Facing up to the Inequality of Crowdsourced API Documentation

Hewijin Christine Jiau
Department of Electrical Engineering
National Cheng Kung University
Tainan, Taiwan
jiauhjc@mail.ncku.edu.tw

Feng-Pu Yang
Department of Electrical Engineering
National Cheng Kung University
Tainan, Taiwan
jopwil@nature.ee.ncku.edu.tw

ABSTRACT

API usability is a crucial issue in software development. One bottleneck of API usability is insufficient documentation. This study empirically confirmed the inequality of crowdsourced API documentation, which is one of the main sources of API documentation. To manage the inequality, a method for documentation reuse is proposed based on the nature of object-oriented programming language, inheritance. A case study was conducted in Stackoverflow, which is a widely used Q&A site, to study the feasibility of the documentation reuse. Results of the case study indicate that documentation reuse is feasible in improving both the coverage and quality of crowdsourced API documentations.

Categories and Subject Descriptors

D.1.5 [**Software**]: Programming Techniques - *Object-oriented Programming*, D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement - *Documentation*, G.3 [**Mathematics of Computing**]: Probability and Statistics - *Distribution functions*, H.3.1 [**Information Analysis and Retrieval**]: Content Analysis and Indexing - *Abstracting methods*.

General Terms

Documentation, Human Factors, Measurement.

Keywords

API, Crowdsourcing, Forum, GWT, Inequality, Inheritance, jQuery, Power law, Stackoverflow, Swing, SWT, wxPython.

1. INTRODUCTION

API usability is a crucial issue because of the proliferation of APIs in almost every application domain [DFSM09]. Among the factors that affect API usability, Robillard's study indicates that the critical bottleneck of API usability is the lack of diverse API documents [Rob09]. The requirement of diverse API documents usually results in varying requirements exposed by newly emergent API learning styles, namely that numerous programmers tend to learn API on demands. The demand-driven API learning styles are receiving increasing attention because of several newly emergent paradigms, such as *Opportunistic Programming* [BGL$^+$09], *Pragmatic Software Reuse* [Hol09], and *Example-centric Programming* [BDWK10].

A demand-driven API learning style is too free to be predictable; that is, it is usually not within official API documents' assumptions. API developers are also incapable of generating documents for the newly emergent paradigms because of limited resources. However, programmers of those newly emergent paradigms can leverage numerous unofficial API documents generated by API users. This unofficial documentation is also known as *crowd documentation* because it is generated from *crowdsourcing* [How06].

Because crowd documentation has gradually become one of vital resource for API users, a recent study empirically investigated if it is qualified for practical usages [PT11]. The result of the study indicates that several types of crowd documentation have sufficient coverage for practical usages. In addition to official API documentation, *blog post* and *stackoverflow*[1] are two types of crowd documentation with highest coverage ratios of 87.9% and 84.4%, respectively. Because blog post was comprehensively studied by Parnin and Treude [PT11], this study focused on investigating another crowd documentation: Stackoverflow.

Stackoverflow is a popular Q&A platform for programmers. Four characteristics of Q&A make it a suitable document for the button-up and demand-driven API learning style. The first characteristic is the accessibility. Q&A is one type of online documents that can be easily accessed by querying search engines. The second characteristic is the diversity of usage terms. Programmers who learn API in button-up style usually lack the knowledge to use right or main-stream terminologies to describe their intents. The diversity of usage terms increase the hit rate for current text-based search engines. The third characteristic is the fine granularity. Programmers who learn API on demand tend to focus on tasks at hand and minimize the time required to read irrelevant information. Other types of documents, such as tutorials, may contain more irrelevant information than Q&A. The final characteristic is that Q&A usually covers a wide-spectrum of API usage contexts. This characteristic increases the chances of locating Q&A with a similar context to programmers' tasks at hand. In other words, this characteristic increases the probability of minimizing the effort to adapt acquired information to tasks at hand.

Despite the characteristics of Q&A, it might present a problem that is widely observed in other crowdsourcing activities, that is, the *inequality* of resource allocation. A larger portion of the resource of crowdsourcing is allocated to a smaller percentage of topics. If the inequality is substantial, people who are searching for documents of less popular topics encounter considerable shortages. Understanding the current state of inequality is helpful in designing corresponding strategies to reduce the effect of inequality. However, no empirical evidences of such inequality is available in current API-related studies. In this study, we aimed to measure the degree of inequality in a real-world Q&A forum: Stackoverflow. The empirical results consistently indicate that the inequality of resource allocation in crowdsourced API documentation is severe, that is, the resource allocation is a power-law distribution that obeys the Pareto principle.

To reduce the effect of inequality on minority, we discovered an opportunity to leverage resources of majority in minority. The opportunity is based on one nature of object-oriented programming language, that is *inheritance*. In an object-oriented API, numerous duplications were observed among classes that belonged to the same inheritance hierarchy.

[1] http://stackoverflow.com/

The effectiveness of this leverage depends on the degree of the API's duplication and the resource distribution of the targeted Q&A forum. We conducted an empirical study on three APIs in stackoverlow to understand the extent of the opportunity in the real world. The result indicates that the leverage is practical for all three targeted APIs: Swing, SWT and GWT. We discovered that the coverages of three object-oriented APIs are lower than that measured in jQuery, however, coverage can be considerably improved by the leverage of resources of the majority. Based on our assumption, questions with more views have higher quality; therefore, the opportunities of quality improvement of questions were also examined. Results indicate that Swing and SWT exhibited similar ranges of improvement, at approximately 200 views to 1000 views. The GWT exhibited a relatively large range of quality improvement. However, the effectiveness of quality improvement was relatively lower in GWT because GWT has a higher mean of views. The higher mean of views indicates a lower necessity for quality improvement because the original documents are usually sufficiently qualified.

2. RESEARCH QUESTIONS

Two dependent research questions were used to investigate the inequality of crowdsourced API documentation. First, its current state was clarified to determine if it was sufficiently serious. After confirming the severity of inequality, we devised a solution for easing the effect caused by the inequality. The devised solution is based on the nature of object-oriented programming, that is, inheritance. Theoretically, the duplication among classes of the same inheritance hierarchy can be used in easing the effect of inequality by reusing crowdsource of popular classes in less popular classes. For the devised solution, the second research question was used to empirically determine the actual effect contributed by the solution. In detail, the research questions we addressed are as follows:

- RQ1: Is the inequality severe in crowdsourced API documentation?
- RQ2: What is the actual effect of our solution on easing the inequality?

