Detecting API Usage Obstacles: A Study of iOS and Android Developer Questions

Wei Wang and Michael W. Godfrey
David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, ON, Canada
{w65wang, migod}@uwaterloo.ca

***Abstract*—Software frameworks provide sets of generic functionalities that can be later customized for a specific task. When developers invoke API methods in a framework, they often encounter obstacles in finding the correct usage of the API, let alone to employ best practices. Previous research addresses this line of questions by mining API usage patterns to induce API usage templates, by conducting and compiling interviews of developers, and by inferring correlations among APIs. In this paper, we analyze API-related posts regarding iOS and Android development from a Q&A website, $stackoverflow.com$. Assuming that API-related posts are primarily about API usage obstacles, we find several iOS and Android API classes that appear to be particularly likely to challenge developers, even after we factor out API usage hotspots, inferred by modelling API usage of open source iOS and Android applications. For each API with usage obstacles, we further apply a topic mining tool to posts that are tagged with the API, and we discover several repetitive scenarios in which API usage obstacles occur. We consider our work as a stepping stone towards understanding API usage challenges based on forum-based input from a multitude of developers, input that is prohibitively expensive to collect through interviews. Our method helps to motivate future research in API usage, and can allow designers of platforms — such as iOS and Android — to better understand the problems developers have in using their platforms, and to make corresponding improvements.**

I. Introduction

Today's software development relies heavily on pre-defined software frameworks, presented primarily as application programming interfaces (APIs). APIs provide developers with reusable libraries, programming paradigms, and task delegation, with the aim of helping to deliver useful systems quickly and with high-quality code.

However, developers sometimes complain that software frameworks often provide one-size-fits-all mechanisms, and a lack of quality documentation for typical uses can reduce productivity of developers. Designers of software frameworks may further fail to envision certain functionalities that are required; designers may also introduce complexity in API usage to accommodate a large variety of possible uses. Lines of research regarding API usage includes field studies of API learning obstacles [1],[2], mining temporal correlations among API invocations [3],[4] and harnessing web content to aid API learning [5].

When developers encounter API usage obstacles, a common approach is to post questions on an information-sharing site that is dedicated to programming technology. With that in mind, we propose to mine selected posts from the well-known technology Q&A website *stackoverflow.com* (referred to as **SOF** hereafter), in order to discover APIs from two mobile development platforms — Android and iOS — that frequently cause usage obstacles.

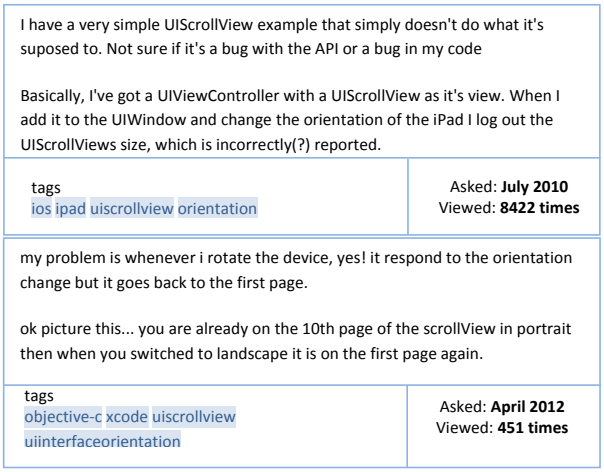

Fig. 1. Excerpts from two posts in *stackoverflow.com*. Both posters had encountered unexpected UI behaviour in a "scrollview" when changing the orientation of mobile devices.

Consider the two SOF posts shown in Figure 1; both posts concern UI defects that manifest themselves when users change the orientation of their implementation of scroll view of iOS devices. As shown in Figure 1, SOF users specify their problems and questions in natural language; users can provide tags to label their posts (shown in the left-bottom part in both posts), and may also include related source code fragments.[1] The first post has been viewed for more than eight thousand times by other SOF members, suggesting that many developers have encountered similar problems.

