SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2017, Vol. 60 092111:1–092111:14
doi: 10.1007/s11432-017-9058-0

Code recommendation for Android development: how does it work and what can be improved?

Junwei WU[1,2], Liwei SHEN[1,2*], Wunan GUO[1,2] & Wenyun ZHAO[1,2]

[1]*School of Computer Science, Fudan University, Shanghai* 200433*, China;*
[2]*Shanghai Key Laboratory of Data Science, Fudan University, Shanghai* 200433*, China*

Received January 7, 2017; accepted March 29, 2017; published online July 28, 2017

Abstract Android applications are developed based on framework and are always pattern-based. For Android developers, they can be facilitated by code recommendation to ensure high development efficiency and quality. Existing research work has proposed several methods and tools to support recommendation in diverse ways. However, how code recommendation work in Android development and what can be further improved to better support Android development has not been clarified. To understand the reality, we conduct a qualitative review on current code recommendation techniques and tools reported in prime literature. The collected work is first grouped into three categories based on a multidimensional framework. Then the review is performed to draw a comprehensive image of the adoption of recommendation in Android development when meeting specific development requirements. Based on the review, we give out possible improvements of code recommendation from two aspects. First, a set of improvement suggestions are presented to enhance the ability of the state-of-the-art code recommendation techniques. Second, a customizable tool framework is proposed to facilitate the design of code recommendation tools and the tool framework is able to integrate the recommendation features more easily.

Keywords Android, code recommendation, code search, code suggestion, code completion, code generation

Citation Wu J W, Shen L W, Guo W N, et al. Code recommendation for Android development: how does it work and what can be improved? Sci China Inf Sci, 2017, 60(9): 092111, doi: 10.1007/s11432-017-9058-0

1 Introduction

Android, as the most popular mobile platform, has attracted a large number of developers to build various applications for its ecosystem. However, the average experience level of developers is falling as the population grows in mobile application development[1)]. Since Android apps are developed on its specific framework and their features are always pattern-based, novice developers, in particular, often require various kinds of support from IDEs (integrated development environments) on the implementation of specific functionalities such as the program structure and APIs (application programming interfaces).

Code recommendation has been one of the most popular features that are commonly provided by modern IDEs. It helps developers improve their programming efficiency by providing suggestions for

* Corresponding author (email: shenliwei@fudan.edu.cn)

1) VisionMobile. State of developer nation q1 2016 (developer economics 2016). https://www.visionmobile.com/reports/developer-economics-state-of-developer-nation-q1-2016. 2016.

their tasks, e.g., which API method to be invoked next, or which sample codes can be referred to. Besides the IDE features, researches have proposed diverse recommendation methods and prototype tools in literature. However, how do code recommendations play their role in Android development and what can be further improved to better support Android development has not been clarified.

To understand the reality and draw a comprehensive image of how code recommendation works in Android development, we conduct a qualitative review on current code recommendation techniques and tools reported in prime literature. The collected work from the literature is first grouped into three categories based on a multidimensional framework and then the review is performed towards specific recommendation requirements. The review results indicate that current code recommendation techniques are able to achieve recommendation requirements in different extent but still need improvement. Therefore, we further give out possible improvements of code recommendation from two aspects. First, a set of improvement suggestions are presented in which several future research directions are summarized to alleviate the deficiencies and to enhance the ability of the current code recommendation techniques. Second, in order to facilitate the design and development of the future code recommendation tools, we propose a customizable tool framework which covers diverse code recommendation features. Toolsmith thus is able to derive a tool architecture by customizing the framework according to specific requirements.

This paper extends our earlier work [1] mainly from the aspect of introducing the customizable tool framework. The framework is proposed to enrich the improvement suggestions toward the code recommendation for Android development.

The contribution of the paper is summarized in the following three aspects.

• First, we provide a multidimensional code recommendation categorization framework for state-of-the-art techniques and tools.

• Second, we conduct a qualitative study and summarize the benefits and deficiencies of the recommendation techniques towards a set of requirements scenarios.

• Third, we propose a customizable tool framework for facilitating the design of future code recommendation tools.

The remainder of this paper is organized as follows. Section 2 presents a multidimensional categorization framework and the categorization of the code recommendation techniques and tools reported in literature. Section 3 presents the qualitative review whose process includes the extraction of code recommendation requirements and the evaluation of the mechanisms towards the requirements. Section 4 presents possible improvements and research directions of state-of-the-art code recommendation techniques, while Section 5 gives out the customizable tool framework for facilitating the design of future recommendation tools. Finally, Section 6 presents the conclusion as well as the future work.

