What Makes a Good Code Example?

A Study of Programming Q&A in StackOverflow

Seyed Mehdi Nasehi, Jonathan Sillito , Frank Maurer, and Chris Burns
Department of Computer Science
University of Calgary
Calgary, AB, Canada
{smnasehi, sillito, frank.maurer, chris.burns}@ucalgary.ca

***Abstract*— Programmers learning how to use an API or a programming language often rely on code examples to support their learning activities. However, what makes for an effective code example remains an open question. Finding the characteristics of the effective examples is essential in improving the appropriateness of these learning aids. To help answer this question we have conducted a qualitative analysis of the questions and answers posted to a programming Q&A web site called StackOverflow. On StackOverflow answers can be voted on, indicating which answers were found helpful by users of the site. By analyzing these well-received answers we identified characteristics of effective examples. We found that the explanations accompanying examples are as important as the examples themselves. Our findings have implications for the way the API documentation and example set should be developed and evolved as well as the design of the tools assisting the development of these materials.**

Keywords- code example; documentation; API; social learning

I. INTRODUCTION

Programming is a non-ending problem-solving adventure. Whether a bug needs to be fixed, a new feature needs to be added, the design of some working code needs to be enhanced, or a legacy system needs to be changed, programmers are seeking different sources of information to get some advice on how to accomplish these tasks. Code examples can come to their help and lift the burden of solving problems on their own. It is well-known in educational psychology that worked examples (i.e. solved problems) are more efficient means of learning new materials than trying to solve problems from scratch [21]. Consequently, the developer burden can be reduced if an example that solves their problem is found, so that they can just (re)use it.

Developers rely on different sources of knowledge to find answers to their problems. During the maintenance phase of software life cycle, code is the main source of knowledge about the system. However, maintainers are eager to find somebody to talk to and to help them understand that code [25]. In other words, fellow developers and their expertise are an invaluable source of knowledge for developers and maintainers. Even if they are not familiar with the system under maintenance, they can transfer their past experience to the information seekers.

There are other sources to find answers containing code, besides other developers. A questioner can refer to documentations, books, tutorials, etc., or search online resources available on the web. The first step in finding answers is to formulate a question. It can be articulated in natural language when the audience is a human being, or has to be expressed in some kind of query language to retrieve relevant examples. However, even if a developer is able to find a relevant example, then they are still bound by the quality of that example. In other words, finding a relevant example is necessary but not sufficient to ease the developer's task; however, finding good examples can do the trick.

But *what are the characteristics of a good example*? This is the main question this paper tries to address. We wanted to find out what kind of code examples actually helps developers and maintainers solve their problems and what attributes distinguish them from not-so-helpful examples.

To answer these questions we needed a source of good examples. We turned to the online programming Q&A web site (StackOverflow, hereby SO) to gather such information. SO is a place for developers to post their programming questions and for fellow developers to provide answers. The questioner can add a few tags to the question to help others (e.g. potential responders) find out about what the question is about. The questioner can select one answer as the most helpful one (called the accepted answer). Site members can vote on questions and answers. The positive and negative votes (called upvote and downvote respectively) show how helpful that question/answer was for the audience. The difference between the number of up/downvotes determines the score of a question/answer. Each site member has a reputation determined by a reputation score. As they participate in different activities on SO, by posting questions or answers, voting on them, posting comments, etc. their reputation score would increase and a greater reputation value means more capabilities for a member. For instance, they can edit questions/answers or can close a Q&A thread.

The interactive nature of SO makes it possible for both questioners and responders to clarify the vagueness in a question/answer. Even other site members can edit questions/answers as they see it fit. SO has the three properties of new social learning technologies [28]: It

978-1-4673-2312-3/12/$31.00 ©2012 IEEE

supports learners to find the right content using natural language, not just relying on keywords for finding the content; it supports learners to connect with the right people (question tags show what domain of expertise is needed and people with that expertise can answer the question); and it motivates people to learn by encouraging them in the question/answer game with the reputation incentive gained from votes.

We used the score of the questions/answers as a metric to determine which questions/answers SO members consider more helpful. A collection of Q&A threads containing answers with relatively high scores gave us a sample of recognized answers [1]. By studying those answers, we identified attributes of good answers, such as example conciseness, importance of familiar context, and presenting best practices. We found that the prose explaining the example has the same importance as the code. We studied the structure of such explanations and different elements present in them. We further did a categorization of questions based on questioner's goals.

The contributions of this study are:

- Characteristics of good code examples;
- Attributes of the explanation that should accompany the code;
- The relationship between these attributes and the question types;
- Implications on how to enhance example and documentation development processes as well as tools that can be used during that process.

The rest of this paper has the following structure. The next section provides basic statistics about the SO web site. The third section explains how we selected our sample and how the qualitative study has been done. It also provides the definition of some important terms used throughout this paper. After that the study findings are presented. Then a discussion about the findings and the implications of them on improving current development processes for examples and documentation are presented. Related work section followed by summary and future work concludes the paper.

II. StackOverflow Statistics

Mamykina et al. conducted a statistical study of the entire SO corpus on the usage patterns to find out what is behind the immediate success of it. Some software developers believe that SO has replaced web search/forums as their main source of finding answers to their programming problems [16]. These findings show that a majority of the questions will receive one or more answers (above 90%) and very quickly (with a median answer time of 11 minutes).

Since this study was done in August 2010 and the SO web site has a rapid growth rate, we decided to recalculate some of the statistics presented in that paper using the publicly available data dump query tool[2]. The mentioned study also lacks any statistical analysis of answer votes which is a main metric in our study design and analysis. The following statistics were calculated at the end of March 2012.

