguity accumulated when interpreting complete operations, and the resultant system correctly interpreted 70% of the steps.

In the realm of online tutorials, which typically contain a lot of “noise” in the data (e.g., ads, site navigation, comments) our previous work [5] explored the possibility of detecting mentions of the software’s user interface elements (e.g., menus, dialogs, tools) referenced in text. In this work, we utilized naive Bayes classifiers with a carefully selected set of features, and achieved an F1-score of 0.87 when processing GIMP photo manipulation tutorials. Comparably, Laput et al. [10] utilized conditional random fields (CRFs) to detect menus, tools and parameters in Photoshop tutorials. Resultant F1-scores of 0.97, 0.98 and 0.36 were achieved for menus, tools and parameters respectively.

In contrast to the work described above, Brasser and Linden [2] strived to automatically extract detailed task models from written scenarios. The authors manually crafted a natural language grammar, which was implemented as a 25-state augmented transition network. Compared to more recent work employing machine learning, the hand-built grammar did not perform particularly well. Tasks were segmented with 63% accuracy, and entities were detected with 48% accuracy.

In this vein, Branavan et al. [1] demonstrated the potential for reinforcement learning approaches for interpreting natural language instructions. Branavan et al.'s technique learns how to interpret instructions by repeatedly testing hypotheses within a virtual machine. This approach has the advantage of being able to interpret some high-level composite actions that lack mention of the specific low-level operations needed to perform those actions in the interface. The authors reported that their method was able to correctly interpret 62% of the high-level actions in their dataset.

"Just Ctrl + click ***that object*** *and* ***it*** *will be selected."* (I)

"Over ***that*** *make a transparent layer."* (G)

Figure 2: Tutorial phrases requiring **coreference resolution** for correct interpretation.

Finally, procedural information can occur in online contexts beyond tutorials. Andrew Ko's Frictionary system [9] extracts software feature/action phrases from Firefox support forums, where users discuss their actions together with problematic software behaviors. Firctionary employs a probabilistic context free grammar, a dictionary of Firefox-related language, and a set of parsing and filtering heuristics, to identify and aggregate problem topics indicated in the corpus.

In reflecting on this past work, the greatest successes to date have been in recognizing interface widgets and commands mentioned in tutorials. Recovering the parameters of those actions has proven to be more challenging [10]. Furthermore, performance quickly decreases when systems are tasked with performing (or fully modeling) the steps described by the tutorial [1,2,14]. In the sections that follow, we examine the reasons why extracting information beyond specific widgets and commands is challenging.

Method

We derive research challenges for machine understanding of tutorials from two sources: 1) failure cases and limitations explicitly reported in previous published research, and 2) an analysis of tutorials gathered for three feature-rich applications: GIMP, Inkscape, and MS Word. In the latter analysis, we analyzed over 2,000 clauses extracted from a random subset of tutorials for these applications. These clauses were initially collected as part of a project aimed at developing a taxonomy describing the types of information found in tutorials. It was during the course of developing this taxonomy that we gained an appreciation for the imprecise nature of tutorial text.

In presenting examples pulled from these tutorials, we use the notation (W) to denote examples selected from Microsoft Word tutorials, (G) to denote examples selected from GIMP tutorials, and (I) to denote examples selected from Inkscape tutorials.

We structure our presentation by briefly discussing well-known NLP challenges and how these challenges manifest themselves in software tutorials. We then take a bottom-up approach and describe the problems of interpreting *individual* isolated instructions, the problems associated with arranging instructions into *sequences*, and, finally, the problems encountered when determining which parts of a tutorial contain *actual procedural content*.

General NLP Challenges

Text-based instructional materials for software exhibit challenges well known in the fields of natural language processing and information extraction. We enumerate those problems here, and contextualize them in the domain of software tutorials.

Coreference resolution

A common challenge in natural language processing is coreference resolution, or the problem of resolving pronouns and other references, to the persons, objects or places to which they refer [8]. Correct interpretation of tutorial content also requires coreference resolution, as the following example, and those of Figure 2, illustrate:

> *"****That's*** *the one you need, so click* ***it****."* (W)

Despite the prevalence of coreference resolution discussions in the NLP literature, and with the exception of Frictionary [9], past work in tutorial mining has not acknowledged or attempted to address this challenge.