2 Overview

Code searching, code suggestion, code completion and code generation are used as related terms in existing literature or tools. We adopted these terms as tightly related keywords to collect a set of publications in prime software engineering conferences and journals. In this section, a multidimensional categorization framework is first proposed which is used to characterize the mechanisms of the collected publications. Furthermore, the related techniques and tools are further divided into three categories from the viewpoint of recommendation scenarios.

2.1 Multidimensional categorization framework

The framework is shown in Figure 1. It is composed of six dimensions, each explains a specific feature of a recommendation mechanism.

(1) Scope. **parameter**, **method**, **API usage** and **snippet** are scopes with different granularity which can be derived by a recommendation mechanism.

(2) Objective. The dimension indicates the objective of the recommendation when it is adopted. Four objectives are defined. **search** helps to fetch related programming assistant resources by constructing

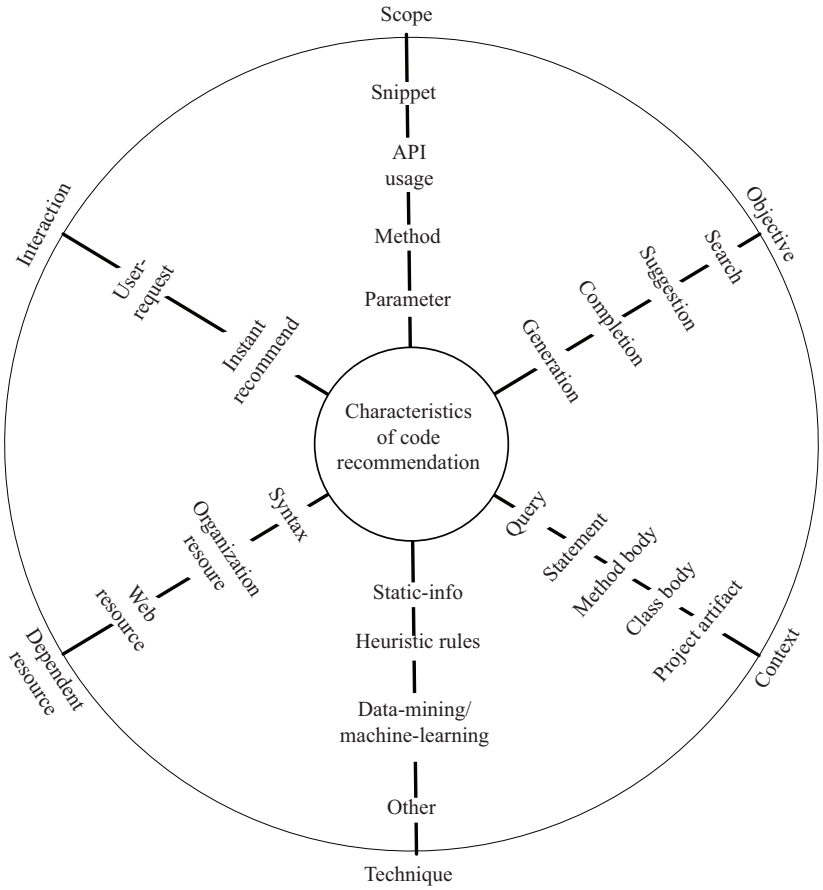

Figure 1 Multidimensional categorization framework.

querying statements with users' anticipation. **suggestion** offers users coding advices covering sample codes and design solutions by taking the programming context into consideration. **completion** enables to recognize a user's typing behavior and fill in the complete program elements especially when he is not sure about the names of the elements. **generation**, on the other side, helps to construct one or a group of program sentences and fill them in the place they are necessary by analyzing the programming context.

(3) Context. The dimension indicates the artifacts that are collected and analyzed in order to derive the accurate recommendation results. **query**, **statement**, **method body**, **class body** and **project artifact** are the five defined groups of context. The first group of context denotes the querying statements through which developers search for programming suggestions. Statements are always formed in natural language or sentence templates. The rest groups correspond to the different granularity of artifacts which can be a single code statement, or even the artifacts of the whole project. In particular, an Android project artifact includes Android-specific resources such as layouts and manifest files besides the program codes.