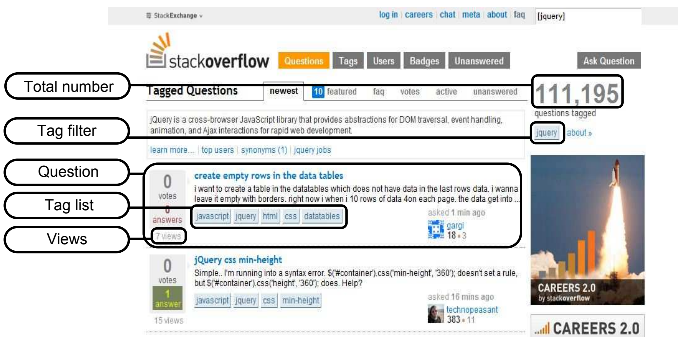
3. THE INEQUALITY OF CROWDSOURCING

This section presents an empirical study to answer RQ1: '*'Is the inequality server in crowdsourced API documentation?*" First, the methodology used for this empirical study is presented. The result is subsequently presented in graphical form to highlight the inequality. The result confirmed the existence of inequality for all investigated APIs. Finally, the discussion and threats of validity are presented.

3.1. Methodology

To answer RQ1, five APIs were stratifiedly sampled from Stackoverflow: jQuery, GWT, Swing, SWT and wxPython. Three layers were made in this stratified sampling, as follows: APIs with approximately 10^5 questions (jQuery), APIs with approximately 10^4 questions (GWT and Swing) and APIs with approximately 10^3 questions (SWT and wxPython). Stratified sampling was used to investigate whether a confounding effect on the size of crowdsourced documentation occurred on the inequality of crowdsourcing.

Figure 1 depicts five data blocks of Stackoverflow, including *total number*, *tag filter*, *question*, *tag list* and *views*. Stackoverflow provides a filtering mechanism based on its tagging system. Figure 1 presents an example of using "jquery" as the value of the filter. When using "jquery" as the value of the filter, the *total number* block presents the total number of questions tagged as "jquery." The value of the filter also presents in



jQuery is a cross-browser JavaScript library that provides abstractions for DOM traversal, event handling, animation, and Ajax interactions for rapid web development.

i want to create a table in the datatables which does not have data in the last rows data. i wanna leave it empty with borders. right now i when i 10 rows of data 4on each page. the data get into ...

Simple.. I'm running into a syntax error. $('#container').css('min-height', '360'); doesn't set a rule, but $('#container').css('height', '360'); does. Help?

Figure 1: A screen-shot of Stackoverflow.com with annotations on major blocks.

the *tag filter* block. The summary of questions tagged "jquery" presents in a list of *question* blocks. In each *question* block, tags of the question are presented in the *tag list* block and the view numbers of the question are presented in the *views* block.

We collected questions related to these five APIs. To determine questions related to a specific API, we assumed that if a question's tag list contained the tag, which is the most representative tag for the API, the question was considered relevant. In this study, the most representative tags for the five sampled APIs were "jquery," "gwt," "swing," "swt," and "wxpython". We investigate distributions of *views* for relevant questions of each API. The value of *views* is proportional to the popularity; that is, the more views a question has the more popular the question. We selected popularity of a question as an indicator for the amount of crowdsourcing allocated to the question.

3.2. Results

Figure 2 consists of three sub-diagrams for three layers of sampling. For each sub-diagram, the x-axis represents the rank of questions and the y-axis represents the popularity of questions. All three distributions are not uniform and their sharp skewness indicates the existence of inequality. Because the inequality of popularity, which is the indicator of allocated crowdsource, is consistent for all three layers, two conclusions were derived as follows:

1. The inequality is severe in the context or crowdsourced API documentations
2. The inequality is independent of the size of documentation.

We further tested if these crowdsourced API documents obeyed the Pareto principle, also known as the 80:20 rule. The most intuitive manner to test if a distribution obeyed the Pareto principle is to draw it in log-log scale. If the log-log plot of a dataset converges to a straight line, the dataset has a power law distribution, which is also known as Pareto distribution. Figure 3 presents question popularity versus question rank of five APIs in log-log scale. As shown in Figure 3, all five APIs converge to straight lines.

3.3. Discussion

As depicted in Figure 4, Andjam generated a log-log graph of page popularity versus page rank in several wikis. The trend of log-log graphs of these wikis are similar to the log-log graphs generated in this study. Because wikis are also crowdsourced documents, the similarity may imply that inequality occurs universally in any crowdsourced documents. Although the solution presented in this paper is limited to object-oriented

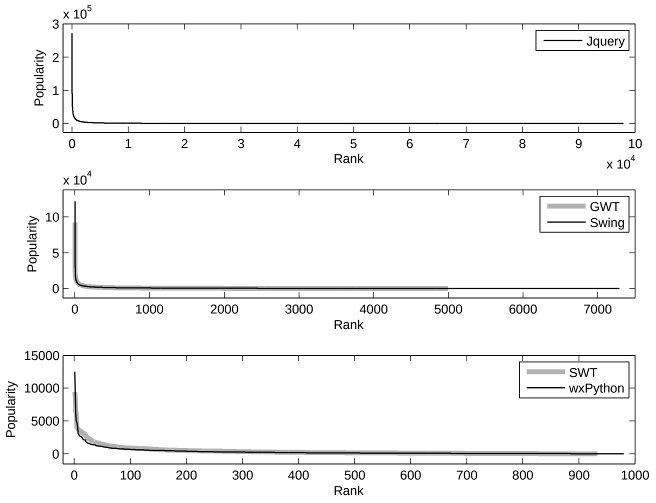


Figure 2: The question's popularity distributions.

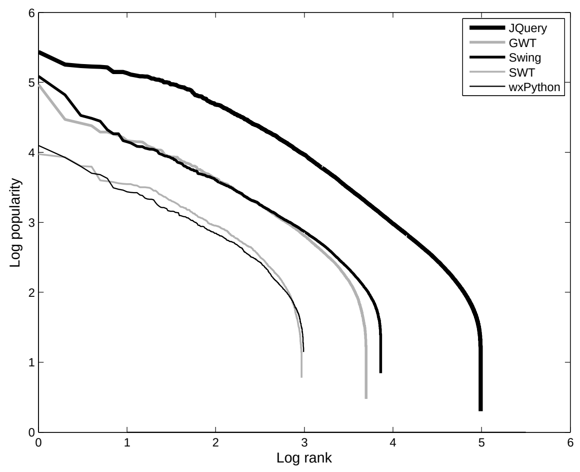


Figure 3: The question's popularity distributions in log-log scale.