*"With your text highlighted, click the **'B' icon**."* (W)

*"Make sure that the **'x'** is selected, if not then please click the icon to rectify the problem."* (I)

Figure 3: Tutorial authors often describe UI widgets visually, rather than referring to them using their official names. In the top example 'B' refers to a button in Microsoft Word's interface that toggles the bold text style. Likewise, the 'x' in the lower example corresponds to a choice in Inkscape's color pallet that removes an object's background fill color.

Entity linking

Related to corefrence resolution is the problem of mapping entities mentioned in text to tools, commands, document objects, and other widgets in the target software application. This is an instance of the more general *entity linking* [7] problem in the information extraction literature (also known as *named entity disambiguation*). While the captions, labels, and official names given to user interface objects typically constrain the terms used by tutorial writers, numerous factors can give rise to a broader vocabulary and the use of synonyms for identical objects. For example, Ekstrand et al. noted that Inkscape users often refer to a "line tool" rather than to the correctly named "pen tool" because this tool is commonly used to draw straight lines [3]. Likewise, we have observed that GIMP tutorials often refer to an "Eyedropper" tool rather than the official name, "Color Picker," because the tool's icon depicts an eyedropper. Similar examples for MS Word and Inkscape are presented in Figure 3. There also exist extreme cases, where authors specify entities in manners that are especially novel or elaborate, giving rise to challenging entity resolution requirements:

> *"Click on the icon that **looks like a piece of paper with the top right corner folded down**." (G)*

Text Segmentation

Another challenge widely acknowledged in past work is that of splitting procedural text into discrete steps or operations [10,11,14]. As will be discussed in the next section, the concept of an operation is itself rarely precisely defined, but, independent of definition, there arise cases where a single operation spans multiple sentences, as well as cases where a single sentence contains numerous steps. The latter scenario is illustrated below:

> *"Once you've got the base image, you can **duplicate it** (Ctrl + D), **flatten the duplicate** (<Image> Layers -> Flatten Image) and then **experiment**."* (G)

The need to identify steps in streams of text is a specific instantiation of the more general NLP problem of text segmentation [12]. More common examples of text segmentation include word breaking, sentence splitting, chunking, and detecting topic boundaries. In the context of segmenting instructions into steps, Pedemonte et al. have demonstrated that conditional random fields (CRFs) can be successfully leveraged in limited contexts [14].

Extracting a single step or operation

While a seemingly intuitive concept, what constitutes a *single operation* in a tutorial is rarely defined in previous research. However, how one defines a single operation has important implications for developing systems that can extract individual instructions.

Consider the following example:

> *"Add a new transparent layer named Vignette."* (G)

On the face of it, this appears to be a straightforward operation to perform. However, deconstruction of this directive into a set of discrete operations will vary

In GIMP's main menu:

1. Click the "Layers ⇒New Layer" menu item.

In the "New Layer" dialog:

2. Enter "Vignette" as the name of the layer.
3. Select "Transparency" as the background fill type.
4. Click "Ok"

Figure 4: The 4 manual operations that a user must perform in GIMP's interface to perform the same task as GIMP's single 'gimp-layer-new' macro function.

"Increase the stroke width **until you see a nice smooth outline**." (I)

"Play around with the other ***settings until you get something that looks like a snowflake****"* (G)

Figure 5: Example tutorial phrases where parameters are specified by the effect that they have on the resultant document. These effects are often expressed as *purpose clauses*, which we describe later in the document.

depending on the goals of the system. If the goal is to generate a guided help system [11], then the extracted operations should probably characterize each action that a user is expected to perform in the interface. See Figure 4 for one interpretation of how this directive can be interpreted.

Conversely, if the goal is to automate instructions via something akin to executable scripts [10], then this directive may correspond to a single function in the application's programming interface (API) – indeed, GIMP's macro API provides a fully-parameterized method for creating new layers in one step.

Unfortunately, tutorials often contain a mix of instruction granularities, utilizing high-level expressions for common operations (often located at the beginning of a tutorial [1]), while expressing later operations in more detail. As noted earlier, Branavan et al. used reinforcement learning to map high-level instructions to low-level steps [1], and were able to correctly form these associations in 64% of test cases. Improving these results remains important future research.