There are some one million registered users on SO. They have posted 2.78 million questions, 5.77 answers and 10.5 million comments. The average score of answers is 1.8 (σ=6.2), of questions is 1.5 (σ=5.9). Fig. 1 shows the distribution of answer scores. From this figure it is obvious that higher scores are quite rare. In fact, 87% of answers have a score of 3 or less, and only 2.3% of answers have a score of 10 or greater.

Most of the questions receive at least one answer (91.8%) and receive it rather quickly (with median time of 15 minutes to receive the first answer). In the first hour, 70% of questions receive their first answer, and in the first day, 89% of them. Since accepting an answer is optional, the ratio of questions with an accepted answer is not very high (only 62.5% of questions with one or more answers have an accepted answer with the average answer score of 2.94). The median time of accepted answers being posted is 24 minutes. 63% of accepted answers are posted in the first hour after the question been posted and 87% in the first day.

III. Study Design

A. Sample Selection

Having a minimum reputation score, every SO member can vote on all the questions and answers posted on the site[3], with a positive/negative vote. Higher scores indicate a question/answer is considered more helpful by site members. To start our analysis of what a good code example is, we needed a sample of Q&A threads that:

- Contains an answer with a relatively high score.
- That answer has to contain some code.

From site statistics we know that only 13% of all answers have a score of 4 or more. We chose 4 as the high score threshold (which we consider a conservative approach for determining good answers). Therefore we consider an answer with a score of 4+ as one with a high score. For pragmatic reasons we focused on one programming language (Java). Each question is normally labeled with some tags showing the categories under which this Q&A thread would be classified. We developed a Java program to crawl the site and retrieve pages that contain the *java* tag, an answer worth 7 points or more, and the HTML <code> tag (used for highlighting programming code inside questions/answers). The crawling was stopped after visiting 150,000 pages and resulted in 497 pages having all the mentioned attributes. Then we manually inspected these pages and removed duplicates and pages that only had the javascript tag. As a result, the sample size was reduced to 357 pages. In the next round of inspection, we removed pages without any real code examples (due to the use of the <code> tag for highlighting normal text or keywords), pages without any

[1] The definition of these terms is given in Section III.

[2] http://data.stackexchange.com/stackoverflow/query/new

[3] Unless the thread is closed for some reason.

answer containing example code that also had 4 points or more, and some more duplicates[4]. Our final sample has 163 unique Q&A threads, each has at least one answer containing code that gathered 4 or more points. The shortest threads only have one answer and the longest one contains 29. Since the scores of questions and answers are subject to change due to the openness of the site, our sample shows those numbers on the date these pages were retrieved. Since the crawling program has a nondeterministic behavior due to changes in the site's contents, running it again will not reproduce that same group of pages. Therefore, we provide our sample as a compressed file for other researchers to download[5].

B. Definitions

In the rest of this paper we use a few terms and to clarify things, their definitions are provided here.

Score: The difference between upvotes and downvotes for an answer.

Normalized score: To be able to make answers to different questions comparable, we normalized the scores of all the answers in a Q&A thread using linear scaling transformation. Therefore all the normalized scores are between 0 and 1.

Recognized answer: These are answers that are distinguished either by the questioner, the community, or both. We use the following three rules to determine if an answer is recognized. All the accepted answers are recognized. All unaccepted answers with a normalized score of 0.4 and more are recognized. Since only 2.4% of all the answers posted on SO have a score of 10 or more, we decided to consider all the answers with such scores in our collection to be recognized, regardless of their normalized score.

Low-vote answer: All the unaccepted answers that have a normalized score of 0.1 or less are in this group.

Long answer: If the printed version of an answer fills two or more pages (with our printer driver), we considered it as a long answer.

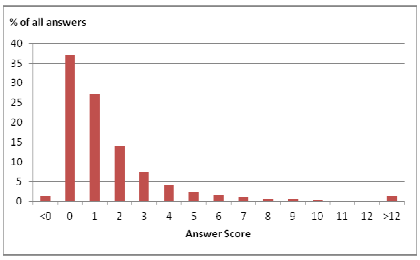

Figure 1. Distribution of answer scores on SO

[4] We could alternatively use SO's public data dump for selecting the sample, but the nondeterministic nature of the crawler ensured the randomness of the sample which was enough for this study. We used a lower threshold of 4 during our manual filtering to keep threads with answers containing real code, even with a lower score of 4 or more. We chose this lower threshold since we knew its appropriateness from the statistical analysis.

[5] It can be found at http://ase.cpsc.ucalgary.ca/uploads/ICSM2012.zip.

C. Analysis

We started our analysis by open-coding questions/answers in a subset of our sample similar to the way it is done in grounded theory [4]. We chose this method because we did not have a priori knowledge what categories would be relevant, so we chose a methodology that supports developing categories from data. We coded interesting elements of questions/answers and tried to use the language of the users for our codes. The first 63 threads were gone through open-coding and after doing it for some of them we started to find higher level categories concurrently. After doing it for those threads we realized that we reached saturation (no more new categories did emerge), but we only saw some aspects of already found categories. So we decided to stop open-coding and continue with focused-coding. Some of the categories were adopted as theoretical concepts to help us create interpretive description of the data[6].

Our main research question is what makes a good code example, so we have mainly focused on the answers being posted on SO that contain code examples. Our findings provide the attributes of those examples present within recognized answers. It was also observed that recognized answers do not just provide code. They usually provide some explanation besides the code. Therefore, another part of our findings deals with the structure and organization of the explanation element of recognized answers. Our first impression was that these two elements of an answer, namely code and prose accompanying it, are equally important and studying their attributes will help us understand what makes a good answer to a programming question. To make sure that these attributes actually make an answer a better one, we also compared the recognized answers to the low-vote ones based on these attributes. Answers do not live in the void. They are answers to some questions. Our analysis also resulted in a categorization of question types in our sample.