One of the basic assumptions made in this paper is that SOF posts that are tagged with API class names are mostly HOW-TOs concerning the tagged APIs. Treude et al. [8] report that in general, most SOF posts are programming questions. It is our experience that this observation is especially true for posts with API tags. Therefore, by analyzing SOF posts that are tagged by API-classes [9], we can attempt to detect which

[1]For details of the particular SOF posts, please refer to the links [6], [7]; only excerpts of posts are shown here due to space limitations.

978-1-4673-2936-1/13 © 2013 IEEE

MSR 2013, San Francisco, CA, USA

parts of the iOS and Android APIs are particularly likely to cause learning or usage obstacles. We decided to study only iOS- and Android-related SOF posts for two reasons: first, these two platforms belong to the same problem domain — mobile device applications — so we expect that results from two platforms can be contrasted with each other; and second, both platforms were both released in 2007 — one year before the launch of SOF itself — so we can capture a relatively complete history of API-related questions regarding the two platforms.

Our approach is novel in two ways over previous work in studying API learning and usage obstacles. First, we harnesses input contributed by a large number of developers with relatively little effort compared to, say, interviews or similar resource-intensive techniques. Second, we explicitly aim at pinpointing concrete problems that cause API usage obstacles; we believe concrete API usage problems are more likely to lead to actionable solutions.

Our approach comprises two stages. First, we compare the rank of API classes mentioned in SOF posts against the rank of API usage frequency mined from a collection of iOS and Android open source applications. Second, after identifying API classes with apparent usage obstacles, we then summarize possible scenarios of each API by applying a topic modelling tool on all the posts that are tagged with the underlying API.

Our method is intended to supplement existing techniques to study API usage obstacles. Our method can filter out API classes that do not commonly lead to usage obstacles; this allows us to apply more resource-demanding techniques, such as mining API usage partial orders [3] or field studies [2] on API classes that are more likely to cause usage obstacles.

We do not analyze answers to the developer questions posted to SOF in this paper. Answers under each post may represent examples about how to use each API classes correctly. However, according to policies of SOF, there is a rather low entry-barrier to submitting answers, and the only incentives for good answers are SOF "reputation marks". A collaborative information site that bear these two features generates answers with a large quantity but poor quality, an attribute that can be explicitly inferred under a free-entry Nash equilibrium model [10]. Therefore, researchers must overcome this problem before attempting to extract problem solutions from SOF answer posts.

II. Research Method

A. API Usage in Source Code

Simply counting the number of times an API element is mentioned in SOF may not yield a fair estimate of how problematic it is. For example, a widely used API that has relatively few problems may have more mentions in SOF than a poorly designed API that gets less use. To account for this, we establish a baseline of API usage frequency by mining source code repositories of open source projects that use the iOS and Android platforms.

The open source repository of iOS applications used in this paper is provided by*maniacdev.com* [11], a website that provides iOS development tutorials and education. This repository has undergone six iterations over two years. We found 55 valid iOS applications implemented in Objective-C, including some popular products, such as the official application for *Wikipedia* and *Last.fm*. Likewise, F-Droid [12] is our FOSS repository for Android applications, from which we collected 250 valid Android applications.

A natural approach to analyzing API usage would be to extract elements of the abstract syntax tree (AST) that results from compiling the source code. Most compilers that generate an AST (such as *clang* or *GCC*) require a build tool to specify compilation order and external dependencies. However, for the Android and iOS application sources we collected, a majority of them either did not include proprietary libraries that they depend on or or did not include an build recipe. To overcome this problem, we adopt a straightforward searching method. Specifically, we extract all class names from the official reference site of both iOS [13] and Android [14]. Using the class names from the official reference sites, we created regular expressions that match three key usage patterns: extending from a platform class, instantiating a platform class, and invoking a static method from a platform class. These three patterns would seem to cover most typical library usages, so we feel that this is a reasonable approximation of the API usage that would be inferred by AST analysis.

