Mining Whining in Support Forums with Frictionary

Amy J. Ko
The Information School
University of Washington
ajko@uw.edu

Abstract

Millions of people request help with software in support forums, creating a massive repository of user experiences ripe for mining. We present Frictionary, a tool for automatically extracting, aggregating, and organizing problem described in support forums, enabling timely problem frequency and prevalence metrics. We applied it to 89,760 Firefox support requests from 4 sources gathered over 10 months. Interviews with the Firefox principal designer and support lead suggest that Frictionary could be a useful tool for prioritizing engineering efforts, but that the extraction would need to be more precise to be useful.

Author Keywords

Software problems; bugs; topic extraction; support.

ACM Classification Keywords

H.5.2 [Information Interfaces and Presentation]: User Interfaces - Evaluation/methodology

Introduction

Each day, millions of people struggle to make software meet their needs, recovering from frustrating crashes, disabling nag windows, and even just learning an application's basic features. As much as software producers strive to prevent these negative experiences through careful upfront design and rigorous testing, there are often many unanticipated problems discovered post-deployment [13].

Unfortunately, discovering these problems is not always straightforward [6]. Users rarely contact support, and when they do, it is often only for the most critical and idiosyncratic of problems [5]. Moreover, support staff primarily report bugs to engineers, overlooking user experience issues [6]. Instrumentation (e.g., [1]) can overcome some of these limitations, providing large scale data about user experiences, but such data requires careful interpretation since it lacks user intent. While these methods are all useful, new methods are needed for discovering problems at a large scale.

With the rapid rise of social media, this is increasingly feasible. Users can easily share their experiences in discussion boards [19], support forums [6], Q&A sites, other venues, describing software problems, expressing frustration, and sometimes getting help. For any given software product, users may write thousands of requests like this every day, creating a massive catalogue of user experience. How can software producers automatically mine to learn what issues users are commonly experiencing?

We contribute *Frictionary***,** which extracts, aggregates, and organizes *problem topics* in users' requests. Frictionary uses a natural language parser to perform linguistic pattern matching, extracting problematic software behaviors in a consistent grammatical form. It then groups them, ranking them by frequency and prevalence to reveal trending problems over time. This work contributes: (1) a new pattern matcher that distinguishes between software problems and other topics in support requests; (2) a collection of techniques for transforming problems into standardized grammatical forms; (3) a faceted browsing interface that reveals trends; and (5) evidence that a Mozilla support lead and the Firefox principal designer believe Frictionary could help prioritize engineering efforts, but that it's topic extraction would need to be more precise.

Related work

To our knowledge, there is no prior work that extracts topics from support requests. There are, however, several techniques for extracting topic from similar data. Perhaps the most recent and related work is by Fourney et al. [10], in which they describe a technique for mining frequent queries about software from Google Suggest, using a set of filtering templates such as "can *system* __" and "how to ___ in *system*". This is a powerful approach, backed by a vast number of searches. The authors identify several limitations, however: queries do not necessarily indicate problems, products with generic names may result in irrelevant queries, and the timeliness of data is limited to an estimated 20 day window, which may be too long for the rapid release cycle of many software producers.

Researchers have also extracted topics from bug reports, one form of support request. Work has varied in the granularity of extraction. For example, some have considered reports at the level of sentences, using text summarization to select sentences that summarize a bug report discussion for reading purposes [17]. Others have focused on document characterization, most commonly by adapting *tf-idf* vector space models and cosine similarity metrics to bug report text [19]. This approach, which is commonly used in information retrieval problems, identifies words that are common in a document, but rare in a corpus. Researchers have also combined vector space models with program execution information [9], both supervised and unsupervised machine learning techniques [7], and information entropy [20]. Other researchers have focused on discriminating between certain classes of problems, such as bugs and features [2].

Researchers have also adapted information extraction techniques for other social media. For example, Naaman et al. extracted trends from raw Twitter

1) lost 1 bookmark

   help! **one of my bookmark folders disappeared, containing many bookmarks I use all the time**. What can I do?? Thanks!

2) (S (NP (NP (CD **one**)) (PP (IN **of**) (NP (PRP$ **my**) (NN **bookmark**) (NNS **folders**)))) (VP (VBD **disappeared**) (, ,) (S (VP (VBG **containing**) (NP (NP (JJ **many**) (NNS **bookmarks**)) (SBAR (S (NP (PRP **I**)) (VP (VBP **use**) (NP (PDT **all**) (DT **the**) (NN **time**))))))))) (. .))