IV. FINDINGS

We start with a categorization of the questions. These question types are important, since we found that some answer attributes are related to the question types. The main part of this section is dedicated to the presentation of our main findings about the attributes of recognized answers. Finally the common attributes of low-vote answers will be discussed.

A. Question Types

The main goal of SO visitors is to find an answer for their programming questions. The first step is to formulate an appropriate question that provides enough information about the problem to the potential responders. The flexibility of the site enables a questioner to refine their question as the Q&A session unfolds. It is common for a question to be edited multiple times by the original poster or even by other site members. Our sample of edited questions shows that the wiki-like feature of the site makes it possible to improve the

[6] Our coding summary is publicly available at http://ase.cpsc.ucalgary.ca/uploads/ICSM2012-Coding.zip.

question content based on the feedback from other site members if the question did not originally provide enough information or it was somewhat ambiguous.

SO question types can be described based on two different dimensions. The first dimension deals with the question topic: It shows the main technology or construct that the question revolves around and usually can be identified from the question tags. These types are API related questions, object-oriented programming/design questions, language basics questions, questions about migrating to a new language and comparing constructs of two languages, and questions asking for algorithms or how to improve the performance of an algorithm.

The second dimension is about the main concerns of the questioners and what they wanted to solve. The question types in this group are:

- *Debug/Corrective*: Dealing with problems in the code under development, such as run-time errors and unexpected behavior. It could also be about working code which is not satisfactory due to its current design or structure and thus seeking a better design.
- *Need-To-Know*: Questions regarding possibility or availability of (doing) something. These questions normally show the lack of knowledge or uncertainty about some aspects of the technology (e.g. the presence of a feature in an API or a language).
- *How-To-Do-It*: Providing a scenario and asking about how to implement it (sometimes with a given technology or API).
- *Seeking-Different-Solution*: The questioner has a working code yet is seeking a different approach for doing the job.

We did not find a meaningful relationship between the question types from the first dimension and the attributes of recognized answers, so hereby we just focus on the second dimension. Question types do not necessarily show separate sets, some of them actually overlap, i.e. a question might belong to more than one type. Fig. 2 shows the distribution of second dimension question types in our sample.

B. Attributes of Recognized Answers

The responders are encouraged to provide code in their answers, since SO is dedicated to programming questions. But just providing code is not enough. The code usually is accompanied by some explanation. The analysis of recognized answers in our sample showed that they possess some attributes, either regarding the code or the explanation in the answer. Table I lists these attributes with a short description of each attribute and some examples. In the rest of this section we present the most important attributes of recognized answers in more details. For each attribute we also compared the recognized answers having it to the low-vote answers in the same thread to find out if that attribute is the distinguishing factor between them. Consider a recognized answer attribute, A. From these comparisons we found that A is a distinguishing factor if the only difference between a recognized answer and a low-vote answer, is that the low-vote answer lacks A. On the other hand, just having some of these attributes is not enough to make an answer a recognized one. Having some other (negative) attributes such as lack of enough explanation, not providing any code when it is the main goal of the questioner, using unfamiliar context, and not using best practices can lead to a low score. In other words, a good answer needs to have some basic features such as explanation, code (if it was asked by the question), and it should be correct. Only when an answer has these basic attributes, having other recognized answer attributes, presented in the rest of this section, would be a distinguishing factor to make it a recognized one.

1) Concise Code

The code presented within answers is considered as concise if either it is shorter than similar code inside other answers to the same question, has less than 4 lines of code, being labeled as concise in the comments by the audience, or it is apparent that its complexity has been reduced; for instance, when it is clear that some parts of the implementation were removed to make the code simpler.

We found that many recognized answers in our sample provide the concise solution code: 48 Q&A threads have one or more recognized answers with this attribute. One way to make code concise is to leave unnecessary details out and show their absence with some place-holders (such as comments or ellipses) which usually transforms the code to a solution skeleton. This technique is used when the implementation details are considered irrelevant or readers are assumed to easily figure out how to put the missing details back there. In our sample we have 18 recognized answers using this technique. Most of them (15) use it to show the structure of the solution (e.g. a pattern or general usage scenario of an API) and imply that the structure is more important than the implementation details. However, some implementation tips could be included as comments to show the way of transforming the skeleton into full-fledged code.

For instance, the following code snippet shows how anonymous inner classes are used in Java as an idiom in place of function-pointers in other languages. The structure of the solution is the main point of the solution, and thus the implementation of the method body is considered irrelevant and left out:

```
Collections.sort(list, new Comparator<MyClass>(){
    public int compare(MyClass a, MyClass b)
    {
        // compare objects
    }
});
```

Sometimes one line of code is the complete answer for the question (it is called a one-liner by the site members). As an example consider the following method with a single line body.

```
public static byte[] toByteArray(String s) {
    return DatatypeConverter.parseHexBinary(s);
}
```

The API method call provides the functionality needed by the questioner, namely converting a string of hex values to a byte array.