(4) Technique. Different underlying techniques of this dimension may lead to comparable recommendation quality. We simply explains commonly adopted techniques from the literature. **static-info** is one of the techniques which gives out the recommendation results based on the programming language syntax and the static type system as well. **heuristic rules** is the technique to derive the suggestions by computing the pre-defined rules based on the current context. **data-mining** and **machine-learning** are popular techniques to mine the re-occurring programming patterns from a large scale of resources, e.g., the online web repositories. Based on the collected knowledge, they are able to to provide the suitable patterns in similar programming context. Besides the above techniques, there exist other techniques which show their effectiveness in code recommendation. We group these techniques in the category **other**. Collaborative filtering [2], the manually resource collection and the traditional static and dynamic program analysis are the typical techniques that have been adopted in literature.

(5) Dependent resource. This dimension means the source of the recommendation results, i.e., from what scope of resources the suggestions are synthesized. We identify three categories of dependent resources that are **syntax**, **organization resource** and **web resource**. Syntax indicates the static typing system of a programming language. The available API methods of an object instance can be derived on the language level. Organization resource includes the artifacts from all the projects implemented by an organization while web resource denotes the online repositories such as GitHub.

(6) Interaction. The dimension indicates the user-interaction mode when a recommendation is required. We identify two categories of interaction according to the trigger time of the recommendation. **user-request** is an active interaction mode in which user triggers a recommendation through typing specific characters or clicking specific buttons. On the contrary, **instant-recommend** is a passive mode from the viewpoint of users. Under the circumstance, user is passively offered with the suggestions once the mechanism decides to trigger the recommendation by means of continuously analyzing user's programming context.

2.2 Category of recommendation

The dimensions of the categorization framework indicate the important factors for establishing a recommendation mechanism. Among the factors, some are visible to the end users including scope, objective, dependent resource and interaction. These factors differentiate the corresponding tools so that the Android developers can easily recognize and choose the suitable tool. The others are underlying techniques and contexts that differentiate the recommendation quality but may be not clearly understood by developers. In this paper, from the perspective of how the Android developers anticipate the recommendation and how they adopt them, we define the following three categories.

• Code searching. Searching is a special type of recommendation since it can perform independent of the programming context of IDEs. Now almost all the developers are accustomed to submitting queries in public code repositories, e.g., GitHub or online search engines, e.g., Google. In addition, there are also search engine embedded plugins in modern IDEs which help to accelerate the searching process.

• Basic recommendation. This category offers fundamental recommendation features which have been equipped by all the modern IDEs. These features are acknowledged by all the developers and the developers have been facilitated to quickly complete the API methods for example by typing 'dot' after an object instance. Under the circumstance, users read the recommendation results generally organized in lists and decide which to follow. In technical aspect, the basic recommendation mechanism mainly depends on the static information of an Android applicatioin.

• Advanced recommendation. The rest of the recommendation mechanisms are grouped into this category. Different from the basic recommendation, the advanced recommendation mechanisms adopt advanced techniques rather than the static information. The adoption of the techniques such as data mining and machine learning which compute and derive the suggestions from large scale of resources usually outperforms the basic recommendations. Therefore, the objective of code completion and code generation can be satisfied by the mechanisms.

According to the category definition, the methods and tools from the literature we collected have been allocated in Table 1.

Among the listed mechanisms, there exists methods or tools contributing to Android programming recommendation. For example, ParamHarver [11] enables to extract parameters from Android applications and the parameter values can be offered to users as recommendations in similar program sentences. The method proposed by Raychev et al. [12] indexed the method call sequences of plenty of Android applications into a statistical language model. The corresponding mechanism then adopted the language model to predict and generate the most likely call sequences for a program with holes. DroidAssist [19] is introduced to recommend Android API usages based on hidden Markov model. The method enables to advise the next method call as well as more suitable method sequence. In addition, they proposed another method to train a statistical, generative model of API usage from byte codes [22]. Code completion systems can rely on the model to derive possible method calls.