3) 1) (S (NP (NP (CD **one**)) (PP (IN **of**) (NP (PRP$ **my**) (NN **bookmark**) (NNS **folders**)))) (VP (VBD **disappeared**)))
   2) (S (VP (VBG **containing**) (NP (NP (JJ **many**) (NNS **bookmarks**)))))
   3) (S (NP (PRP **I**)) (VP (VBP **use**) (NP (PDT **all**) (DT **the**) (NN **time**))))

4) 1) subject=[**bookmark**, **folders**], verb=[**disappeared**], object=[]
   2) subject=[], verb=[**containing**], object=[**bookmarks**]
   3) subject=[**I**], verb=[**use**], object=[**time**]

5) 1) feature=**bookmark folder**, action=**disappearing**
   2) none
   3) none

Figure 1. (1) an request's 3rd sentence as (2) a parse tree, (3) clauses, (4) subject/verb/object words, and (5) topics.

streams using a notion of bursts based on tf-idf metrics [16]. Bernstein et al. used a similar approach, extracting topics from both tweet text and text retrieved by performing web searches on tweet words [3]. Yatani et al., focused on product characterizations, used part-of-speech taggers to extract adjective-noun pairs in user reviews [21]. Chen et al. considered the similar problem of identifying trending topics in news, using time-based named entity recognition [4].

Support requests pose many unique extraction challenges. Unlike bug reports, which tend to be limited strictly to reproduction steps and unexpected output [15], support requests have non-problem topics such as user goals, attempted workarounds, and personal consequences of a problem, and emotions [18], as well as error codes, logs, and other non-word text, which may be related to a problem, but not describe it [19]. Also, unlike in social media and news, where named entities are more easily shared via hash tags and headlines, support request authors tend to describe software features with wildly different phrases [11], even within the same request. This limits the effectiveness of named entity recognition techniques.

Extracting topics from support requests

Frictionary leverages insights about the *genre* of support requests, applying natural language parsing to discriminate between software problems and other topics. The core insight underlying this approach is that support request authors generally identify two types of problems: *undesirable output* (bugs, errors, unexpected output, etc.) and *desirable output* (feature requests, how to questions, etc.). Non-problem topics in a support request generally describe the user's actions and state, such as workarounds the user has tried, how they feel about the problem, or why the problem matters to them. Frictionary operationalizes these insights to extract problem topics.

To demonstrate and evaluate Frictionary, we used a corpus of 89,760 support requests about the Firefox web browser from 4 sites across a period of 10 months. We chose Firefox because it is a widely used consumer application with millions of users and has many distinct online technical support communities. We focused on 4, including **71,072** requests posted to *http://www.mozilla.org/support* (provided by Mozilla), **9,783** Mozilla bug reports from *http://bugzilla.mozilla.org* (downloaded as XML), **8,212** discussion threads from four Firefox support forums at *http://mozillaZine.org*—including the forums titled *support* (6,550), *general* (1,019), *bugs* (323), and *features* (320)—and **693** questions posted at *http://superuser.com* and tagged *firefox* (SuperUser is a Q&A support site).

All posts were written between Nov. 1st, 2009 and July 23rd, 2010, which was the time window of the Mozilla support dataset. The data spanned 4 releases of Firefox (versions 3.5.5, 3.5.6, 3.5.7, and 3.6) and contained a total of 47,815 unique usernames.

Parsing and Filtering Sentences

Given a single support request (ignoring replies), Frictionary starts with the whitespace-preserved text of a request, including a title and body, as in Figure 1.1. The text is then segmented into line break separated paragraphs (not shown) and then into sentences and word tokens using the Stanford sentence tokenizer [12]. The sentences are then provided to the Stanford probabilistic context free grammar parser [12]. The resulting parse trees are tagged with parts and phrases of speech, as in Figure 1.2.

In addition to English words, requests can also include logs, error codes and other non-natural language text. Frictionary uses two heuristics to exclude such text. First, it excludes any sentences with more than 100 words, since most English sentences have fewer than

50. Second, for each word in a sentence, Frictionary determines whether each word is all letters with an optional hyphen (e.g., *pigeon-hole*), all digits (e.g., *6284*), or a sequence of identical punctuation marks (e.g., *!!!* or *???*). If a word does not conform to one of these three patterns, it is classified as a non-word. If a sentence is more than 20% non-words, the sentence is excluded from extraction. This allowed for sentences to contain 1 or 2 non-words, such as quoted error codes.