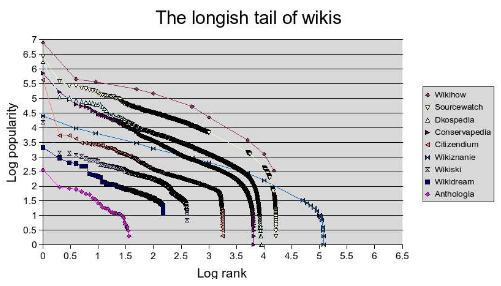


Figure 4: A log-log graph of page popularity (total hits ever) versus page rank in several wikis.
Credit: Andjam (2007)/ public domain

APIs, solutions for the inequality of other crowdsourced documents may be derived similarly. For example, Wikipedia used "category page" to interconnect individual pages of the same category. Users can reuse category-level information by following those "category pages" to more popular individual pages.

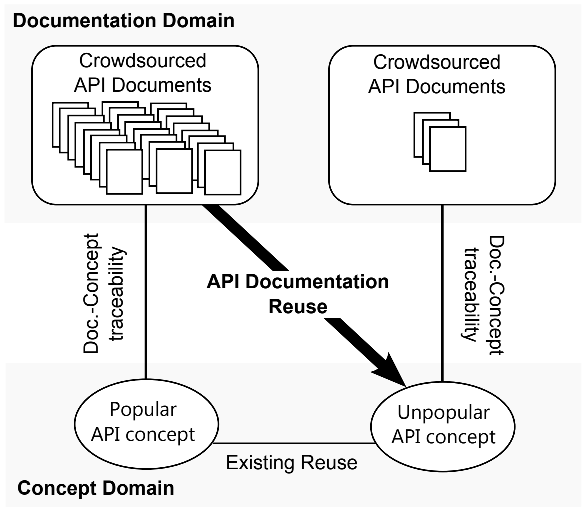


Figure 5: Reusing crowdsourced API documentation based on existing reuse of API concepts.

4. Process for Exploring Opportunity of Crowdsource Reuse

Because inequality is an inevitable result of crowdsourcing, it should be leveraged instead of changed. A method to leverage inequality is to reuse resources of popular documents in unpopular documents. However, reuse of crowdsourced documentation is usually difficult because it is usually written in natural language, which lacks machine-understandable semantics. Instead of reusing crowdsourced documentation directly, we projected them into a concept domain and then reused them based on existing reusability of the concept domain.

Figure 5 illustrates the main concept of crowdsource reuse. Based on the doc.-concept traceability from a documentation domain to a concept domain, a set of crowdsourced API documents is mapped to an API concept. If the inequality exists among concepts of the concept domain, a popular API concept is mapped to a relatively large set of documents than the set of documents mapped to an unpopular API concept. We assume that if an existing reuse between one popular API concept and one unpopular API concept occurs, an opportunity for "API documentation reuse" can be obtained between the documents of the popular API concept and the unpopular API concept.

The second research question (RQ2) of this work was *What is the actual effect of crowdsource reuse on easing inequality?* The power of crowdsource reuse depends on the effectiveness of the selected concept domain. To determine the effectiveness of a selected concept domain, RQ2 was divided into four checkpoints, as follows:

- CP1: Is the reusability of the concept domain high?
- CP2: Is the effort for recovering documentation-concept traceability low?
- CP3: Is inequality of resource allocation found among concepts?
- CP4: Is the effect of documentation reuse considerable?

We discovered an effective concept domain known as "inheritance hierarchy" in the context of object-oriented APIs. Inheritance hierarchy is a concept domain with high reusability. Two main reasons for using inheritance are "code reuse" and "concept reuse" [Bud01]. For code reuse, a derived class can reuse the source-codes written in its super class. Code

TABLE 1: Lists of Selected Widgets

API	Selected Widgets
Swing *6.0*	JButton, JCheckBox, JCheckBoxMenuItem, JColorChooser, JComboBox, JDesktopPane, JDialog, JEditorPane, JFileChooser, JFormattedTextField, JFrame, JInternalFrame, JLabel, JMenu, JMenuItem, JMenuBar, JOptionPane, JPanel, JPasswordField, JPopupMenu, JProgressBar, JRadioButton, JRadioButtonMenuItem, JRootPane, JRadioButtonMenuItem, JRootPane, JScrollBar, JScrollPane, JSeparator, JSlider, JSpinner, JSplitPane, JTabbedPane, JTable, JTextArea, JTextField, JTextPane, JToggleButton, JToolBar, JToolTip, JTree, JViewport, JWindow
GWT *2.3*	Button, PushButton, RadioButton, CheckBox, DatePicker, ToggleButton, TextBox, PasswordTextBox, TextArea, Hyperlink, ListBox, CellList, MenuBar, Tree, CellTree, SuggestBox, RichTextArea, FlexTable, Grid, CellTable, CellBrowser, TabBar, DialogBox, PopupPanel, StackPanel, StackLayoutPanel, HorizontalPanel, VerticalPanel, FlowPanel, VerticalSplitPanel, HorizontalSplitPanel, SplitLayoutPanel, DockPanel, DockLayoutPanel, TabPanel, TabLayoutPanel, DisclosurePanel
SWT *Galileo*	Button, Canvas, Combo, Composite, CoolBar, DateTime, ExpandBar, Group, Label, Link, List, Menu, ProgressBar, Sash, Shell, Slider, Scale, Spinner, TabFolder, Table, Text, ToolBar, Tree

reuse can improve productivity in implementation phase and reduce the bad smell of code duplication. For concept reuse, a derived class overrides implementations of methods inherited from its super class; however, it reuses the definition of inherited methods. Concept reuse ensures a common protocol shared among all derived classes of a specific super class. The common protocol among derived classes can reduce unnecessary conditional complexity by the support of polymorphism.

In the next section, we present a case study to empirically prove that "inheritance hierarchy" is an effective concept domain for crowdsource reuse.

5. CASE STUDY: CROWDSOURCE REUSE IN OBJECT-ORIENTED API DOCUMENTATION

To understand the effectiveness of "inheritance hierarchy" as the concept domain for crowdsource reuse, we investigated three object-oriented widget-toolkit APIs: Swing, GWT and SWT. Instead of investigating all classes of the APIs, we focused only on widget classes. Widget classes can directly map to visual GUI elements, which are extensively used in GUI prototyping and are usually API-independent. For programmers who tend to learn APIs on demand, they usually seek crowdsourced documents for widget classes required by their GUI prototyping. Table 1 lists all the widget classes investigated in this case study.