Table 1 Categorization of methods and tools for code recommendation

Category	Method/tool	Scope	Objective	Context	Technique	Dependent resource	Interaction
Code searching	Thung et al. [3]	API usage	Search	Query	Data-mining	Web resource	User-request
	ROSF [4]	Snippet	Search	Query	Machine-learning	Web resource	User-request
	QECK [5]	Snippet	Search	Query	Other (query expansion)	Web resource	User-request
	DeepAPI [6]	API usage	Search	Query	Machine-learning	Web resource	User-request
Basic recommendation	Graphite [7]	Snippet	Generation	Project artifact	Heuristic rules	Syntax	User-request
	Pythia [8]	Method	Completion	Statement	Other (static analysis and dynamic analysis)	Syntax	Instant-recommend
	RCC candidate sorter [9]	Method	Completion	Statement	Static-info	Organization resource	Instant-recommend
	IDE completion system	Parameter / method	Completion	Statement	Static-info	Syntax	Instant-recommend
Advanced recommendation	Precise [10]	Parameter	Suggestion	Statement	Heuristic rules	Organization resource	User-request
	ParamHarver [11]	Parameter	Suggestion	Statement	Data-mining	Web resource	User-request
	Raychev et al. [12]	Snippet	Generation	Statement	Machine-learning	Web resource	User-request
	Parc [13]	Parameter	Completion	Statement	Data-mining	Organization resource	User-request
	GraPacc [14]	Method	Completion	Project artifact	Data-mining	Organization resource	User-request
	Heinemann et al. [15]	Method	Suggestion	Method body	Data-mining	Organization resource	User-request
	CSCC [16]	Method	Completion	Project artifact	Data-mining	Web resource	User-request
	Holmes et al. [17]	Method	Suggestion	Class body	Heuristic rules	Web resource	User-request
	PBN [18]	Method	Completion	Class body	Machine-learning	Organization resource	User-request
	DroidAssist [19]	API usage	Suggestion	Project artifact	Machine-learning	Organization resource	User-request
	BMNCCS [20]	Method	Completion	Method body	Data-mining	Web resource	Instant-recommend
	Amann et al. [21]	Method	Suggestion	Method body	Other (collaborative filtering)	Web resource	User-request
	Nguyen et al. [22]	API usage	Generation	Method body	Machine-learning	Web resource	User-request

3 Qualitative review

This section describes the process of the qualitative review. The review was conducted by invited engineers to evaluate the effectiveness of the three recommendation categories towards a set of typical recommendation requirements.

3.1 Recommendation requirements

Recommendation requirements are explained in scenarios when an Android developer may require to

accomplish his tasks. Four scenarios are illustrated either as fundamental anticipations or complex requirements.

R1: Instant suggestion and completion. This scenario occurs when a developer does not know the name of parameters or API methods to complete a sentence, or even does not know what to code in the following step. For example, when a developer requires to decide the proper actual arguments of a method he is just invoking, the variables or constants declared beforehand can be recommended in a rapid manner. In addition, when a developer wants to decide which method of an object should be invoked subsequently, the available methods or the predicted ones conforming to the context are required to be recommended.

R2: Program reference. A developer may expect to obtain program codes or snippets they can refer to in order to achieve a functional feature. One scenario of the requirement is to learn the usage of a specific API. In this case, the usage pattern, i.e., the method call sequences of the API can be followed. Furthermore, example codes containing the API method invocation can also be imitated. Another scenario is to learn how to implement a feature involving different API methods from a set of Android applications. These method invocations compose programming conventions that can be recommended to developers.

R3: Third-party library recommendation. Developers always adopt third-party libraries to achieve applications with more attractive features besides the official Android SDKs, and they usually spend much time seeking the anticipated libraries and learn how to use them. These libraries are contributed by active developers which can be obtained from the public repositories. In this requirement scenario, automatically recommendations of the third-party libraries according to the context and expectation is much desired.

R4: SDK-version related API suggestion. There inevitably exist APIs that are changed, deprecated and newly introduced along with the rapid evolution of the Android SDKs. APIs not ensuring backward compatibility support are typically hard to use due to their instability, and API breaking-changes could further introduce bugs in the client codes [23, 24]. In this recommendation requirement scenario, developers are notified about the proper APIs to use when the Android SDK version is specified. Under this circumstances, the backward compatibility can be ensured while the API usage error can be avoided.