Extracting Topics from Subject-Verb-Object Patterns
Once Frictionary filters sentences, it analyzes sentence *clauses* (subject/verb/object sets). The goal of this part of the analysis is to distinguish problem clauses from non-problem clauses by looking for patterns in these subjects, verbs, objects, and other clause attributes. To

kind	problem	example clauses	verb	animate	copular	past	desire	context	able	not
fragment	×	*firefox tabs*	×	-	-	-	-	-	-	-
user state	×	*I was angry*; *I am angry*; *I was not angry*; *I am not angry*; *I can be angry*; *after I was angry*; *I would be angry*; etc.	✓	✓	✓	-	-	-	-	-
user workaround	×	*I tried clicking the x*; *I tried not clicking the x*	✓	✓	×	✓	-	-	-	-
user behavior	×	*I close tabs a lot; I don’t close tabs; I won’t close tabs*	✓	✓	×	×	-	×	×	-
user ability	×	*I can open windows*	✓	✓	×	×	×	×	✓	×
user consequence	×	*Once I can open a tab...; when I can't open a tab...*	✓	✓	×	×	×	✓	✓	-
non-grammatical	×	*I will could open a tab.*	✓	✓	×	×	✓	-	✓	-
user inability	✓	*I cannot open windows*	✓	✓	×	×	×	×	✓	✓
user input	✓	*after I open a tab; if I don't open a tab; after I would open a tab*	✓	✓	×	×	-	✓	×	-
problematic behavior	✓	*firefox tabs will not close; firefox tabs should close*	✓	×	×	-	-	-	-	-
problematic state	✓	*Firefox is slow; tabs are stuck.*	✓	×	✓	-	-	-	-	-

Table 1. A truth table defining Frictionary’s clause classification function, via 8 attributes (in columns). A - indicates a “don’t care”. Frictionary extracts topics matching the last 4 rows.

begin, Frictionary identifies clauses by finding the *S*-nodes in a clause’s parse tree. For example, the tree in Figure 1.2 contains the three clauses in Figure 1.3, two of which are children of the root clause. Frictionary also splits any clause with a coordinating conjunction child into two clauses (as in (S (NP (NP (NNP FF) (NNP freezes)) (CC and) (NP (DT a) (JJ dialog) (NN box))) (VP (VBZ opens)))) since they may share a subject, but have different verbs and objects.

Next, Frictionary extracts subject, verb, and object words from each clause (ignoring descendent clauses, since they are analyzed independently), as follows. **Subject** words include all nouns and gerunds of the 1st noun phrase preceding a verb (e.g., *bookmark folders* in Figure 1.4.1). If there is no noun phrase (as in Figure 1.3.2), there is no subject. **Verb** words include (1) the 1st finite verb of the 1st verb phrase following a *to* (if there is one) and preceded by a coordinating conjunction, subordinate clause, or prepositional phrase (if present), and (2) the word *not,* if modifying the selected verb, to capture the polarity of the sentence. In Figure 1.4, the verbs are *disappeared*, *containing*, and *use*. **Object** words include all nouns, adjectives, past participle verbs, and gerunds following the selected verb and preceded by a coordinating conjunction, subordinate clause, or prepositional phrase. Figure 1.4 shows the object words extracted from the example sentence.

Next, Frictionary uses the subject, verb, and object words to compute the 8 clause attributes listed in the rightmost columns of Table 1. The **verb** attribute is true if the clause has a verb. Clauses with no verb are unlikely to explicitly indicate a problem. For example, the clause *firefox tabs* indicates a software feature, but not an undesirable behavior or state. The **animate** attribute is true if the subject of a clause is a personal pronoun other than it (e.g., I, we, you, he, she, they, etc.). This is a critical factor in determining whether a clause regards software or something else. The **copular** attribute is true if the clause has an exclusively copular verb, providing information about the subject. This includes all conjugations of the verbs be, seem, and become (e.g., is, am, are, was, were, etc.). This does not include optionally copular verbs, such as appears, because their role is often ambiguous. The **past** attribute is true if if the clause has a past tense verb. The **desire** attribute is true if the clause has any of the English modal verbs *will*, *would*, *shall*, or *should*, which tend to indicate desired states [14]. The **context** attribute is true if the clause has a temporal *wh-adverbs* (namely *when*, but also *whence*, *whereupon*, and *wherein*) or a temporal preposition (*when*, *if*, *while*, *before*, *after*, or *until*). These tend to indicate the action the user input that preceded a problem [14]. The **able** attribute is true if the clause contains the modal verb *can* or *could*, which tend to indicate the possibility of action. And finally, the **not** attribute is true if *not* modifies the clause's verb.

Frictionary uses these 8 boolean attributes to compute the function defined by Table 1's truth table. The function identifies clauses that indicate (1) actions a user took that caused a problem, (2) actions the user is unable to take, (3) problematic software behaviors, and (4) problematic software state. These are the last 4 rows of Table 1; all other clauses are excluded. For example, of the three clauses in Figure 1.4, only the first was kept (Figure 1.5). It is worth noting that none of the patterns above attempt to address the word *it* (resolving such pronominal anaphora is a long-standing challenge in natural language processing).