B. Normalizing and Comparing API Usage and API Tags

For all SOF posts, we collect all tags that can be mapped to API classes in the official reference of the iOS and Android mobile platforms. After searching for API uses of every project, API classes are ranked in each project and in each platform. Some API classes are used particularly frequently — such as *NSString* in iOS — while some other API classes are used in almost all projects but with low frequency. In order to summarize API invocations for different projects, we normalize API calls within each project. Not knowing the potential distribution of API invocations, we adopt an *ad hoc* approach to normalize the scores of each invocation calls across projects. For an API class c_i in a project p_n, a scoring function $R_{Pn}(c_i)$ has four possible values — 1, 2, 3, or 4 — depending on which quartile the count of c_i lies within; the upper quartile maps to 4, and the lower quartile to 1. The overall score of an API class $\mathbf{R}(c_i)$ is the arithmetic mean of $R_{Pn}(c_i)$ of each project. Formally, for an API class c_i, and the set of projects that call that API class P, the overall score $\mathbf{R}(c_i)$ for c_i is:

$$\mathbf{R}(c_i) = \frac{\sum_{n \in P} R_{p_n}(c_i)}{|P|}$$

By using this scoring method, we do not disproportionately reward API classes that are frequently used in a project while at the same time preserving the relative rank among API classes.

C. Topic Mining For Posts

After listing tags that are frequently mentioned in SOF posts but not frequently used in source code, posts with these tags

TABLE I
API USAGE FREQUENCY SHORTLIST AND API USAGE OBSTACLE LIST FOR IOS AND ANDROID PLATFORMS. API CLASSES THAT APPEAR ON BOTH THE FREQUENCY AND OBSTACLE LISTS ARE HIGHLIGHTED. WE OMIT FULLY QUALIFIED NAMES FOR API CLASSES WHEN THERE IS NO REPETITIVE NAME.

iOS (tags)	iOS (quartile)	Android (tags)	Android (quartile)
UITableView	NSUbiquitousKeyValueStore	**Intent**	ViewManager
UIView	NSString	android.app.Activity	DataType
UITableViewCell	**UITableView**	**android.widget.***	EntityIterator
UINavigationController	NSArray	AsyncTask	RemoteCallbackList
UIScrollView	NSIndexPath	BroadCastReceiver	**android.widget.***
UIImageView	UIColor	android.app.Fragment	MediaStore
UIButton	UIApplication	MapView	DropBoxManager
UIImage	NSFileHandle	WebView	CommonDataKinds.Im
UITabbarController	NSError	ActionBar	ValueAnimator
MKMapView	**UITableViewCell**	android.manifest.*	PropertyValuesHolder
UITextView	UIImage	ViewPager	TwoLineListItem
UITableViewController	NSDictionary	android.hardware.Camera.*	BufferInfo
UIImagePickerController	**UIView**	SharedPreferences	KeyFrame
UIPopoverController	NSURL	android.R.layout	android.R.id
UITabBar	NSURLConnection	ContentProvider	**Intent**
UISegmentedControl	UIBarButtonItem	ExpandableListView	ContactsContract.Presence
UILocalNotification	UIDocumentInteractonController	AlarmManager	ResultReceiver
UISlider	UIFont	ProgressDialog	android.database.Cursor
UIActionSheet	NSPort	MediaPlayer	android.util.Base64
NSURLRequest	NSException	android.provider.Contacts	AnimatorSet
MFMailComposeViewControll	NSUserDefaults	android.gcm.*	android.app.Dialog

TABLE II
SCENARIOS OF USAGE OBSTACLES. FOR ACCESS OF ORIGINAL POSTS UNDER EACH SCENARIO, PLEASE REFER TO FOOTNOTE 2.