TABLE I. ATTRIBUTES OF RECOGNIZED ANSWERS

Attribute Name	What is it?	Explanation and Examples
Concise Code	Less complex and shorter code examples	- Eliminating implementation details - Using place-holders
Using Question Context	Answer code can build on top of the question code; either to correct it or improve it	- Usually results in more concise code - Using question code closely, if possible; otherwise using identifiers from it
Highlighting Important Elements	The answer starts with highlighting the key element of the solution	- The element name could be hyperlinked to external pages for further information - Some examples of these elements: the cause of error in question code, the name of an API class, and the name of a pattern/best practice
Step-by-Step Solutions	The code divided to multiple chunks, each chunk is described separately	- Suitable for explaining facts to novices - Can be used to explain how things work in detail (e.g. how code is executed) - Suitable for solutions with code in multiple files
Links to Extra Resources	The answer has hyperlinks to other sources of information	- The external resource could show more complex code for almost similar scenarios, and/or more explanation - The preferred resources could have an authoritative/official nature
Multiple Solutions	Answers in a Q&A thread could provide alternative solutions to a question	- Can be used as a refernce for people with similar questions - Alternative solutions could use different classes from one API, different classes from different APIs, or different versions of an API
Inline Documentation	Comments can be used as an alternative way of explanation	- Using normal comments, Javadoc comment, or exception messages as a means of explaining the code or using comments as place-holders for implementation details - Directly usable code with a mini-guide inside it
Solution Limitations	The answer explains the limitation of the solution	- Could be initially part of the answer, or added later due to the comments - Some examples: performance issues, usability of the solution, and security risks
API Limitations	Explicitly mentioning the shortcommings of the API used by the questioner	- It could be the lack of functionality, API design issues, or even a bug in the API - Highlighted when it is the main source of the problem for the questioner - A workaround solution is usually presnted

It is wrapped inside a method with a name reflecting the functionality mentioned in the question which makes it more relevant to the question's vocabulary than the original method name.

Comparing the recognized answers having concise code to the low-vote ones based on the length of the code inside them revealed that for half of the low-vote answers the distinguishing factor is the code length. For the rest, we saw that even though the low-vote ones have a code with similar length, some other factors contributed to their low scores, such as lack of explanation, inefficiency of the code, using a complex domain, and not covering all cases.

2) Using Question Context

Providing code is not restricted to answers. Some questions also present code snippets as part of their problem description. It is very common for the Debug/Corrective questions to contain code, but it does not mean that only this type of questions presents code. 58% of questions in our sample provide some code snippets. Fig. 3 shows the distribution of questions with code among different question types.

When the question code has some flaws or is not working, the answer usually provides some corrective suggestions. We have 10 recognized answers that apply corrections to the question code and represent the corrected version. Since the answer code does not necessarily have to show the whole code, using the question code also results in more concise code (7 out of 10). For instance, if the question code can be fixed by adding some missing statements or changing some arguments, only those changes need to be represented, probably with a few other lines of code surrounding them. A good sample of this is the answer to a question regarding polymorphic CriteriaQuery in the Hibernate API. The question code has more than ten lines of code, but the answer just suggests that the following line from the question code

```
Path<Object> path = from.join("idTag").get("code");
```

needs to be replaced with this one:

```
Path<Object> path = ((Path) from.join("idTag")
    .as(RfIdTag.class)).get("code");
```

The unexpected behavior of the question code can be a result of questioner's inaccurate knowledge of the API or language elements. Misleading documentations or misreading them can be blamed for questioner's wrong conception, as one questioner puts it:

"*the Javadoc is a bit confusing.*"

Then the solution will show how to apply changes to the question code in order to use those misused elements correctly.

The answers that use the question code are not limited to those that just provide a working solution. Some of them also provide suggestions on how to improve the code, either by improving its readability or by applying best practices/patterns/idioms to the question code, regardless of this change being the main goal of the question. This type of improvements could also have some pedagogical benefits for questioners as they learn how to write better code.

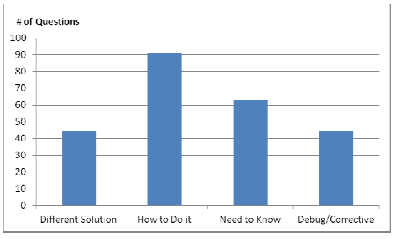

Figure 2. Distribution of question types in our sample

For instance a responder suggests the following as a best practice that the questioner should follow, even though it was not the source of the problem:

"*You should also get used to define the type of the elements in the list.*"

The extent to which the elements of the question code are used in the solution varies. The question code can be used closely if the amount of change for fixing the problem or improving the design is not considerable. In our sample, we observed that the question code has been used closely on occasions when adding or removing a few statements would fix the problem or some minor restructuring would improve the code. On the other hand, even if the solution is dramatically different in design and implementation from the question code, or when the question code is incomplete, some elements from the question (i.e. identifiers) can be used, probably to make it easier for the questioner to understand the solution. We observed that in the majority of these cases (7 out of 9) the questioner does not know how to do the task and this leads to an inadequate code snippet in the question. For instance, a questioner wants to get the integer values of `enum` members. His code shows an `enum`:

```
public enum TK{
    ID,GROUP,DATA,FAIL;
}
```

Then it shows the questioner's current solution of defining and getting those values. The accepted answer explains that this approach is against best practices, thus the solution has a completely different structure, but at least it uses the name of the `enum` type (`TK`) in the solution:

```
Map<TK,T> map = new EnumMap<TK,T>(TK.class);
TK tk = ...;
T something = ...;
map.put(tk, something);
```

Comparing the recognized answers that use question context to the low-vote ones revealed that for almost half of the low-vote answers, not using the question context in the solution was in fact the distinguishing factor. For the rest, we found that even though the low-vote ones use the question context in their code, some other factors contributed to their low scores, such as lack of explanation, longer solutions, and not using best practices.