Once Frictionary selects problem clauses, it applies one last filter to ensure that the clauses regard the software and not other software or inanimate objects. To do this, Frictionary takes as input all of the English localization files for an application of all versions of the software, containing all user interface labels and error messages that can be displayed in the application. Frictionary then parses each user interface string, extracting nouns and verbs, and stems them, storing them in application terminology dictionary. With this dictionary, Frictionary then excludes any problem clause whose subject, verb, and object lack any known application noun or verb (using the stemmed version of each word).

After this last filtering, Frictionary converts the subject, verb, and object words (as in Figure 1.4) into into a *feature/action* phrase. It first lemmatizes each word with the Stanford stemmer [12] (e.g., *closed* into *close*) and then converts verbs to *gerund* form (nouns ending with *-ing*). This normalizes the topic's words, ensuring that topics with the same lemmatized word are equivalent when compared as strings. Lastly, the topic is created by adding to a list all subject words, in order, then the object words, in order, creating the *feature* of a topic. The gerund form of the verb then becomes the topic's *action*. Figure 1.5 shows examples of resulting topics. Lastly, the topic extracted from a clause added to the set of topics accumulated for a request; string equivalent topics appear only once. The extraction process is then repeated for all clauses of all sentences of all support requests in the provided corpus.

Aggregating Problem Topics

After Frictionary extracts topics from the requests, it groups topics by addressing differences in spelling, spacing, hyphenation, phrasing, and word choice. First, Frictionary uses the WordNet [8] database to find synonyms of application terminology. WordNet is a large graph of English nouns and verbs, connected by various relationships, one of which is a synonym relationship. Frictionary goes through each topic, and for each word that is *not* a application term that also has a single synonym that *is* an application term, replaces the word with the synonymous application term. For instance, in the WordNet corpus, *background* is the only synonym of *wallpaper* that appears in the Firefox UI, and therefore, topics that use the word *wallpaper* are rephrased with the word *background*. Terms that are synonymous with multiple terms remain unchanged, as they may have more nuanced meanings.

After synonym renaming, Frictionary standardizes topics by performing the following pairwise topic comparisons. If two topics' features are equivalent after stripping hyphens and plurals (e.g., ***plug-in*** vs ***plugins***), both topics' features are relabeled without hyphens and plurals (***plugin***). If two topics' features are equivalent after removing one or more spaces between two feature words (e.g., ***fire bug plug in*** vs. ***firebug plugin***), both topic's features are relabeled with the more frequent topic in the corpus. If the Levenstein string distance [16] between the two feature strings or verbs is 1 (e.g., ***forefox*** vs. ***firefox***), the more frequent spelling in the corpus is used. Finally, if any words appear in a custom dictionary of application-specific synonyms, those words are remapped to the standardized form. For example, people may refer to Firefox also as FF; this gives Frictionary users the ability to improve the aggregation with terminology that may not appear in the software's user interface.

Once each topic in the corpus is standardized, Frictionary creates string-equivalent topic groups. For example, all topics with the phrase ***firefox crashing*** would be combined into a single set. Frictionary then performs one final aggregation to address differences in phrasing, merging topics whose feature words are a superstring of another topic's feature words, but with the same action words. For example, Frictionary adds the topics matching ***firefox window crashing*** to the group of topics matching ***firefox crashing***, while preserving the ***firefox window crashing*** group. This enables for some variation in how features are phrased.