For documentation domain, we collected API-specific crowdsourced documents from Stackoverflow. A total of 7,292, 4,995 and 933 questions were collected on August 2, 2011, for Swing, GWT and SWT, respectively. Each question contained one question entry and zero-to-many answer entries. A total of 22,910, 13,304 and 2,362 entries were collected for Swing, GWT and SWT, respectively. For concept domain, we collected data from the official API reference documentation. The versions of selected APIs were 6.0, 2.3, and Galileo for Swing, GWT, and SWT, respectively.

Figure 6 presents two data models used for preserving data extracted from documentation domain and concept domain. Data related to a question were stored in *Question*. Each question had a unique id, that is, *Question.qid*. The number of views that belonged to a question was pre-

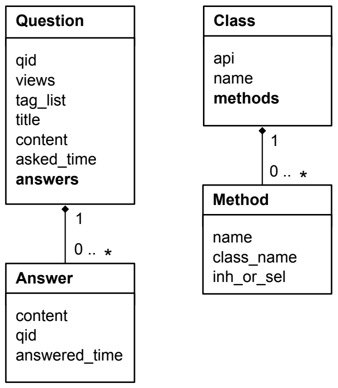


Figure 6: Data models used in this case study: documentation domain (left) and concept domain (right).

served in its *Question.view*. A list of tags that were tagged to a question was preserved in its *Question.tag_list*. Every question in Stackoverflow had a title, that is, *Question.title*, for briefly describing the intent of the asker. The details of a question were preserved in its *Question.content*. The time a question was asked was preserved in *Question.asked_time*. A question contains zero-to-many answers, which are preserved in *Question.answers*. Three attributes of an answer are retained in *Answer*, as follows: *Answer.qid*, *Answer.content*, and *Answer.answered_time*, where the *Answer.qid* is a reversed index for an answer to locate the question that it has addressed. Data related to a class was stored in *Class*. The API that a class belonged to was stored in *Class.api*. The name of a class was stored in *Class.name*. Because a class might contain zero-to-many methods, these methods were stored in a collection called *Class.methods*. Data related to a method are stored in *Mehtod*. The name of a method is preserved in *Method.name*. Parameters are not contained as part of *Method.name*. For simplicity, we treated all overloading methods as the same method, and retained only one copy in the data model. Each method belonged to one class, which was stored in *Method.class_name*. Two types of methods: inherited methods and self methods were used. The information of method type was stored in *Method.inh_or_sel*.

5.1. CP1: Reusability of Inheritance

The occurrence of reuse is determined by two factors: the frequency of recurrent problems and the availability of corresponding solutions. Based on these two factors, we designed two indicators to estimate the the occurrence of documentation reuse for a specific API. In the context of Stackoverflow, the occurrence of problems can be indicated by either asking a question or viewing an existing question. The corresponding solutions can be mapped to existing answers of the asked or the viewed question. Based on probability theory, we define an event space of problems to estimate the frequency of recurrent problems and the availability of corresponding solutions through two metrics, that is, portion of recurrent problems $\mathbf{P}_{rp}$ and redundant ratio $\mathbf{r}$. Before introducing these two metrics, several definitions are presented with an example illustrated in Figure 7. Finally, the estimating results of Swing, GWT, and SWT measured by these two metrics are presented in Table 2.

Definition 1 (Scope of Reuse $\mathbf{S}_r$). *Scope of Reuse $\mathbf{S}_r$ is a set of classes selected to define the scope of API documentation reuse.*

Definition 2 (Atomic Problem-Asking Event **p**). *Any problem on API usage can be divided into one or several atomic problems, which are related only to one method. The event of asking an atomic problem is called atomic problem-asking event **p**.*

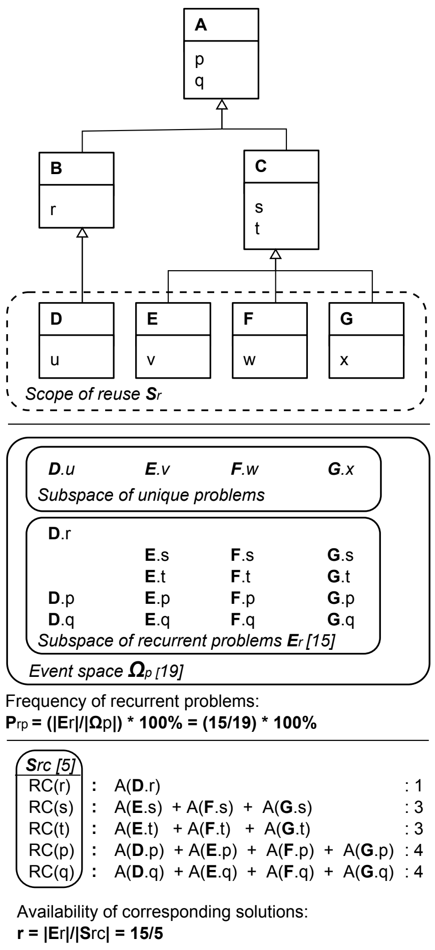

A
p
q
B
r
C
s
t
D
u
E
v
F
w
G
x
Scope of reuse Sr
D.u E.v F.w G.x
Subspace of unique problems
D.r
E.s F.s G.s
E.t F.t G.t
D.p E.p F.p G.p
D.q E.q F.q G.q
Subspace of recurrent problems Er [15]
Event space Ωp [19]
Frequency of recurrent problems:
Prp = (|Er|/|Ωp|) * 100% = (15/19) * 100%
Src [5]
RC(r) : A(D.r) : 1
RC(s) : A(E.s) + A(F.s) + A(G.s) : 3
RC(t) : A(E.t) + A(F.t) + A(G.t) : 3
RC(p) : A(D.p) + A(E.p) + A(F.p) + A(G.p) : 4
RC(q) : A(D.q) + A(E.q) + A(F.q) + A(G.q) : 4
Availability of corresponding solutions:
r = |Er|/|Src| = 15/5

Figure 7: An example of estimating crowdsourced API documentation's reusability .