3) Highlighting Important Elements

A considerable number of recognized answers (43 answers, each belongs to one of 36 different questions) start the discussion of the answer with highlighting the most important element of the solution. For How-To-Do-It question, it is usually the name of the API element to be used in implementing the desired questioner's scenario. The answer highlights that element, and sometimes links it to external resources like the Javadoc entry of the API element, followed by the solution using it. The highlighted element could also be a design pattern name to be used to improve the current design of the question code, when the questioner seeks to improve the current design of their code. The name of the pattern is mentioned first followed by the solution. For instance,

"*This sounds like a good candidate for the Specification pattern.*"

In 8 recognized answers to some Debug/Corrective questions, the answer starts with highlighting what causes the problem for the questioner: The lack of knowledge on how to use an API method/class; e.g. not knowing how to use special characters when using the `String.split` method. Bad programming, wrong assumptions about what the questioner implemented, or not being familiar with a development tool are some other causes of problems present in our sample. For instance:

"*You may have inadvertently selected Java Desktop Application.*"

In 5 occasions the answer starts with highlighting a list of things (in contrast to only one element). These answers have the common property that their questions can be solved in multiple ways, each with its advantages/disadvantages. The answer first provides a list of pros/cons of these alternative solutions to justify the preferred solution that follows.

Comparing the recognized answers with highlighted elements to the low-vote ones revealed that for almost half of the low-vote answers, they do not highlight the element and that is the main difference between them and the recognized ones. For the rest, we found that even though the low-vote ones do the highlighting, some other factors contributed to their low scores, such as lack of explanation and lack of code to show how to implement the required functionality or fix the problem.

4) Step-by-Step Solution

As the name implies, these answers (32 answers in our sample, each belongs to one of 21 different questions) present the solution in a detailed and ordered fashion. They might divide the code into some chunks, describing each small piece of code separately. These informative answers are utilized for different reasons.

In 7 answers, when the questioner explicitly asks about some basic facts, or when it is clear from the question or problem that they are not aware of those basic facts, this format is used; e.g. explaining the benefits of getters/setters or the bridge methods in Java. In some of these questions, the questioner explicitly describes themselves as non-experts. Apart from the step-by-step format, the examples presented in these answers have the following interesting attributes: they are quite simple and use a familiar domain. They also explain to newcomers how things work.

A good example that shows the last attribute is a question asking about an infinite loop containing a strange `x=x++` statement. Unlike normal Debug/Corrective questions, the main goal is not to find how to fix it (the fix is obvious), but rather to learn why it behaves like this. The accepted answer tries to explain how the code actually works by providing a simulation (a method that simulates the post-increment operator) and step-by-step using this method (by rewriting the original code) to explain what happens when the original code is run.

This format is also used for some How-To-Do-It questions (7 instances). The questioner is not usually a novice in these cases. For instance, they might be familiar with a construct in Java and want to know how to simulate it in C#. Since an answer to these questions normally involves multiple elements (some class and sub-class definitions, method implementations, and usage code), it seems a suitable format to use a step-by-step approach which

introduces and discusses these elements one by one and explains how they fit together: This format is used when the answer needs to provide multiple elements (sometimes in different places/files). We have 3 instances of step-by-step solutions that use this format due to the both previously mentioned reasons. They suggest using a design pattern or a best practice. Since a design pattern/best practice usually consists of multiple elements, to explain why it should be used/it is a good practice, a step-by-step format is used. But they also describe the philosophy of the design and how things work for non-experts.

Our Comparison of the recognized answers with step-by-step solutions to the low-vote ones revealed that for almost half of the low-vote answers, they do not use this format and that is the main difference. For the rest, even though the low-vote ones use this format, some other factors contributed to their low scores, such as less explanation and using a more complex code/context.

5) Providing Links to Extra Resources

The expected length of an answer, and the time window for answering, make short answers more appealing for responders (consider the high percentage of answers posted in the first hour of a thread's life). We should mention that there are few really long recognized answers in our sample, but most of the recognized answers are relatively short. To keep the answer short, the responder can provide links to external web sites that contain longer/more complex examples, contain examples with slightly different scenarios, and/or provide more detailed explanation. It is encouraged to provide such links, however the answer should be self-contained by providing a summary of the external source contents. This self-containment will prevent the answers from becoming useless due to dead links. So, an answer is expected to actually answer the question, regardless of having links to other resources. This technique is quite common for answers to questions of the Need-To-Know category (19 recognized answers to 15 different questions use this technique).

Let's see what these sources have in common: In 6 instances the external resource presents information from some well-known and respected people in the community, and it is used to justify the appropriateness of the answer. Examples of these resources are a book by a well-known author about best practices in Java, an official book presenting the language specification, an interview with the creator of the C# language, and an online FAQ web site about the Java language constructs by a well-known instructor.

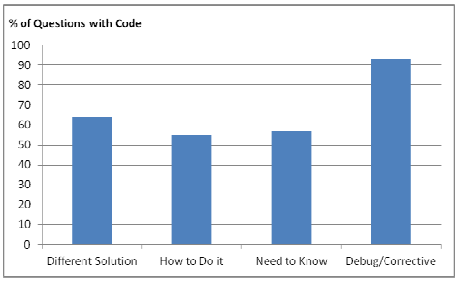

Figure 3. Percentage of questions containg code for each question type

Another source of information that can be referred to is the documentation of the API or language. The standard Javadoc entries can be referenced when the questioner is unaware of a method or class to be used. The documentation can have the format of a tutorial/reference which provides more detailed explanation of the language/API elements and has more code samples.

C. Common Attributes of Low-Vote Answers

As we discussed in the previous section, having some of the attributes of recognized answers does not necessarily translate into becoming a recognized answer. Therefore, we decided to study low-vote answers in general to see what attributes they have in common that might have led to their poor score. Lack of code, especially when the question asks for it explicitly or implicitly[7] is one shared attribute of low-vote answers. Lack of explanation is another one, even if the explanation comes in the form of code comments, it would be better than similar solutions without any explanation on how the code works. Table II presents a summary of these attributes with some examples. A good answer is one without any of these attributes in the first place. Only then possessing the attributes shown in Table I would lead to a better solution more likely to be recognized by community.