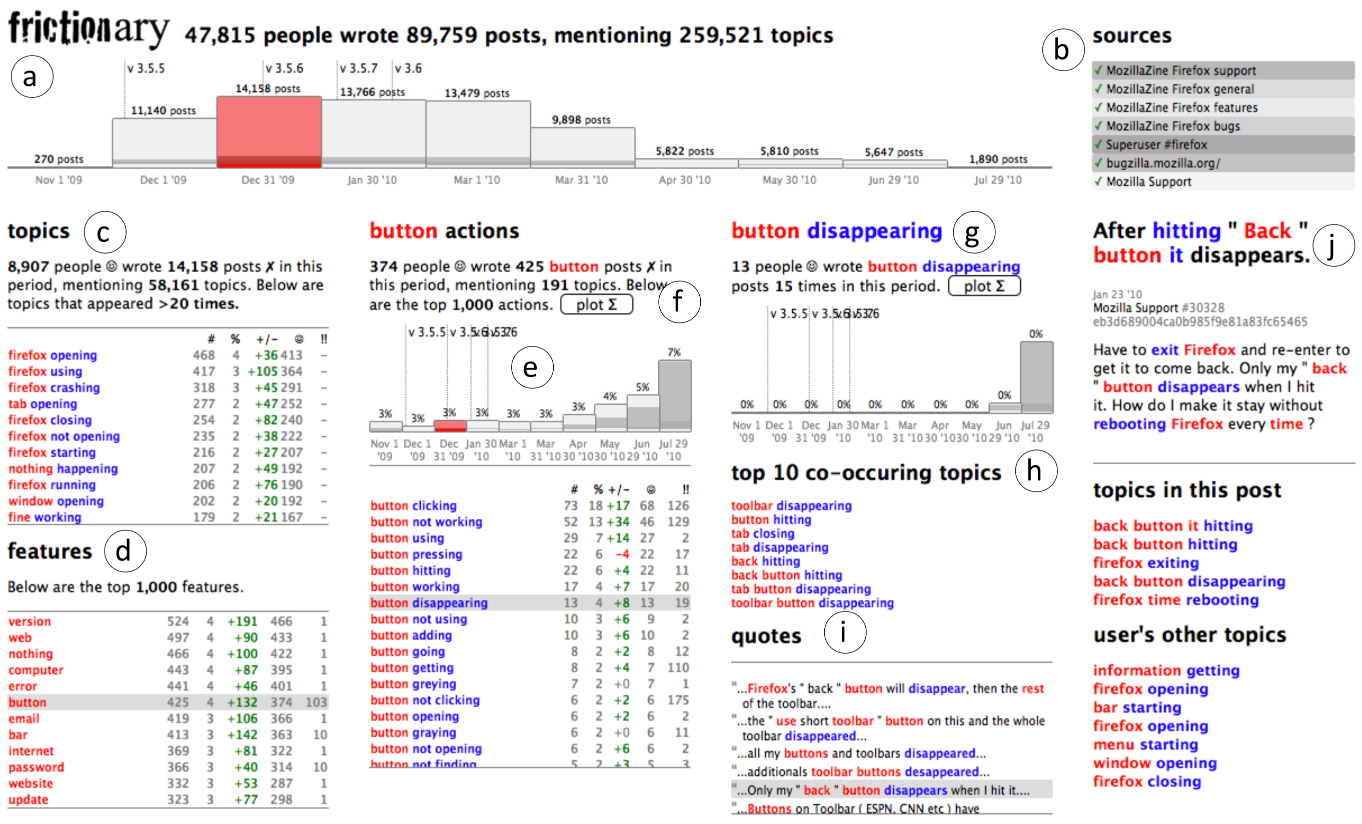
Faceted Browsing of Problem Topics

To help software producers make sense of software support topics, Frictionary provides the topic browsing interface in Figure 2. The interaction flows from the bar chart at the top (Figure 2a), which visualizes the number of requests in a 30 day period. The bar chart also plots a vertical line for each release of the software. Each bar is also divided into stacks representing the request counts from each data source. The sources are listed at Figure 2b; toggling them includes and excludes their data from plots and tables.

Clicking on any bar in the frequency plot at Figure 2a shows two tables: the *topics* table (Figure 2c) and the *features* table (Figure 2d). The *topics* table contains all of the topics in the selected time period, sorted by one of the columns. For example, the screenshot shows that in the period starting December 31st '09, which contained the release of Firefox 3.5.6, the most common topics were ***firefox opening***, ***firefox using***, ***firefox crashing***, ***tab opening***, and so on. The *features* table contains the top 1,000 most frequent topic *features* (e.g., ***version***, ***web***, ***nothing***, ***computer***, ***error***, etc.)

Clicking on a feature in the features table shows the topics for that feature (Figure 2e). This table also provides a bar chart plotting the proportion of requests containing a topic with the selected feature in each period. For example, the feature ***button*** appeared an increasing proportion of requests over time. The button above the chart (Figure 2f) toggles between plotting the *proportion* and number of requests containing the feature. Clicking this reveals that while the proportion of requests containing the ***button*** increased, the absolute count decreased after 3.5.6 but then increased again a few months later.

Clicking on a topic, either in the topic table (Figure 2c) or the actions table (Figure 2e), shows the *topic* view



47,815 people wrote 89,759 posts, mentioning 259,521 topics

sources

- ✓ MozillaZine Firefox support
- ✓ MozillaZine Firefox general
- ✓ MozillaZine Firefox features
- ✓ MozillaZine Firefox bugs
- ✓ Superuser #firefox
- ✓ bugzilla.mozilla.org/
- ✓ Mozilla Support

topics

8,907 people ☺ wrote **14,158** posts ✗ in this period, mentioning **58,161** topics. Below are topics that appeared **>20 times.**