API Class	#posts with this tag	Example of usage obstacles
UINavigationController	3,506	When adding navigation controller to a nested view, the nested view does not work properly
UIScrollView	3,448	Content not displayed as desired when user changes the orientation of a mobile device
UIImageView	2,837	Memory leaks when using UIImageView
android.app.Activity	2,885	Alarm timer does not work as intended within a lifecycle of an instance of android.app.Activity
AsyncTask	1,851	Progress indicator (ProgressDialog) is frozen when it is in an AsyncTask
BroadCastReceiver	1,409	Developers do not know how to trigger BroadCaseReceiver when a service is stopped

can be further examined to discover concrete scenarios of each obstacle. Similar to the approach of Baura et al. [15] we use a topic modeling technique —- Latent Dirichlet Allocation (LDA) as implemented by the Mallet tool [16] — to discover potential scenarios behind each usage obstacle. We tuned Mallet to generate 100 topics for posts under each API class. When applying LDA, we do not remove source code from posts since LDA does not analyze semantic relations among words. To our knowledge, no tools are readily available to produce human-readable topics. Therefore, we manually crosschecked between keywords of each topic and contents of posts to extract common scenarios in which API usage obstacles occur.

III. EXPERIMENT RESULT

A. Shortlists of API Usage Obstacles

Shortlists of API usage obstacles and API usage frequency are shown in Table I. The shortlist of API usage obstacles of iOS contains mainly of UI elements and UI controllers, while for Android, we find components of many kinds. This is partly due to the fact that most Android UI elements are contained within the directory "*android.widget*", which maps to only one tag (*Android-Widget*) in the SOF community.

With the shortlist of API usage obstacles, we then run a topic mining tool — Mallet [16] — on posts of each API. Mallet generates lists of keywords from posts, then ranks items according to a probabilistic model. Table II shows the partial result of our manual generalization of scenario under which some usage obstacles occurs. Each scenario is discussed in SOF at least four times by different authors. Due to the space limitations, we show examples for only the top three API classes for each platform. To encourage discussion, we also provide links to SOF posts under each scenario in a publicly available website.[2]

In the obstacle scenarios shown in Table II, most of them appear to arise from interactions among components of the platform. It seems that many developers do not know how to define interactions between *UIScrollView* and UI elements that control orientation display (landscape or portrait) in iOS,

[2] *http://swag.uwaterloo.ca/~w65wang/msrchallenge.html*

nor do many developers know how to design an alarm timer within a lifecycle of an event-driven Android object.

Of course, we believe that there are likely to be many more scenarios that cause API usage obstacles. While we do not provide a complete list of these scenarios nor attempt to validate them, we argue that the results shown in Table II are important for two reasons. First, our technique opens up the possibility to invite designers of platforms such as Android and iOS to collect and analyze usage obstacles of their APIs as a *disciplined* practice. To the best of our knowledge, there are no dedicated channels through which platform designers can learn about problems of their platform designs from feedback of average developers. Second, our approach has the potential to direct researchers to "obstacle hotspots" by using forum-based input contributed by a multitude of developers, so that more resource-demanding techniques — such as natural language processing [17] and field studies — become feasible to pinpoint concrete issues.

B. Threats to Validity

Internal Threats: As pointed out by Treude et al. [8], nearly five percent of posts on SOF are non-technical, such as experience sharing or career advice. We believe this would not bring substantial error to this paper, since it is safe to assume that few developers would discuss issues other than programming questions in a post that is tagged with an API class name. Our method also relies on correct tagging of each post. However, analyzing APIs in content of posts would bring new source of error, and may require semantic analysis. We leave this to the future work. Another source of internal threat comes from our manual intervention of generating common scenarios of usage obstacles. To our knowledge, there is no automated tool that generates similar results. We also provide access to posts under each scenarios to encourage review and discussion from our colleagues.

External Threats: We have only studied one information-sharing site and software artifacts under two mobile development platforms. This paper emphasizes on presenting the feasibility of harvesting posts from SOF to infer API usages obstacles; we also notice that tags for both Android and iOS topics happen to match well with API classes in two platforms. This may not apply well to other development platforms.

IV. Conclusions

Modern software frameworks provide rich sets of functionality; they are designed so that developers can take advantage of powerful common infrastructures, avoid reinventing the wheel, and delegate many routine design tasks. However, even well-designed frameworks are large and complicated, and their use entails a significant conceptual cost and a steep learning curve.