3.2 Review process

The review process is designed as a semi-formal meeting which depends on stakeholders to derive a comprehensive evaluation of the different code recommendation techniques towards the different recommendation scenarios. The evaluation result is scored by stars. Three stars is the full score which means the full satisfaction of the recommendation towards a specific requirement. No star, on the contrary, indicates the dissatisfaction of the recommendation or the adoption of the mechanism is meaningless. Under the principle, we invited eight engineers covering junior ones to senior ones from two collaborative software companies to attend the review meeting. All the engineers have experience of developing android applications on Android Studio or Eclipse ADT. The meeting was organized in four phases. First, we introduced the four selected requirements to the engineers. Second, we explained the code recommendations categories including the typical techniques and tools to the engineers. In each category, some of the techniques whose tools can be downloaded and installed were illustrated by means of simple recommendation demos. Others were explained on the theory level and the expected recommendation results were focused. Third, each engineer was invited to give his scores on each recommendation category aiming at the recommendation requirements separately. Along which the star scores, he was also required to propose the reasons of giving out the scores. For example, if a recommendation mechanism was considered to partially satisfy a requirement, he could give out two-star score with an example that might not be solved by the specific recommendation technique. Fourth, we collected all the scores and synthesized the final review result. In this phase, if the evaluation results of a recommendation category differed, the proposed reasons given by the developers were reviewed and discussed. In order to avoid the endless

Table 2 Qualitative review result

	Code searching	Basic recommendation	Advanced recommendation
R1	–	⋆⋆	⋆⋆⋆
R2	⋆⋆	–	⋆⋆
R3	⋆⋆	–	⋆
R4	⋆	⋆	⋆

argument, when the engineers could not achieve an agreement in ten minutes, we made the scores given by more than six engineers as the final decision. If none of the scores met the criteria, the lowest score was selected as the final score.

3.3 Review result

Table 2 illustrates the synthesized qualitative review results towards the four requirements.

For the first recommendation requirement, mechanisms from code searching category were considered meaningless since the granularity of the recommendation results is too small to be worth searching from the engineers' viewpoint. The basic recommendation was agreed to provide engineers the most basic support. Parameters or methods which can be transferred or invoked according to the static type system were listed immediately when engineers required to complete a sentence. For example, IDEs such as Eclipse has the ability to suggest the arguments by matching the type of the parameters between the available variables and the required formal arguments. Advanced recommendation was considered to achieve full satisfaction. The mechanisms from this category are designed to learn from a large amount of exiting applications to collect the reusable knowledge that can be offered to developers [11,12]. Engineers believed they could help to reduce the size of the recommendation list compared with the mechanism of mainstream IDEs.

For the recommendation requirement of program reference, code searching and advanced recommendation were regarded having good support while basic recommendation was agreed not applicable. Mechanisms of advanced recommendation such as DroidAssist [19] enable to extract the API usage patterns from existing applications, thus were thought pretty applicable for the API learning scenario. Code searching allows engineers to construct queries to obtain the reference programs relatively. Although the searching results may be diverse and required to be filtered out further, the category has advantages in providing a wide range of results. For example, Thung et al. [3] proposed an approach to recommend API methods by comparing the textural description of a request with the textural description of learnt API methods. Furthermore, DeepAPI [6] is designed to derive API invocation sequences for a given natural language query. The underlying mechanism is to understand the semantic of the query based on a neural language model so that the recommendation quality can be improved. However, the two categories of recommendation have difficulty in obtaining the reference programs involving several classes which was pointed out by engineers. As an example, drag and drop is a widely used UI-related feature. To implement the feature, developers usually program the drag view and the drop view in separate source files, and assign listeners to both of the views. When an engineer requires recommendation for the drag and drop feature, methods of code searching and advanced categories may derive the result from one of the view classes. Engineers then have to take extra effort to find the complete reference programs.

For the requirement of third-party library recommendation, engineers thought code searching could provide acceptable support while advanced recommendation performed better. However, basic recommendation was regarded not contain the knowledge about third-party libraries so that the related mechanisms could not be adopted to find the external resources. For instance, ActionBar-PullToRefresh is a library available from GitHub. It is easy for developers to obtain the library through formulating a query with the keywords pull to refresh in code search engine. However, developers have to further filter out the most suitable result from the diverse feedbacks. In addition, they also need to follow the instructions to manually integrate the library into the applications. In particular, the integration may be complex when the implementation of the library involves the definition of the layout and logics of the source code. On the other hand, advanced recommendation may provide more accurate results if the usage of the library