Definition 3 (Event Space of Problems Ω_p). *Event Space of Problems Ω_p is a set of all possible values of **p** within a specific $\boldsymbol{S}_r$. In the context of object-oriented API, Ω_p is the set of all the methods in $\boldsymbol{S}_r$.*

$$\Omega_p = \bigcup_{c \in S_r} c.methods$$

Definition 4 (Subspace of Recurrent Problems E_r). *Subspace of Recurrent Problems E_r is a set of atomic problem-asking events, which can be reused because of their similarity. In the context of object-oriented API, E_r is the set of all inherited-methods in $\boldsymbol{S}_r$.*

$$E_r = \{m | (m \in \Omega_p) \wedge (m.inh\_or\_sel = inh)\}$$

Definition 5 (Set of reuse categories $\mathbf{S}_{rc}$). *Problems in E_r can be categorized into reuse categories based on similarity. Documents for problems of the same reuse category can be reused for each other. Set of reuse categories $\boldsymbol{S}_{rc}$ is a set of all reuse categories in E_r.*

$$S_{rc} = \{m.name | (m \in \Omega_p) \wedge (m.inh\_or\_sel = inh)\}$$

Metric 1 (Portion of recurrent problems $\mathbf{P}_{rp}$). *$\boldsymbol{P}_{rp}$ is used to estimate the actual frequency of recurrent problems. In the context of object-oriented API, $\boldsymbol{P}_{rp}$ is the ratio of the size of E_r versus the size of Ω_p.*

$$\boldsymbol{P}_{rp} = (|E_r|/|\Omega_p| * 100)\%$$

Metric 2 (Redundant ratio **r**). *Redundant ratio **r** is used to estimate the availability of solutions corresponding to problems of a specific API. In the context of object-oriented API, $\boldsymbol{P}_{rp}$ is the ratio of the size of $\boldsymbol{E}_r$ versus the size of S_{rc}.*

TABLE 2: Estimations of API reusability

API	$\|\Omega_p\|$	$\|\mathbf{E}_r\|$	$\|\mathbf{S}_{rc}\|$	$\mathbf{P}_{rp}$	**r**
Swing	14,376	13,064	559	90.87%	23.3703
GWT	3,446	2829	297	82.09%	9.5252
SWT	2,174	1824	91	83.90%	20.0440

$$\boldsymbol{r} = |E_r|/|S_{rc}|$$

An example illustrated in Figure 7 shows the use of these two metrics. Figure 7 consists of three sections: an inheritance hierarchy, the estimation of $\mathbf{P}_{rp}$, and the estimation of r (from top to bottom). This inheritance hierarchy consists of seven classes with inheritance depth 3. Four classes (D, E, F and G) are selected as the *scope of reuse* $\mathbf{S}_r$. Within the scope of reuse, the *event space of problems* Ω_p consists of 19 *atomic problem-asking events*. Among 19 events of Ω_p, 15 events are related to inherited methods and these 15 events define the *subspace of recurrent problems* $\mathbf{E}_r$. The frequency of recurrent problems can subsequently be estimated by $\mathbf{P}_{rp}$, (15/19) * 100%. Five reuse categories are presented in the bottom of Figure 7. The availability of corresponding solutions can subsequently be estimated by **r**, 15/5.

Table 2 lists estimations of three selected APIs. The estimations of frequencies ($\mathbf{P}_{rp}$) of recurrent problems are considerably high, with over eighty 80% chances of being measured for all three APIs. This indicates that programmers ask questions with high frequency, which can leverage reusable solutions. The GWT has a relatively low value for the estimations of availability (**r**) of solutions. A possible reason for the relatively low value of GWT is that GWT is designed for Web applications, which contain a number of web-specific widgets that have relatively low similarity with other simulated desktop widgets. Because Swing and SWT are pure desktop widgets APIs, their availability of solutions are considerably high. Although GWT has a relatively lower availability of reusable solutions, it has over nine reusable sources of solutions for a specific question.

5.2. CP2: Recovery of Traceability

Checkpoint CP2 is used to assess if the effort of recovering traceability from crowdsourced documentations to selected concept domain is too high to be affordable. In this case study, the selected concept domain was the inheritance hierarchy. Two main types of semantics defined in inheritance hierarchy are *classes* and *methods*. To recover the traceability, semantics behind the plain-texts of documentation must be recognized, and these plain-texts must be annotated with semantic tags. The recognition of semantics is not a difficult task because words used in API naming usually differ from words of natural language. However, there are a number of exceptional cases of method names must be treated differently. In this case study, all three APIs used camelCasing in method naming, where the camelCasing convention [CA05] capitalizes the first character of each word, except the first word. CamelCasing method names are usually easily recognizable in plain-text documentation. However, when a method name consists of one word, such as "add" or "open," it often causes ambiguity in semantics recognition. To sacrifice recall for precision in recovering traceability, we excluded methods named by one word. To exclude ambiguous method naming, we measure the lengths of camelCasing method names by a predicate len_{cc}, defined in Lemma 1. Table 3 lists the coverage of methods with the len_{cc} values, which are higher than 2. All three coverages were higher than or approximately 90%, which indicates the effect of scarifying recall was small in this case study.

TABLE 3: Coverages of methods with len_{cc} values > 1.

API	$\lvert\{\text{m} \mid \text{len}_{cc}(\text{m}) > 1\}\rvert$	coverage
Swing	13049	90.77%
GWT	3302	95.82%
SWT	1950	89.70%

TABLE 4: The numbers of class-name and method-name found in question titles.

API	class-name found	method-name found
Swing	3,128	268
GWT	6,75	57
SWT	4,65	14

Lemma 1 (len_{cc}.)**.** *The convention of camelCase is to capitalizes the first character of each word, except the first word. Based on this convention, the number of words of a camelCasing term can be calculated.*

len_{cc}(camelCasing term) = number of words contained in the camelCasing term.

In addition to the relation between name length and ambiguity in the method naming, another observation is the concepts distribution in question title. We discovered that class names are mentioned more frequently than methods in question title. Table 4 shows the numbers of questions that mentioned class-names and the questions that mentioned method-names for three studied APIs. The askers asked more questions with class-name than questions with method-name. Based on the six barriers of API learning [KMA04], the relatively low frequency of using method-name in question titles may indicate that, in the context of learning API, the *use barriers* usually occur in class level and the *selection barriers* usually occur in method level. Because askers of questions may have difficulties in the usage of classes, they may explicitly mention class-names in their questions. Because askers may have difficulty in selecting methods of a specific classes, they may not mention any method-name in their questions.