	#	%	+/−	☺	!!
firefox opening	468	4	+36	413	–
firefox using	417	3	+105	364	–
firefox crashing	318	3	+45	291	–
tab opening	277	2	+47	252	–
firefox closing	254	2	+82	240	–
firefox not opening	235	2	+38	222	–
firefox starting	216	2	+27	207	–
nothing happening	207	2	+49	192	–
firefox running	206	2	+76	190	–
window opening	202	2	+20	192	–
fine working	179	2	+21	167	–

features

Below are the top **1,000** features.

version	524	4	+191	466	1
web	497	4	+90	433	1
nothing	466	4	+100	422	1
computer	443	4	+87	395	1
error	441	4	+46	401	1
button	425	4	+132	374	103
email	419	3	+106	366	1
bar	413	3	+142	363	10
internet	369	3	+81	322	1
password	366	3	+40	314	10
website	332	3	+53	287	1
update	323	3	+77	298	1

button actions

374 people ☺ wrote **425** button posts ✗ in this period, mentioning **191** topics. Below are the top **1,000** actions.

	#	%	+/−	☺	!!
button clicking	73	18	+17	68	126
button not working	52	13	+34	46	129
button using	29	7	+14	27	2
button pressing	22	6	−4	22	17
button hitting	22	6	+4	22	11
button working	17	4	+7	17	20
button disappearing	13	4	+8	13	19
button not using	10	3	+6	9	2
button adding	10	3	+6	10	2
button going	8	2	+2	8	12
button getting	8	2	+4	7	110
button greying	7	2	+0	7	1
button not clicking	6	2	+2	6	175
button opening	6	2	+2	6	2
button graying	6	2	+0	6	11
button not opening	6	2	+6	6	2
button not finding	5	2	+3	5	3

button disappearing

13 people ☺ wrote button disappearing posts **15** times in this period.

top 10 co-occuring topics

- toolbar disappearing
- button hitting
- tab closing
- tab disappearing
- back hitting
- back button hitting
- tab button disappearing
- toolbar button disappearing

quotes

"...Firefox's " back " button will disappear, then the rest of the toolbar....

"...the " use short toolbar " button on this and the whole toolbar disappeared...

"...all my buttons and toolbars disappeared...

"...additionals toolbar buttons desappeared...

"...Only my " back " button disappears when I hit it....

"...Buttons on Toolbar (ESPN, CNN etc) have

After hitting " Back " button it disappears.

Jan 23 '10
Mozilla Support #30328
eb3d689004ca0b985f9e81a83fc65465

Have to exit Firefox and re-enter to get it to come back. Only my " back " button disappears when I hit it. How do I make it stay without rebooting Firefox every time ?

topics in this post

- back button it hitting
- back button hitting
- firefox exiting
- back button disappearing
- firefox time rebooting

user's other topics

- information getting
- firefox opening
- bar starting
- firefox opening
- menu starting
- window opening
- firefox closing

Figure 2. The Frictionary user interface, enabling users to browse topics and sort them by frequency and prevalence metrics.

(Figure 2g). This view shows the same kind of plot as in the actions view, but for the selected topic. Below the plot, it shows the top 10 topics that also occur in requests containing the selected topic (Figure 2h). For example, one frequently co-occurring topic with ***button disappearing*** in this period was ***toolbar disappearing***. This allows Frictionary users to better interpret the meaning and context of the selected topic.

The bottom of topic view also includes all of the clauses containing the selected topic (Figure 2i). Clicking on one of these shows the *request* view (Figure 2j), which includes the title of the request, the request text, the source, and the date it was written. This view also lists all of the topics extracted from the request and a list of all of the other topics extracted from requests by the same user. These views are intended to give a sense of whether the user is a frequent requester and if so, for what topics they often request help.

All tables have 5 sortable columns of statistics. The 1st column (#) is the absolute number of requests that contain the feature or topic in the selected period; by default, all tables are sorted in decreasing order with this metric. The 2nd column (%) is the proportion of requests in this period containing this topic. The 3rd column (+/−) shows the change in number of requests containing the feature or requests containing the feature or topic relative to the previous period. This enables Frictionary users to see whether a feature or topic has changed in frequency in the last 30 days. The 4th column (☺) shows the number of unique users mentioning this topic. For example, in Figure 2c, 277 requests mentioned ***tab opening***, but these requests were only written by 252 users, meaning that some individuals wrote multiple requests mentioning the same topic. (Of course, some people may have multiple accounts on a site, so there is no guarantee that this number actually represents the number of unique individuals). This metric may help Frictionary users see how prevalent an issue is: the fewer the number of unique users mentioning a topic, the more the topic might be the idiosyncratic concern of a vocal minority.