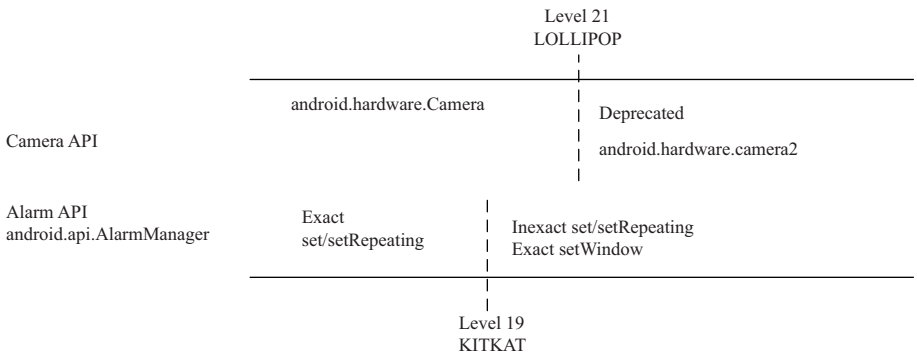

Figure 2 API evolution of different SDKs.

has been learnt. It is also a risk to the developers that the newly released third-party libraries may be ignored because the libraries have not be learnt beforehand.

The recommendation for SDK-version related API usage is critical to the success of released Android apps in the context of rapid SDK upgrading. For example in Figure 2, android.hardware.Camera is deprecated in SDK 21+ but still can be invoked to preserve the backward compatibility. Developers are recommended to adopt android.hardware.camera2 as replacement in the official document since the new API has enhanced capabilities. Another example exhibits the difference of the Alarm API (android.app.AlarmManager) between the versions 19– and 19+. The two methods set and setRepeating changed their behavior from time critical to non-time critical to enable alarm batching across applications. Moreover, a new API method setWindow is introduced to support the non-deferrable alarm. An Android application is compiled based on a specific version of Android SDK which is specified in AndroidManifest.xml. Therefore, if a developer uses the APIs such as the exemplified ones that is not encouraged in the specific SDK version, the application's behaviour may be impacted when the app runs in a compatibility mode [25]. Aiming at this concern, engineers agreed that the three categories of recommendation could all support the requirement in an acceptable level. Code searching helps to retrieve the results by means of the version related querying statements. There may be a defect that the search engine is not SDK version sensitive so that developers cannot clarify whether the APIs have been upgraded. Basic recommendation usually has a tightly relationship with SDKs. It has the ability to notify developers about the API status. For example, editor of Android studio can warn developers about the deprecated API as well as the replacement based on the specified SDK version. Advanced recommendation, on the other hand, may feedback all the related API usages that have been learnt, e.g., both android.hardware.Camera and android.hardware.camera2 can be retrieved, because the underlying mechanism does not take the SDK version into consideration. However, engineers revealed key scenarios that cannot be handled effectively by all of the exiting mechanisms. The most notable is how to improve the API usage for preserving backward compatibility. Taking the camera API as the example, when the minSDKVersion item in AndroidManifest.xml is specified less than 21, a single version of camera API may not cover all the devices with different Android SKD levels. In this case, a suitable solution can be applied to adopt corresponding API methods towards the current device SDK at runtime. In implementation, the solution can be represented by an if-else clause with a condition statement (Build.VERSION.SDK_INT >= 21).

4 Discussion for improvement

In this section, improvements suggestions for state-of-the-art code recommendation techniques are explained. Before the suggestions, we first conclude the special nature of the Android development which may inspire the improvements for Android code recommendation.

4.1 Special nature of Android development

By investigating the daily work of the invited Android engineers, all of them build their apps in IDEs

supporting programming, compiling, testing and releasing of Android applications. Now Eclipse and Android studio are the two mainstream working environments for developers. In any IDE, code recommendation has been one of the indispensable features frequently used by developers. From the brief overview of the code recommendation mechanisms and characteristics, we are sure that existing recommendation approaches can strengthen the IDEs facilitating developers' tasks either as integrated plugins or as theoretical feasible solutions. Conventional approaches are language independent so that they are also effective for Android programming and are able to improve the coding efficiency especially for the novice developers. However, as a specific framework to build mobile apps, Android has a few special nature that makes it different from the traditional Java programs.

First, an Android application involves different types of artifacts maintained by developers other than Java classes. The most important is the AndroidManifest.xml which declares the components of the app and the permissions as well. Other resource files define the UI elements such as the layouts and the shown strings. The UI related design and development now becomes more critical since a user-friendly and easy-to-use application can improve the stickiness of end users.