D. Recognized Answers without Code

In coding our sample we encountered a few recognized answers (32) that have one of the common attributes of the low-vote answers, i.e. they do not provide any code. This was strange, since we did not expect to see recognized answers without code due to the way we did our sampling. A further investigation clarified the situation. Half of the recognized answers without code actually refer to the code in the question and provide some explanation about that code (e.g. their dis/approval of the design of the code and the reason behind it). Three of these answers provide links to external web pages that provide code examples. The rest (except for 3 interesting cases) are answers to some high-level Kneed-To-Know questions regarding concepts, such as the pitfalls in the design of an API or comparing two API classes, which normally does not need any code to explain.

The 3 interesting answers in this group could have provided some code examples to the How-To-Do-It questions, but instead they just provide a link to a Javadoc of the API class/method or just describe the algorithm. From their comments we found that the responders might have assumed the questioners can go from this starting point and would be able to solve their problem, even without code examples. For instance, a question on how to compare two version strings has an accepted answer that just describes the scenario steps of doing the asked functionality, without showing any code. The questioner confirms in the comments that these steps are similar to *"what I suspected I'd have to resort to. This also involves looping over the tokens in the shorter of the two version strings. Thanks for confirming."*

[7] How-To-Do-It is the major question type containing questions that ask for some code examples. In our sample 95% of their recognized answers provide code examples in the answer.

TABLE II. COMMON ATTRIBUTES OF LOW-VOTE ANSWERS

Attribute	Explanation/Examples
Lack of code	Especially when a questioner asks for code example (e.g. how to initialize a static map in Java?)
Lack of explanation	e.g. not mentioning solution limitations
Shortcomings of solution	Avoidable shortcomings (e.g. throwing exceptions) Not conforming to question constraints Using an unfamiliar context Long code examples when shorter solutions exist

V. DISCUSSION

Our study has several implications for providers of the support material for developers. It can show how the process of producing supportive materials can be enhanced and what kind of tools could be used during that process. We start with a summary of some of our findings that leads to those implications. Then we discuss the implications. Finally we discuss the limitations of our study.

Customized answers: We found that a main attribute of good answers is that they are normally customized to the questioner's needs. Many answers are directly applicable to the problem, especially for the Debug/Corrective and How-To-Do-It questions. If the context of the problem is provided (either using code or in plain English), good answers would try to use it while presenting the solution. The familiar solution context makes it easier for the audience to understand and use the solution. It reduces the questioner's intellectual effort needed to apply the solution to their problem and thus decreases the cognitive distance [15]. This very attribute is one that is absent from some other information sources (documentation, open source repositories, and code search engines).

Familiar context: Even when no context is provided for the question, the responder can choose a familiar one (at least familiar for most of the audience). In this case, the questioner does not have to ponder on the unfamiliar context before they are able to understand the solution. We saw that when the questioner is relatively novice, the answer can use a more detailed format (e.g. step-by-step) to ease the learning process. In other words, the responder would choose the proper presentation format based on the questioner's expertise level. Another example is those accepted answers that just describe the solution very briefly, since they know the questioner is able to get started with this initial information and do the rest herself.

The impact of question type: The analysis of the distribution of attributes of recognized answers and their question types showed that those attributes are likely to be determined by the second group of question types (i.e. "what the questioner wants") than by the technology or construct of the question. This relationship is shown in Table III. These attributes stand out for the shown question types (i.e. they are more common for these types); however, it does not mean that they are not used for other question types at all.

Although knowledge sharing sites like SO seem to replace traditional sources of support for developers (such as API documentation), our findings show that there is still a need for more thorough sources of information dedicated to a specific API/language. The authority attribute of information sources originated from API/language designers makes them more dependable for users. Robillard had a similar finding that examples tied to API designers are more attractive for developers [23].

A. Implications

Our findings have some implications on how the current state of tools, examples, and documentation can be improved. The main advantage of the online Q&A sites, namely customized answers to the questions, is hardly transferable to the mentioned sources of knowledge; however, some of the attributes of the recognized answers can be adopted by the information providers.

Retrieval Tools: Many tools use code repositories and apply different mining techniques to retrieve examples. As our findings show, the code itself is not useful enough in many occasions. Therefore, mining knowledge repositories, such as SO and developers' forums, should be considered as an alternative for retrieving more useful examples accompanied with necessary explanation, and perhaps formatted in some ways to facilitate understanding.

Documentation: Documentations that just provide textual descriptions are not as useful as those sprinkled with several examples. To prepare such documentation, writers need to predict what kind of examples the audience would need and what questions they might have. Since nobody can anticipate each and every question, the documentation should evolve (as the API/language it describes does), not only to cover new features, but also to add examples and explanation for problems that API/language users are facing frequently which are absent from the current version of the documentation. To find out about these problems, the tools mentioned in the previous section can be handy. It is also possible to add a wiki-like capability to the online documentation, so that users' contributions can be added to it, but in a way that is easily distinguished from the official segments.

Examples: Developers of an API/language should provide a comprehensive set of examples for the potential adopters of their product [23].