Although the traceability in method-level is not easily discovered in a specific question because of selection barriers, it can be obtained in the answers of the question. Answers of a question with selection barriers usually provide suggestions of method selection. We decided to recover method-level traceability based on suggestions of answers on method selection. Two heuristics were used in this case study to recover traceability in both class level and method level.

Heuristic 1 (Class-level Traceability)**.** *With a Question **q** and a Class **c**, a class-level traceability is recovered if the following condition is true:*

- *The name of **c** is discovered in the title of **q**.*

Heuristic 2 (Method-level Traceability)**.** *With a Question **q** and a Class **c**, a method-level traceability is recovered if the following conditions are true:*

- *A class-level traceability is recovered between **c** and **q**.*

- *$\exists$ **m** $\in$ **c**.methods: (**m**.name is found in **a**, where **a** $\in$ **q**.answers) $\wedge$ (len_{cc}(**m**.name) > 1).*

Based on Heuristic 1 and Heuristic 2, we used Algorithm 1, which is dependent on Algorithm 2, to recover traceability. The time consumptions of Algorithm 1 were 60.70, 13.10 and 1.13 seconds for Swing,

```
Input: $Q_{SO}$, $\Omega_p$, api
Output: $\mathbf{A}_C$, $\mathbf{A}_M$
Use: MethodAnnotate(c,q) return subset of $\mathbf{A}_M$
foreach $q \in Q_{SO}$ do
    if $api \in q.tag\_list$ then
        foreach $c \in \Omega_p$ do
            if $q.title$ contains $c.name$ then
                push $\langle$c.name, q.qid$\rangle$ onto $\mathbf{A}_C$
                $\mathbf{A}_M \leftarrow \mathbf{A}_M \cup$ MethodAnnotate(c, q)
            end
        end
    end
end
```

Algorithm 1: The Annotate Procedure

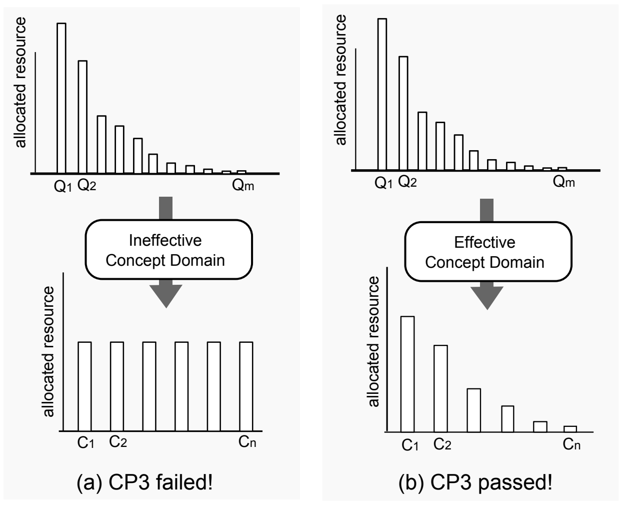


Figure 8: CP3 is used in testing the effectiveness of selected concept domains.

GWT and SWT, respectively, where the values were measured in a computer equipped with Q9550@2.38GHz CPU, 8G ram, and Windows 7 64-bits operating system. Because Algorithm 1 must be computed only once for each API, the measured time-consumptions imply that the effort of traceability recovery in this case study is affordable.

```
Input: c, q
Output: $A_m$
foreach $m \in c.methods$ do
    if $len_{cc}(m.name) > 0$ then
        foreach $a \in q.answers$ do
            if $a.contents$ contains $m.name$ then
                push $\langle$c.name, q.qid, m.name$\rangle$ onto $A_m$
            end
        end
    end
end
```

Algorithm 2: The MethodAnnotate Sub-Procedure

5.3. CP3: Resource Allocation among Classes

CP3 can be assessed after recovering traceability. Figure 8 depicts two possible results of the assessment of CP3. The effectiveness of a selected

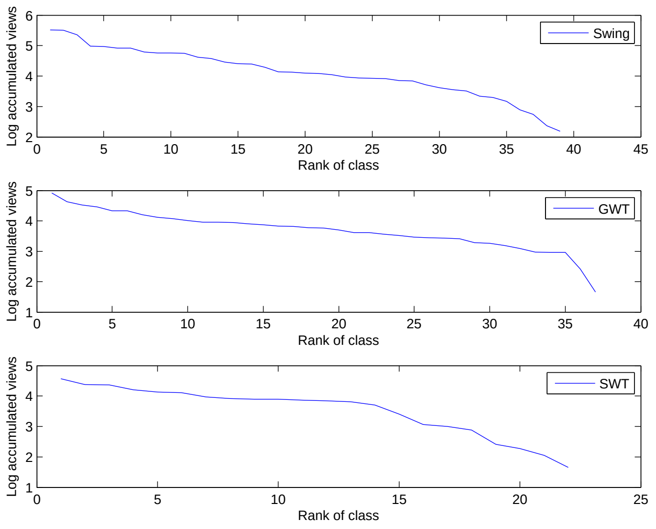


Figure 9: The lin-log diagrams for accumulated views of Swing, GWT and SWT.

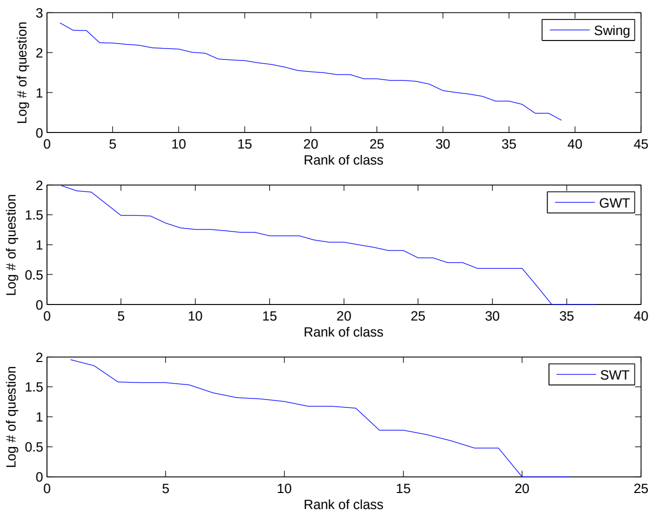


Figure 10: The lin-log diagrams for question numbers of Swing, GWT and SWT.

concept domain is assessed by the distribution of allocated resource in the concept domain. If inequality is discovered in the distribution, the selected concept domain is effective. In this case study, we assessed if inequality occurred in the distribution of allocated resources among classes. Two indicators were used to estimate resource allocation for a specific class: the accumulated views of questions related to the class and the number of questions related to the class. The questions related to a specific class were traced by the class-level traceability recovered in CP2. In this case study, resource allocation was nearly exponentially decayed from popular class to unpopular class. If a distribution is exponentially decayed, the lin-log diagram would converge to a straight line. Figure 9 shows lin-log diagrams for the accumulated views of Swing, GWT and SWT. Figure 10 shows lin-log diagrams for the question numbers of Swing, GWT and SWT. As shown in Figure 9 and 10, inequality was consistently for both indicators of resource allocation. This indicates that the inheritance hierarchy is an effective concept domain for crowdsourced API documentation reuse on three selected APIs.