The last column (!!) represents a metric we call *vocality*, which is intended to measure the extent to which an issue is mentioned by "vocal" users. This is operationalized by computing the median number of requests written over all periods by the users mentioning the selected topic. For example, the ***button clicking*** topic shown below Figure 2e has a vocality of 126, meaning that the median number of requests written by the 68 users mentioning the topic was 126—these are prolific help seekers. In contrast, ***button adding***, further down the table, has a vocality of 2, meaning that the users mentioning it were less prolific.

Evaluating Topic Extraction and Utility

From the 89,760 support requests in our corpus, Frictionary extracted 77,349 unique feature phrases, 9,120 unique action phrases, and, combined, 259,521 unique topic phrases. Of all topics, 212,723 (82%) only appeared once. There were also 7,879 requests (9%) for which zero topics were extracted.

Although there are many reasons why Frictionary excludes clauses, we wanted to identify the most common ones. To do this, we randomly sampled and analyzed 250 requests for which there were no topics. The first noticeable cause was that requests with no topics were short: the median number of sentence clauses per request was 3, including the request titles, which generally were sentence fragments and thus excluded. Of the 1,044 rejected clauses in this sample of 250 requests, the reasons for rejecting a clause were, in deceasing order: the subject did not have an application term (29%), the subject regarded the author and not the software (26%, corresponding to rows 2-7 of Table 1), there was no subject due to parsing errors (20%), the subject was *it* (14%), there was no verb due to parsing errors (6%), the verb was copular but with a phrasal descriptor, rather than an adjective (5%, as in *tabs are the last thing on my mind*). Of these 250 requests, all but 8 identified real problems, but typically in one or two incomplete sentences (the other 8 were spam or not English).

These metrics and user interface ideas are simply one sketch of a wide range of possible ways of browsing, viewing, and analyzing Frictionary topics. This particular form did, however, reveal a number of unexpected trends. For example, as seen in Figure 2c, the #8 topic was ***nothing happening***, which could refer to the general experience of Firefox ignoring user input. Upon further investigation, we found that the most co-occurring topics in this period and others following the release of Firefox 3.6 were ***firefox opening***, ***file downloading***, ***link clicking***, and ***button clicking***, which suggests that there were many unresolved issues with missing feedback throughout the Firefox UI. We browsed many of the requests containing this topic and found many problems with missing feedback. We also used the Frictionary UI to find the 717 bug reports written since the release of Firefox 3.6 and containing the topic ***nothing happens*** and identified them in Firefox Bugzilla database at http://bugzilla.mozilla.org. Of these, 72% are still flagged NEW or UNRESOLVED as of August 31st, 2011, 18 months after being reported.

To further assess the utility of Frictionary to a software organization, we solicited expert critiques from a *support lead* who runs *support.mozilla.org* and is responsible for gleaning user insights from support, and the Firefox principal designer, who sets interactive and visual design directions for Firefox. We contacted both by e-mail, sending a document describing how Frictionary works and providing a link to the Frictionary

Firefox dataset corpus. We asked them to address the following questions after exploring the prototype: (1) Did you discover anything you didn't know about Firefox users, Firefox use, or a particular Firefox release? (2) If Frictionary had live data, what role do you think the information would have in your own work or in the larger Mozilla community? (3) Are there ways you wish you could view or analyze the data that would be more useful to your role at Mozilla? (4) Is there information you would find more useful than the topic data presented in Frictionary? We asked each expert to provide honest judgments, even if harsh or negative.

The support lead felt that the tool was "*quite impressive*" and that the "*it could be useful to gather all of the comments about Firefox from all over the web into one place and the UI for slicing the data is cool*." Ultimately, however, he felt that the topic extraction was "*good, but not good enough*":

> A "message" could be an error message, an email message, a message box, an IM… all of which are distinct things to support… some people say "open" some say "load"… "open" could be dozens of different distinct behaviors from loading pages to starting the browser to opening downloaded files to downloading attachments from email.

The support lead struggled to find trends that he was not already aware of, but admitted that he was unlikely to, since the data was a year old. He explained that he finds trends by just replying to dozens of support requests every day: "*I just have a sense for how many lost bookmark threads (for example) to expect and then when that suddenly increases or we see new issues that I haven't seen before, I report it.*"

The Firefox principal designer was more positive:

> It was really interesting to see changes over time for the most critical features, like the application being able to install or update. Unlike crash reports, we currently don't have a good way of instrumenting and monitoring when an install or update failed, so visualizing quantitative data coming out of support requests for that feature is really valuable.

He also felt that Frictionary could help open source volunteers better prioritize their efforts:

> I think this would really help an open source community prioritize work on particular engineering challenges. Otherwise people in an open source community will naturally gravitate towards only working on the things that they personally find interesting…

He also described a chart that support creates manually that plots frequency versus severity; he believed Frictionary would be a useful way of automating the creation of this chart, allowing Frictionary users to label particular features and topics with severity ratings.