TABLE III. RELATIONSHIP BETWEEN RECOGNIZED ANSWER ATTRIBUTES AND QUESTION TYPES

Question Types	Answer Attributes
Debug/Corrective	Highlighting Important Elements (e.g. cause of problem) Using Question Code (to revise it)
Need-To-Know	Highlighting Important Elements Step-by-Step Solutions Using Question Code Link to External Resources
How-To-Do-It	Highlighting Important Elements (e.g. API class name) Step-by-Step Solutions Using Question Code
Different Solution	Highlighting Important Elements (e.g. pattern name) Step-by-Step Solutions Using Question Code

This set needs to evolve, as it is the case for the documentation. Therefore they can benefit from the same knowledge mining techniques. It is also important to provide different examples for audience with different levels of expertise.

B. Limitations of the Study

Our criteria for building our sample might be regarded as too restrictive (focusing on Java Q&A and using threads containing code examples with relatively high score). However, we used it since these criteria serve our main goal of finding attributes of good code-based answers. There might be a lot of recognized answers without code, but would barely provide much insight into reaching our goal. Another limiting assumption of our study was the score of answers. We assumed that answers with higher scores generally mean better solutions; however, other factors such as answer posting time, the question topic, and the responder identity might affect these numbers. To alleviate the effects of these factors we chose a high score of 4 for answers in our sampling process to make sure that some niche questions that attracted fewer people would not be truncated. We also compared answers to the same question and not with answers to other questions.

The generalizability of our findings could be regarded as another limitation. However, our findings have some overlaps with some other studies [13, 23, 27] which suggests that our results could be generalized to some extent. At least our findings can be used as hypotheses to be examined in later studies.

Another concern would be the reproducibility of our findings. The coding of questions and answers were done by the first researcher. A second researcher then coded all the questions and 32 answers from 10 randomly selected Q&A threads. We computed Cohen's kappa inter-rater reliability values [5] for different question types and answer attributes. This value is between 0.69 and 0.79 for the question types and between 0.68 and 1 for the answer attributes. This ensures that our results could be reproduced with a high probability by other researchers.

VI. Related Work

APIs play a major role in developing software (65% of the questions in our sample ask a question related to some API). There are some studies addressing issues in learning APIs. Ko et al. found six learning barriers that end-user programmers would face. They found that programmers can overcome some of these barriers by finding relevant examples. However, they might face some other barriers when trying to adapt these examples to their needs [14]. This is not often the case with examples on Q&A web sites, since the answer code is usually tailor-made to the questioner's needs. Robillard's findings based on a survey from professional developers show that API learning resources have a similar importance as the API design on the ease of learning for developers. He also found that developers want some well-structured and complete documentation and a comprehensive set of examples that show usage best practices for different scenarios [23]. A study of discussions in a programming forum categorized three major types of questions (asking for a solution, using a wrong solution, and using a solution incorrectly) and found several obstacles imposed by the API for each category, for instance improper default solutions and hard to find elements [13]. Finding relevant Q&A threads could be a challenge, especially in large forums. Gottipati et al. developed a text classification method on different types of entries in software forums and showed that their tool would outperform normal search engines in recovering relevant answers to user queries [10].

Using the web as a main source for programmers to gather information has been the focus of some research studies. Programmers use the information available on the web for three main reasons: just-in-time learning, clarification, and remembering difficult things. They use different types of queries (natural language, code, or a mixture of both) based on their main objective [3]. Q&A web sites can be a good source to provide the needed information to programmers. An analysis on how programmers conduct Q&A on SO resulted in several question categories (how-to, review, error, conceptual, etc.) and that questions containing code are very common for review questions. Besides the question type, other factors, such as the question posting date and time, the identity of the questioner, the technology in the question, the length of the question, and the availability of code in the question affect receiving good answers [27]. The web can also provide alternative means of API documentation. A study showed that blogs cover 87.9% of a specific API's methods and provide tutorials and personal usage experiences. They also found that these sources are the main way of support for some niche communities [20].

Using examples is a main source of help for both professional [24] and end-user programmers [29]. Both groups are trying to solve a problem by reusing those examples. For novice developers, having more knowledge about the architecture of a framework will be a major factor in adapting framework usage examples more efficiently, so it was suggested that learning a framework should start with teaching novices about the design of that framework [12].

There are also tools designed to extract code from source repositories to provide them as examples. Some of them use mining technique to locate features [17, 22]; others extract common API usage patterns [1, 2, 30]. These tools rely on queries consisting of keywords to find examples. Holmes et al. developed a tool that uses the current development context (i.e. types used in code under development) as the query to find similar code snippets that might be reused [11]. The extracted code by all of these tools does not necessarily have a familiar context and is not always accompanied with helpful explanation. To overcome some of these shortcomings, Stylos et al. developed a tool that extracts some common usage scenarios using web search (such as object creation) and injects them into the API documentation [26]; albeit for very limited scenario types.

Documentation is also a main source of information for developers [23]. Dagenais and Robillard studied the creation and evolution of the documentation, and found that different types of documentation are more appropriate for libraries and

frameworks. They also found that mailing lists could be considered as bug reports and be used in the evolution of the documentation [7]. Dagenais and Ossher built a tool that helps developers to create light-weight documentation while using a framework by defining different steps of a task and adding elements to each step and thus creating a set of guides [5]. Nontrivial and infrequent knowledge about how to use elements of an API could be hidden under tons of API documentation text. Two studies defined different types of these hidden elements (called directives) [8, 18]. Dekel also built a tool integrated to an IDE to present these important elements to developers while they are using the API and showed that it improved the outcome of documentation reading for their study subjects [9].

VII. Conclusion and Future Work

Q&A web sites like SO are providing a new means for programmers to participate in social learning [28]. The shear amount of questions and answers posted on the site shows the popularity and success of this way of learning. This also provided us with a good source for studying the properties of well-received answers. We found that code examples and the accompanied explanation are two inseparable elements of recognized answers. We also found attributes of these two elements and techniques being used to shape these elements, such as making concise examples, shaping the explanation based on the questioner's expertise level, and making use of the question context to decrease the cognitive distance. These findings can be used by documentation and example developers to create more usable artifacts for potential users.