Second, there are widely-accepted programming conventions in Android development that are encouraged to be used. These conventions are either concluded from the technique perspective or from the business perspective. For example, the official Android documents advise to access GUI only from main thread (UI thread), otherwise may lead to races because Android UI toolkit is not thread-safe [26]. From the perspective of business, one should have a more perfect consideration of the app to make it accepted by end users. For instance, if a developer designed the UI layout (one single view) in the screen portrait mode, she should also continue to design the layout (one main-view and one sub-view) under the screen landscape mode. This kind of design enhanced the exhibition effects to a large extent.

Third, Android applications are component-based and event-driven [27]. The lifetime of the activities and services is managed by the Android framework. In addition, apps are driven by invoking the event handling callbacks as well as the activity lifecycle callbacks through operating on the GUI related objects. These callbacks should be integrated properly in order to ensure the correct behavior.

4.2 Improvement suggestions

Based on the feedback of the qualitative review, we found that developers can eventually obtain the recommendation results by taking different efforts. However, there still exist some requirements that cannot be fully satisfied due to the special nature of the Android development. In view of this situation, we conclude a set of improvement suggestions developers may concern.

First, the scope of the recommendation for Android development could go beyond the code level to other kinds of artifacts. For example, the user interface can be recommended. As a possible solution, user interfaces of applications under use can be recorded and learnt by developers during their daily lives. The collected resources can thus be recommended to developers when they design layouts. Another example is to provide useful information when resolving deprecated Android APIs. Many deprecated API usages remain unresolved since the documents for API evolution often do not have sufficient information [28]. The recommendation of the rationale and the examples of the alternative APIs is able to benefit the developers. Therefore, these kinds of code recommendations can be regarded useful to speed up the design process and even to improve the design quality.

Second, the recommendation results on the code level should also be expanded to involve more than one method, even more than one class. Along with the increasing complexity of Android features, the implementation codes for these features may scatter in different classes or methods. Conventional code recommendation offering snippet of a single method may not establish a overall perspective of the feature implementation, thus affecting the understanding of developers. Therefore, a comprehensive image is necessary. The objective can be achieved by learning from online posts containing snippets from different modules, or by investigating example codes from the perspective of program structure to derive a more complete recommendation. In addition, in order to facilitate developers to understand, the presentation of wider scale of snippets should be suitably designed.

Third, the searching and integration of the public resources should be simplified. With the resources such as third-party libraries upgrading continuously, developers may take more efforts on finding suitable libraries than completing a program sentence. Using code searching mechanisms, developers construct queries and then learn how to integrate the feedback libraries through reading the guidance. A third-party library usually includes a jar package implementing the feature, a group of extended definitions applied on the resource files, and possibly exemplified programs. In this case, a recommendation tool which is able to automatically download and integrate the library into the application according to the querying results will be widely appreciated.

Forth, code recommendation mechanisms should emphasize the context of the entire project to a greater extent. As a specific development paradigm, the Android artifacts such as configuration files besides Java codes may have an impact on the results of recommendation. For instance, Android SDK version is an important factor and the SDK-related API usage is influenced by the value. Therefore, advanced algorithms taking input the configuration information could improve the recommendation quality.

5 Customizable tool framework

Recommendation tools from industry or academic are designed independently and cannot be integrated easily. In this section, we propose a customizable tool framework as a possible improvement for code recommendation in Android development. In the following subsections, the components constituting the framework are first described and then the customization process is presented.

5.1 Components of the tool framework

Implementing a code recommendation tool as an IDE plugin is more practical than implementing it as an independent system since the plugin can be tightly integrated with the programming environment. Therefore, we design the tool framework in a plugin style which is illustrated in Figure 3. The framework contains components grouped in four different layers which are named as the context layer, the engine layer, the UI layer and the training layer. We describe the constitution of the framework from the layer perspectives as follows.

• The context layer is responsible for obtaining and monitoring the programming context during the recommendation process. There are two important components designed towards different working scenarios. First, the context recognizer is implemented to capture the context, especially the surrounding programs, once a recommendation which requires to analyze the context is activated. According to the specified capturing requirements, the component decides its recognition boundary based on the recommendation point i.e., the location in the editor which requires suggested codes. The program statement, the method body even the class body can be captured depending on the recommendation mechanism applied. In addition, the recognizer also helps to capture the project-related information from the AndroidManifest.xml such as the target SDK version. These context information is useful for performing the basic recommendation and the advanced recommendation successfully. On contrast, the context monitor is sensitive to the changes of the programming context in order to drive the instant recommendation. Under this circumstance, the component accesses and obtains the editor status continuously and notifies developers when the context change satisfies predefined rules. On the implementation level, both the two components relate tightly with the development environment. It is feasible for them to get aware of the operations taken by developers by means of invoking APIs provided by IDEs. Furthermore, the surrounding programs and the application resources can also be acquired by means of other groups of IDE APIs.