In this paper, we investigate posts in an information sharing website, *stackoverflow.com*, where most posts concern technical problems that software developers have. By comparing the list of frequently mentioned API classes in posts against the list of API usage frequency, we are able to discover API classes that often cause usage obstacles but are not frequently used. We also discover a few scenarios that appear to be the common cause of API usage obstacles. To the best of our knowledge, among research in API usage obstacles, this paper is the first that uses the input of a multitude of developers to reduce the problem space to a few "obstacle hotspots", so that more resource-demanding techniques can be applied to pinpoint specific problems behind each API usage obstacles. This would in turn make possible concrete solutions to these problems.

We believe the result of this paper opens new research opportunities. For example, researchers can adjust weights to each post by analyzing the emotion of developers through sentiment analysis. An *ad hoc* solution to usage obstacles is to provide aid through improved API documentation. Suggestions for API documentation improvements can be generated by linking official documentation with API usage obstacle hotspots. We leave these to future work.

References

[1] M. P. Robillard, "What makes APIs hard to learn? Answers from developers," *IEEE Software*, vol. 26, no. 6, pp. 26–34, 2009.

[2] M. P. Robillard and R. DeLine, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.

[3] M. Acharya, T. Xie, J. Pei, and J. Xu, "Mining API patterns as partial orders from source code: From usage scenarios to specifications," in *Proc. of 15th FSE*. ACM, 2007, pp. 25–34.

[4] G. Uddin, B. Dagenais, and M. P. Robillard, "Analyzing temporal API usage patterns," in *Proc. of the 26th IEEE/ACM Intl. Conf. on Automated Software Engineering*, November 2011, pp. 456–459.

[5] C. Parnin and C. Treude, "Measuring API documentation on the web," in *Proc. of the 2nd Intl. Workshop on Web 2.0 for Software Engineering*, vol. 2, 2011, pp. 25–30.

[6] stackoverflow.com, "UIScrollView not getting new size on orientation change," accessed in Feb. 2013. [Online]. Available: http://stackoverflow.com/questions/3221456/uiscrollview-not-getting-new-size-on-orientation-change?rq=1

[7] ——, "Objective-C: Orientation change with UIScrollview," accessed in Feb. 2013. [Online]. Available: http://stackoverflow.com/questions/10380896/objective-c-orientation-change-with-uiscrollview

[8] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web?" in *33rd ACM/IEEE Intl. Conf. on Software Engineering, NIER track*, 2011, pp. 804–807.

[9] A. Bacchelli, "Mining challenge 2013: StackOverflow," in *The 10th IEEE Working Conf. on Mining Software Repositories*, 2013 (to appear).

[10] A. Ghosh and P. McAfee, "Incentivizing high-quality user-generated content," in *Proceedings of the 20th international conference on World wide web*. ACM, 2011, pp. 137–146.

[11] ManiacDev.com, "Open source iPhone and iPad apps: Real iOS source code examples," accessed in Feb. 2013. [Online]. Available: http://maniacdev.com/2010/06/35-open-source-iphone-app-store-apps-updated-with-10-new-apps/

[12] F-Droid.com, "Android foss repository," accessed in Feb. 2013. [Online]. Available: http://f-droid.org/

[13] "iOS developer library," Apple Inc., accessed in Feb. 2013. [Online]. Available: https://developer.apple.com/library/ios/navigation/index.html#section=Resource\%20Types\&topic=Reference

[14] "Class Index of Android API Classes," Google Inc., accessed in Feb. 2013. [Online]. Available: http://developer.android.com/reference/classes.html

[15] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? An analysis of topics and trends in StackOverflow," *Empirical Software Engineering*, pp. 1–36, 2012.

[16] A. K. McCallum, "Mallet: A machine learning for language toolkit," 2002, http://mallet.cs.umass.edu.

[17] S. Haiduc, J. Aponte, and A. Marcus, "Supporting program comprehension with source code summarization," in *Proc. of the 32nd ACM/IEEE Intl. Conf. on Software Engineering*, 2010, pp. 223–226.