As work progresses in this space, it is important to explicitly state assumptions about what levels of granularity one hopes to achieve when extracting operations from texts.

Command parameters

The correct interpretation of tutorial steps requires the extraction and interpretation of commands, together with their parameters. Notably, many parameter values are implied, underdefined, or described with constraints, leading to greater variability in how these values are expressed. As a result, F1 scores for extracting parameter values tend to be lower in the literature. For example, Laput et al. [10] report an F1 score of 0.97 for detecting menu items, with an F1 score of 0.36 for detecting command parameters.

In many tasks, there is some flexibility in deciding on parameter values, leading authors to intentionally underspecify the value of parameters, as this example illustrates:

> *"Set its mode to Screen and* ***adjust Opacity to suit****."* (G)

In other cases, constraints are attached to these underspecified parameter recommendations:

> *"Feather the selection by* ***75~85*** *pixels or so."* (G)

Constraints on desired parameter values can also be quite complex:

> *"Set your foreground color to* ***a slightly darker and muted blue than you used for the background."*** (G)

> *"You can use any size you wish as long* ***as you keep that ratio****."* (I)

Finally, authors may choose to indicate the desired output of an operation, leaving the exact parameter settings to be discovered through trial and error. The following example, and those of Figure 5, illustrates this case:

> *"Crank down the opacity of the upper layer* ***so that you can see both images****." (G)*

"Just drag the mouse/pen ***along the bottom*** *of the curves tool." (G)*

"Duplicate this setup and move that tile ***above this one****, remembering to snap it to grid." (I)*

"Drag the arrow to about ***the third line****."* (G)

Figure 6: Example tutorial phrases where spatial reasoning is required for correct interpretation.

*"****If using GIMP for Windows****, you'll have to right-click on the destination button and select the Foreground."* (G)

*"****You might want*** *to decrease the opacity of this layer."* (I)

Figure 7: Examples of conditioned tutorial steps. The first example depends on system state, and may be evaluated automatically. The second example depicts an optional step that must be decided by a human operator.

These examples define numerous challenges for extracting parameter values for forms-based input, but there is a corresponding set of related problems for specifying direct manipulation operations. We describe these challenges next.

Spatial reasoning

User interfaces are two- (and sometimes three-) dimensional, and often employ direct manipulation as an interaction technique. As such, tutorials often make use of spatial information when describing processes, and can include directives that reference absolute and relative spatial coordinates. Examples of these types of instructions can be found below, and in Figure 6:

> *"Click your left mouse button* ***about half way up the page****, and type a title."* (W)

These operations can be challenging to interpret, requiring knowledge of the spatial relationships between components of the interface, as well as the capacity to do spatial reasoning.

Constraining action through purpose clauses

Tutorial instructions often contain *purpose clauses* [4], which state or imply the goal of the operation. These clauses may lead readers to infer additional constraints on their actions, thus posing a serious challenge for machine interpretation. Consider the following example, also noted in the introduction:

> *"Place it underneath the original text,* ***as if it were a reflection.****"* (G)

Here the previous challenges in interpreting spatial instructions apply, but correct execution of the instruction requires consideration of the purpose clause "as if it were a reflection". This clause serves to constrain the position of the object in a manner that can be discerned only if one understands the concept of reflection. Additional examples of instructions containing purpose clauses are listed in Figure 5 on the preceding page.

Sequencing steps

Tutorials contain steps that must be performed in the correct sequence in order to complete the stated task. However, tutorials can vary in how these steps are communicated, and what must be done versus what is optional.

Conditionals

The simplest deviation from a sequential workflow, and the case most frequently discussed in previous work (e.g., [2,11]), is where one step or operation is conditioned on some criteria. For example:

> *"****If yours says DOCX instead of DOTX*** *then click on the Save as Type box to see the file type menu."* (W)
>
> *"****If you plan on printing your card****, make the X and Y resolutions at 300dpi."* (G)

The examples (and those of Figure 7) highlight two important classes of conditional steps: The first example is conditioned on document state, and could conceivably be evaluated by an automated system. Conversely, the second example is an *optional* step, and is inherently problematic for automation. The distinction between the classes has not been discussed in past work.