We are going to combine these findings with our previous study [19] to apply some of these findings on automated tests and study their effects on developers learning experience.

References

[1] M. Acharya, T. Xie, J. Pei, and J. Xu, "Mining API Patterns as Partial Orders from Source Code: From Usage Scenarios to Specifications," in Proceedings of ESEC/FSM 2007, New York, NY, USA, 2007, pp. 25–34.

[2] S. K. Bajracharya, J. Ossher, and C. V. Lopes, "Leveraging Usage Similarity for Effective Retrieval of Examples in Code Repositories," in Proceedings of FSE 2010, New York, NY, USA, 2010, pp. 157–166.

[3] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code," in Proceedings of CHI 2009, New York, NY, USA, 2009, pp. 1589–1598.

[4] K. C. Charmaz, Constructing Grounded Theory: A Practical Guide through Qualitative Analysis. Sage Publications Ltd, 2006.

[5] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, Apr. 1960.

[6] B. Dagenais and H. Ossher, "Automatically Locating Framework Extension Examples," in Proceedings of FSE 2008, New York, NY, USA, 2008, pp. 203–213.

[7] B. Dagenais and M. P. Robillard, "Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors," in Proceedings of FSE 2010, New York, NY, USA, 2010, pp. 127–136.

[8] U. Dekel and J. D. Herbsleb, "Improving API Documentation Usability with Knowledge Pushing," in Proceedings of ICSE 2009, Washington, DC, USA, 2009, pp. 320–330.

[9] U. Dekel and J. D. Herbsleb, "Reading the Documentation of Invoked API Functions in Program Comprehension," in Proceedings of ICPC 2009, 2009, pp. 168–177.

[10] S. Gottipati, D. Lo, and J. Jiang, "Finding Relevant Answers in Software Forums," in Proceedings of ASE 2011, 2011, pp. 323–332.

[11] R. Holmes, R. J. Walker, and G. C. Murphy, "Strathcona Example Recommendation Tool," in ACM SIGSOFT Software Engineering Notes, New York, NY, USA, 2005, pp. 237–240.

[12] D. Hou, "Investigating the effects of framework design knowledge in example-based framework learning," in Proceedings of ICSM 2008, 2008, pp. 37 –46.

[13] D. Hou and L. Li, "Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions," in Proceedings of ICPC 2011, 2011, pp. 91–100.

[14] A. J. Ko, B. A. Myers, and H. H. Aung, "Six Learning Barriers in End-User Programming Systems," in Proceedings of VL/HCC 2004, Washington, DC, USA, 2004, pp. 199–206.

[15] C. W. Krueger, "Software Reuse," ACM Comput. Surv., vol. 24, no. 2, pp. 131–183, Jun. 1992.

[16] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design Lessons from the Fastest Q&A Site in the West," in Proceedings of the 2011 annual conference on Human factors in computing systems, New York, NY, USA, 2011, pp. 2857–2866.

[17] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: A Search Engine for Finding Functions and Their Usages," in Proceedings of ICSE 2011, 2011, pp. 1043–1045.

[18] M. Monperrus, M. Eichberg, E. Tekes, and M. Mezini, "What Should Developers be Aware of? An Empirical Study on the Directives of API Documentation," Empirical Software Engineering, Dec. 2011.

[19] S. M. Nasehi and F. Maurer, "Unit Tests as API Usage Examples," in Proceedings of ICSM 2010, 2010, pp. 1–10.

[20] C. Parnin and C. Treude, "Measuring API Documentation on the Web," in Proceedings of Web2SE 2011, New York, NY, USA, 2011, pp. 25–30.

[21] J. L. Plass, R. Moreno, and R. Brünken, Cognitive Load Theory, 1st ed. Cambridge University Press, 2010.

[22] M. Revelle, B. Dit, and D. Poshyvanyk, "Using Data Fusion and Web Mining to Support Feature Location in Software," in Proceedings of ICPC 2010, 2010, pp. 14–23.

[23] M. P. Robillard, "What Makes APIs Hard to Learn? Answers from Developers," IEEE Softw., vol. 26, pp. 27–34, Nov. 2009.

[24] M. B. Rosson and J. M. Carroll, "The Reuse of Uses in Smalltalk Programming," ACM Trans. Comput.-Hum. Interact., vol. 3, no. 3, pp. 219–253, Sep. 1996.

[25] C. B. Seaman, "The Information Gathering Strategies of Software Maintainers," in Proceedings of ICSM 2002, 2002, pp. 141 – 149.

[26] J. Stylos, A. Faulring, Z. Yang, and B. A. Myers, "Improving API Documentation Using API Usage Information," in Proceedings of VL/HCC 2009, Washington, DC, USA, 2009, pp. 119–126.

[27] C. Treude, O. Barzilay, and M.-A. Storey, "How Do Programmers Ask and Answer Questions on the Web?" in Proceedings of ICSE 2011, New York, NY, USA, 2011, pp. 804–807.

[28] J. Vassileva, "Toward Social Learning Environments," IEEE Transactions on Learning Technologies, vol. 1, no. 4, pp. 199–214, Dec. 2008.

[29] S. Wiedenbeck, "Facilitators and Inhibitors of End-User Development by Teachers in a School," in Proceedings of VL/HCC 2005, 2005, pp. 215– 222.

[30] T. Xie and J. Pei, "MAPO: Mining API Usages from Open Source Repositories," in Proceedings of MSR 2006, New York, NY, USA, 2006, pp. 54–57.