TABLE 5: Coverage of crowdsourced API documentation estimated from Google's perspective.

API	# of Queries	Hit	Mean	Median
Swing	12,868	29.11%	5.7064	6
GWT	3,301	59.19%	5.2984	5
SWT	1,950	37.64%	6.2793	7

TABLE 6: Coverage of reusable documentation estimated from Google's perspective.

API	Sel	Inh	ReusedInh	Δ_{inh}
Swing	33.64%	28.60%	79.19%	276%
GWT	65.20%	57.93%	88.07%	152%
SWT	38.55%	37.45%	97.53%	260%

5.4. CP4: Effect of Documentation Reuse

In CP3, inequality among classes of three inheritance hierarchies was confirmed. In practical usage, two factors are crucial for users of API documentation, as follows: coverage and quality. In CP4, we assess the extend of documentation reuse on increasing coverage and quality.

Parnin and Treude studied the coverage of crowdsourced API documentation on jQuery [PT11]. However, jQuery is often regarded as a functional paradigm API; therefore, its result should not be directly generalized to object-oriented paradigm APIs. To confirm the coverage in the context of object-oriented APIs, we replicated the study of coverage with a number of object-oriented API specific modifications. In the original coverage study, Parnin and Treude generated queries in the form of "jQuery [method name]," that is, "jQuery add" for .add() method. In the context of object-oriented APIs, we modified the format of query for "[API name] [Method name] [Class name]," that is, "Swing JTextArea mouseDrag" for JTextArea.mouseDrag(). To reduce the ambiguity caused by single-word method names, we used len$_{cc}$ to exclude single-word method names. The generated queries were issued to Google, and the first 10 results of each query were analyzed. Table 5 presents the analyzed results of three object-oriented APIs. In the original study of jQuery, the coverage of Stackoverflow was 84.4%, which qualified for practical usage. Although the median of rank was similar to that of the original study, the coverages of object-oriented crowdsourced API documents were considerably less than that in the original study. We further calculated the extent to which documentation reuse can improve coverage. Table 6 demonstrated that self-method (sel) has higher coverage than inherited-method (inh). After introducing documentation reuse, the coverage of inherited-method can be improved from 29%-59% to 79%-97%.

Table 7 presents the values of coverage recovered based on traceability. The coverages measured by traceability recovery are less than

TABLE 7: Coverage of crowdsourced API documentation estimated from recovered traceability.

API	All	Sel	Inh	ReusedInh	Δ_{inh}
Swing	14.40%	28.19%	12.18%	56.07%	460%
GWT	10.36%	15.37%	8.16%	34.04%	417%
SWT	17.48%	24.72%	15.31%	79.01%	516%

Google-based measures. The noise generated by sidebar is a reason Google-based coverage estimation is higher than coverage measured by recovered traceability. The sidebar of Stackoverflow pulls several relevant data, such as related questions and relevant programming job-offers. In addition to noises, Google also use mechanisms to relax query constraints when no results have been exactly matched. Constraint relaxation also causes over-estimation on the average of crowdsourced API documentation. However, improvements contributed by reuse are consistently high for three APIs. A coverage improvement over 400% was measured in documents.

In addition to coverage, the quality of question is another factor for usability of documents. Instead of measuring a variety of quality attributes, such as correctness, clarity, and readability, we used views of a question to estimate the quality of a question. We assume that the more views a question has the higher the quality of the question. Because Stackoverflow is a wiki-based Q&A forum, each user can improve the quality of questions by contributing new contents or revising existing documents.

To understand the extent of quality improvement contributed by documentation reuse, we assessed the view distribution for questions of each reuse category. A question belonged to a reuse category if it had method-level traceability to the reuse category.

We examined the mean and standard deviation of views distributed in each reuse category. Figure 11 presents the mean value of views for each reuse category. As shown in Figure 11, GWT has the highest mean of views. Higher mean value of GWT indicates that the average quality is high. This may imply a relatively lower effect of quality improvement. Quality improvement versus view increment gradually reduces as the views increases. Figure 12 presents standard deviation of views for each reuse category. High standard deviation indicates a potential high gain of views in documentation reuse. Figure 12 demonstrated that GWT has largest variation of standard deviation among reuse categories. This indicates that the gain of quality improvement is not consistent for all reuse categories of GWT. Only some reuse categories have high variations in their view distributions. To visualize the gain of each reuse category, we use a metric “views delta” to indicate the potential quality gain. “Views delta” is defined by the delta between upper quartile and lower quartile for each reuse category. Figure 13 presents views delta of each reuse category. For Swing and SWT, the “views delta” of reuse categories occur almost between 1000 and 200. This indicates that the gains in quality are usually above 200, but only few reuse categories can have gains higher than 1000. In contrast to Swing and SWT, GWT exhibited higher variation in quality gain. Fifty percent of reuse categories fell between 1000 and 2000. Approximate 50% of categories have higher than 1000 gains of quality. Although the GWT had higher mean of views, it also had a higher views delta. The higher views delta may imply the quality improvement in GWT was considerable significant for almost reuse categories.

6. RELATED WORK

Parnin and Treude measured a number of crowdsourced API documents [PT11]. They focused on the document type of blog and the API of jQuery. In this study, we focused on the document type of Q&A forum, specifically StackOverflow, and three APIs of object-oriented GUI toolkits. We discovered a universal inequality in crowdsourced documents. Inequality causes several disadvantages for crowdsourced API documentation, including coverage and quality. For coverage, Parnin and Treude discovered that Stackoverflow has a high coverage in jQuery. They also indicated that various APIs must be studied to generalize the result. We discovered that the coverage is low in object-oriented APIs. This implies that the paradigms of APIs may be one factor that affected coverage of crowdsourced API documents.