*"**Do not** deselect them after duplication."* (I)

*"**There is a problem, however, if you try to** scale this image."* (G)

Figure 8: Examples of anti-patterns or warnings found in online tutorials. Anti-patterns are problematic because they resemble regular tutorial steps in that they mention specific commands, but their actions should be avoided rather than performed.

*"**The layer mask works in such** a way that all black parts of the mask will become transparent parts of the layer."* (G)

*"**The threshold plug-in works by** dividing the image into two parts, dark and light, producing a 2 color image."* (G)

Figure 9: Examples of non-procedural tutorial content that provides background information, but may be confused for tutorial steps.

Alternatives

Similar to conditionals, tutorials often present a series of alternative methods for achieving a stated goal. These are not conditionals, but, rather, represent distinct means of completing the same task or subtask.

Alternatives are potentially problematic for interpretation because systems must accurately recognize and distinguish each of the alternatives as separate entities. Typically, alternatives appear as self-contained sections of a tutorial, but are often foreshadowed by short indicator phrases. For example:

> *"Although you get those commands automatically, you can customize the Quick Access Toolbar,* ***and there are three ways to do this****."* (W)

Prior work has largely ignored the challenges posed in identifying and segmenting alternatives.

Anti-patterns

Finally, tutorials occasionally contain steps that are provided as counterexamples, or *anti-patterns*. These are often offered as warnings, which must be heeded in order to correctly perform the tutorial. The following example, and those of Figure 8, fit these criteria:

> *"**Whatever you do, don't** create a hanging indent by pressing the space key to create spaces, or even by tabbing across the page."* (W)

These anti-patterns are troublesome in that, like valid operations, they often explicitly name actions, targets, and parameters in their text. Systems that specifically pick out such features (e.g., [3,5,6,10,11]) may make the mistake of performing or recommending the precise errors that the tutorial author is warning against. While past work has not discussed anti-patterns, related research areas within NLP, such as sentiment analysis, suggest possible solutions or strategies for recognizing and reasoning about these steps.

Handling non-procedural text

Tutorials can contain rich narratives with motivations, background material, theory, best practices, and a host of other non-procedural material. To correctly segment tutorial steps, a system must be able to distinguish between procedural and non-procedural content. In many cases, this classification is straightforward – procedural content can be recognized by locating sentences that mention specific user interface widgets, actions, or settings [3,5,10,11]. However, there are a few common scenarios where this strategy breaks down. Consider the following example, as well as those from Figure 9, in which authors provide a theoretical background for complex operations:

> *"**Unsharp mask works by** increasing the contrast between edges in the photos (areas of color or tonal transition), making them appear sharper."* (G)

These examples are remarkably similar to steps that might be performed by the reader, and could cause potential confusion in an automated system.

A similar difficulty arises with a type of non-procedural content that Lau et al. call *verification steps* [11]. Verification steps are those that describe the current or future state of the document or system so that the reader knows what to expect. As the following example demonstrates, verification steps also have the potential to be confused for procedural steps:

> *”You will see the image go to black and white, with everything to the left of the selection going to black, and everything in the selection going to white.”* (G)

Finally, tutorial authors often begin or end passages by summarizing the operations that will, or have been, performed. For example:

> *“We'll be adding two more layers.”* (G)

This content is not meant to be executed, but may easily be confused with the tutorial's actual procedural steps. If confused for procedural content, then it may result in a given step being performed more than once. Additional examples are listed in Figure 10.

“On the background layer, we'll apply a gradient.” (G)

“Step 1: Lower the Saturation” (G)

Figure 10: Examples of passages that summarize the operations that are about to be prescribed. Such topic phrases can easily be mistaken for actual procedural content, and could lead to the duplication of some tutorial steps.

In summary, tutorial passages that provide background or theory, together with verification steps and summary sections, can often be confused as part of the tutorial procedure, posing challenges to automated systems.

Discussion

As we argued in the Introduction, online instructional materials contain a wealth of information; from the actual procedural steps they contain, to clues about the target audience, the amount of creativity or input expected of users, or common errors or misperceptions that are likely to occur.