Discussion and Conclusions

Our evaluations show that most of Frictionary's topics were viewed as legitimate problems and that experts see value in the information, but that the extraction may need to be even more precise about specific software features to be useful in practice. Also, while natural language parsers are now quite accurate, their inaccuracies were behind many of Frictionary's invalid topics. Future work will need to further adapt parsers to technical, jargon-laden documents. Frictionary's own extraction also led to invalid topics. For instance, in the Firefox data set, the word ***time*** appeared in the application dictionary, but was commonly matched to the phrase *every time*. Our evaluations also found that despite Frictionary's ability to extracted valid topics from requests, the requests themselves have a relatively low information density, and the extracted topics have an even lower density.

Despite these limitations, Frictionary represents a first step in what we hope to be a new era of user experience information extraction. We hope future work will continue to explore more powerful and more accurate means of understanding not only support requests, but the wide range of other content that users create to describe their software use.

Acknowledgements
This material is based upon work supported by the National Science Foundation under Grant No. CCF-0952733.

References
[1] Akers, D., Simpson, M., Jeffries, R., Winograd, T. (2009). Undo and erase events as indicators of usability problems. *ACM CHI*, 659-668.

[2] Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., Guéhéneuc, Y. (2008). Is it a bug or an enhancement? A text-based approach to classify change requests. *CASCON*, article 23.

[3] Bernstein, M.S., Suh, B., Hong, L., Chen, J., Kairam, S., Chi, E.H. (2010). Eddi: interactive topic-based browsing of social status streams. *ACM UIST*, 303-312.

[4] Chen, K., Luesukprasert, L., Chou, S.T. (2007). Hot topic extraction based on timeline analysis and multidimensional sentence modeling. *IEEE KDE*, 19(8), 1016-1025.

[5] Chilana, P.K., Grossman,T., Fitzmaurice, G. (2011). Modern software product support processes and the usage of multimedia formats. *ACM CHI*, 3093-3102.

[6] Chilana, P.K., Ko, A.J., Wobbrock, J.O., Grossman,T., Fitzmaurice, G. (2011). Post-deployment usability: A survey of current practices. *ACM CHI*, 2243-2246.

[7] Di Lucca, D., Penta, D., Granada, S. (2002). An approach to classify software maintenance requests. *ICSM*, 93-102.

[8] Fellbaum, C. (1998). *WordNet: An electronic lexical database*. Bradford Books.

[9] Francis, P., Leon, D., Minch, M. (2004). Tree-based methods for classifying software failures. *IEEE ISSRE*, 451-462.

[10] Fourney, A., Mann, R., Terry, M. (2011). Characterizing the usability of interactive applications through query log analysis. *ACM CHI*, 1817-1826.

[11] Furnas, G.W., Landauer, T.K., Gomez, L.M., Dumais, S.T. (1987). The vocabulary problem in human-system communication. *CACM*, 30(11), 964-971.

[12] Klein, D., Manning, C.D. (2003). Accurate unlexicalized parsing. *TACL*, 423-430.

[13] Ko, A.J., Lee, M.J., Ferrari, V., Ip, S., Tran, C. (2011). A case study of post-deployment user feedback triage. *IEEE CHASE*, 1-8.

[14] Ko, A.J., Myers, B.A., Chau, D. H. (2006). A linguistic analysis of how people describe software problems in bug reports. *IEEE VL/HCC*, 127-134.

[15] Ko, A.J., Chilana P.K. (2010). How power users help and hinder open bug reporting. *ACM CHI*, 1665-1674.

[16] Naaman, M., Becker, H. Gravano, L. (2011). Hip and trendy: Characterizing emerging trends on Twitter. *JASIST*, 62(5).

[17] Rastkar, S., Murphy, G.C., Murray, G. (2010). Summarizing software artifacts: A case study of bug reports. *ACM/IEEE ICSE*, 504-514.

[18] Singh, V., Twidale, M. (2008). The confusion of crowds: non-dyadic help interactions. *ACM CSCW*, 699-702.

[19] Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J. (2008). An approach to detecting duplicate bug reports using natural language and execution information. *ACM/IEEE ICSE*, 461-470.

[20] Wu, Q., Wang, Q. (2010). Natural language processing based detection of duplicate defect patterns. *IEEE COMPSACW*, 220-225.

[21] Yatani, K., Novati, M., Trusty, A., Truong, K.N. (2011). Review Spotlight: A user interface for summarizing user-generated reviews using adjective-noun word pairs. *ACM CHI*, 1541-1550.