Stackoverflow is one of most successful Q&A forums in this decade. A number of innovate concepts, such as badged systems lent from game design or wiki-style editing systems, enables superior performance of Stackoverflow in several quality attributes, for example, shorter response time, than other forums [LZH10]. L. Mamykina et al. extensively study general properties of Stackoverflow [MHMM11]. Instead of studying

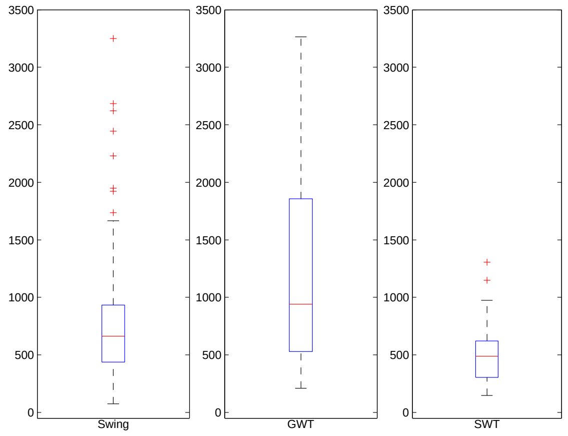


Figure 11: The box plots of view’s mean value for each reuse category in Swing, GWT and SWT.

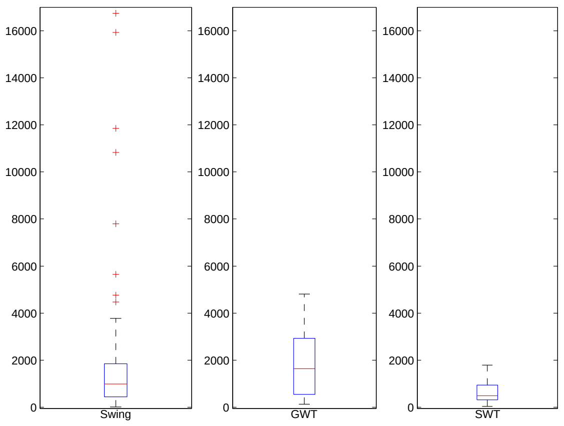


Figure 12: The box plots of view’s standard deviation for each reuse category in Swing, GWT and SWT.

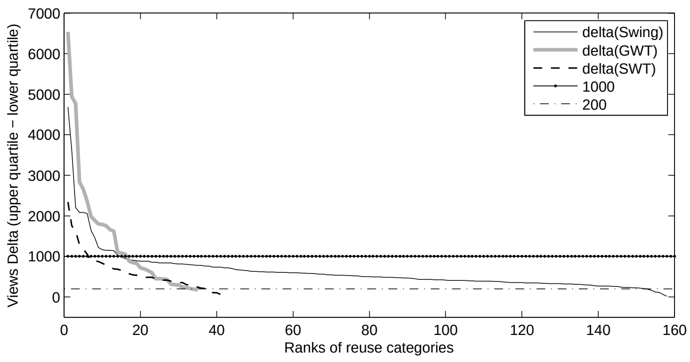


Figure 13: The box plots of view’s stand deviation for each reuse category in Swing, GWT and SWT.

general properties, we investigate Stackoverflow from the view-point of specific APIs. A number of API-specific quality attributes can be studied based on the views of APIs, such as coverage.

G-Finder is a work that is focused on the resource inequality [LZH10]. G-Finder uses a novel method to route relevant questions to experts to reduce the response time. In contrast to G-Finder, which uses more intrusive method to reduce the inequality, we provide a non-intrusive method by the leverage resource of majority. These two types of methods can be used in conjunction to manage the inevitable trend of inequality of crowdsourcing.

7. CONCLUSION AND FUTURE WORK

In this work, we empirically confirmed the inequality of crowdsourced API documentation. The inequality decreases the API's usability in several aspects, such as coverage and quality. To manage the inequality, we discovered an opportunity for documentation reuse based on the nature of object-oriented programming language. Documentation can be reused among methods of various classes, because those methods are all inherited from the same parent class in either terms of code-reuse or concept reuse. An empirical study on three APIs, Swing, SWT and GWT, has confirmed the feasibility of documentation reuse.

In the future work, we may explore and validate more concept system for finding more opportunities of documentation reuse. We should also generalize our empirical result to APIs of other application domains and different paradigms.

ACKNOWLEDGMENTS

The authors thank the support from National Science Council, Taiwan (NSC 98-2221-E-006-099-MY2).

REFERENCES

[BDWK10] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. Example-centric Programming: Integrating Web Search into the Development Environment. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 513–522. ACM, 2010.

[BGL$^+$09] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. Opportunistic Programming: Writing Code to Prototype, Ideate, and Discover. *IEEE Software*, 26(5):18, 2009.

[Bud01] Timothy Budd. *"An Introduction to Object - oriented Programming"*, pages 162–163. Addison Wesley, 3rd edition, 2001.

[CA05] Krzysztof Cwalina and Brad Abrams. *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries*. Addison-Wesley Professional, 2005.

[DFSM09] John M. Daughtry, Umer Farooq, Jeffrey Stylos, and Brad A. Myers. API Usability: CHI'2009 Special Interest Group Meeting. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems*, pages 2771–2774, 2009.

[Hol09] Reid Holmes. *Pragmatic Software Reuse*. PhD thesis, Calgary, Alta., Canada, 2009.

[How06] Jeff Howe. The Rise of Crowdsourcing. *Wired*, 14(6), 2006.

[KMA04] Andrew J. Ko, Brad A. Myers, and Htet Htet Aung. Six Learning Barriers in End-User Programming Systems. In *IEEE Symposium on Visual Languages and Human Centric Computing*, pages 199–206, September 2004.

[LZH10] Wei Li, Charles Zhang, and Songlin Hu. G-Finder: Routing Programming Questions Closer to the Experts. In *OOPSLA/SPLASH*, October 2010.

[MHMM11] L. Mamykina, B. Hartmann, B. Manoim, and M. Mittal. Design Lessons from the Fastest Q&A Site in the West. In *Proceeding of the 29th Conference on Human Factors in Computing Systems*, 2011.

[PT11] Chris Parnin and Christoph Treude. Measuring API Documentation on the Web. In *Proceeding of the 2nd International Workshop on Web 2.0 for Software Engineering*, pages 25–30, May 2011.

[Rob09] Martin P. Robillard. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software*, 26(6):27–34, November/December 2009.