In our work in this area, we have found that one of the most pressing, overarching problems to solve is that of precisely defining and characterizing the information content found in these tutorials. For example, even defining what is meant by a tutorial *step* is a challenging endeavor. A reliable, valid coding scheme is needed to annotate steps and other tutorial content, in order to enable subsequent machine reasoning. This problem is at the root of many of the challenges described above, and is essential to solve in order to make meaningful progress in this space. Most immediately, a coding scheme would enable the creation of labeled training sets for further research in supervised learning approaches.

The availability of a coding scheme could also reduce our dependency on machine learning for mining data from tutorials – a well-designed coding scheme could guide the establishment of a semantic markup format for web tutorials. This markup format would provide machine-readable cues for extracting steps, commands, parameters and other items of interest, therefore mitigating many of the challenges presented in this paper. There is precedent for this strategy: Microformats and Microdata are two standardized semantic markup extensions for HTML, and both conventions include methods for describing cooking recipes [13,15]. Cooking recipes are similar to tutorials in that they prescribe a series of steps to be taken.

In short, paralleling early work in NLP, there is a need to develop more standardized ways for labeling tutorial content, enabling the creation of corpora for training and research, and perhaps offering other ancillary uses.

Conclusion

In this paper, we outlined many of the challenges posed to systems that seek to mine procedural information from written software tutorials. In addition to common NLP challenges, we argued that there are complex issues surrounding the interpretation of individual steps, the sequencing of sets of steps, as well as identifying which parts of a tutorial text provide actual procedural information. Thus, while we doubt that the full machine automation of general tutorials is tractable in the near future, we strongly believe that tutorials are a valuable resource to be mined and analyzed using automated techniques. Moreover, by acknowledging these challenges, we identify new targets and opportunities that may yield interesting applications distinct from those explored in past work.

References

[1] Branavan, S.R.K., Zettlemoyer, L.S., and Barzilay, R. Reading between the lines: learning to map high-level instructions to commands. In *Proc. ACL '10*, Association for Computational Linguistics (2010), 1268–1277.

[2] Brasser, M. and Linden, K.V. Automatically Eliciting Task Models from Written Task Narratives. In C. Kolski and J. Vanderdonckt, eds., *Computer-Aided Design of User Interfaces III*. Springer Netherlands, 2002, 83–90.

[3] Ekstrand, M., Li, W., Grossman, T., Matejka, J., and Fitzmaurice, G. Searching for software learning resources using application context. In *Proc. UIST '11*, ACM (2011), 195–204.

[4] Di Eugenio, B. Understanding natural language instructions: the case of purpose clauses. In *Proc. ACL '92*, Association for Computational Linguistics (1992), 120–127.

[5] Fourney, A., Lafreniere, B., Mann, R., and Terry, M. "Then click ok!": extracting references to interface elements in online documentation. In *Proc. CHI '12*, ACM (2012), 35–38.

[6] Fourney, A., Mann, R., and Terry, M. Query-feature Graphs: Bridging User Vocabulary and System Functionality. In *Proc. UIST '11*, ACM (2011), 207–216.

[7] Han, X., Sun, L., and Zhao, J. Collective Entity Linking in Web Text: A Graph-based Method. In *Proc. SIGIR '11*, ACM (2011), 765–774.

[8] Jurafsky, D. and Martin, J.H. Section 21.7: Coreference Resolution. In *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson Prentice Hall, 2009, 710–718.

[9] Ko, A. Mining whining in support forums with frictionary. In *Proc. CHI EA '12*, ACM (2012), 191–200.

[10] Laput, G., Adar, E., Dontcheva, M., and Li, W. Tutorial-based interfaces for cloud-enabled applications. In *Proc. UIST '12*, ACM (2012), 113–122.

[11] Lau, T., Drews, C., and Nichols, J. Interpreting Written How-To Instructions. In *Proc. IJCAI'09*, (2009), 1433–1438.

[12] McDonald, R., Crammer, K., and Pereira, F. Flexible text segmentation with structured multilabel classification. In *Proc. HLT '05*, Association for Computational Linguistics (2005), 987–994.

[13] microformats.org. hRecipe 0.22. 2013. http://microformats.org/wiki/hRecipe.

[14] Pedemonte, P., Mahmud, J., and Lau, T. Towards automatic functional test execution. In *Proc. IUI '12*, ACM (2012), 227–236.

[15] schema.org. Thing > CreativeWork > Recipe. nd. http://schema.org/Recipe.