• The engine layer contains a set of components to realize the specific recommendation tasks. This layer locates between the context layer and the UI layer. It takes input the current programming context from the context recognizer component when receiving the users' requests delivered from the UI layer. The recommendation components inside the layer can also be invoked when it is activated by the context

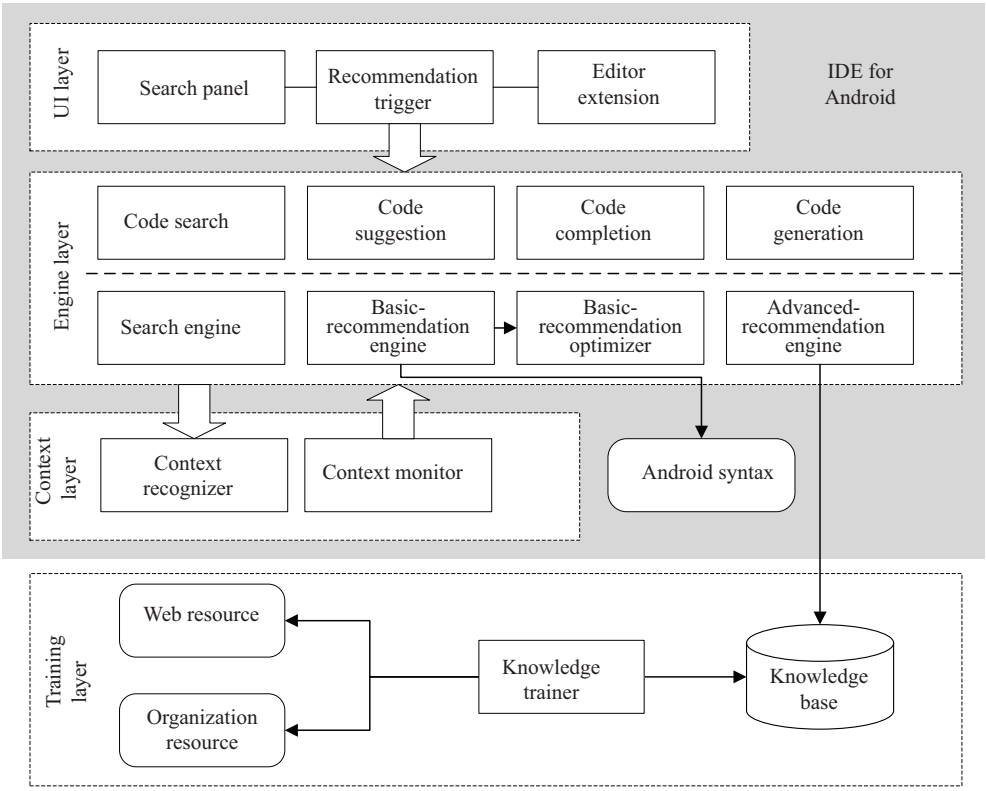

Figure 3 Customizable tool framework.

monitor in an instant recommendation manner. The recommendation results are to be displayed in the user interface. The engine layer is further divided into two sub-layers.

In the upper sub-layer, we design four interface components to support the four types of code recommendations, i.e., code search, code suggestion, code completion and code generation. These interfaces provide APIs which are accessible by the UI layer and can be invoked accordingly. On the other hand, there are recommendation engines in the lower sub-layer that realize the three groups of recommendation mechanisms, i.e., code searching, basic recommendation and advanced recommendation separately.

The search engine takes the query statement as input and obtains the results from the open resources. In addition, the search engine can also access the current programming context to improve its search accuracy by enriching the keywords or semantic of the query statement.

The basic-recommendation engine is usually embodied in modern IDEs. The mechanism included relies on the fundamental Android SDKs, Java SDKs and other third-party APIs to provide the syntax-level code suggestions. Furthermore, a basic-recommendation optimizer can be applied to improve the suggestion quality. For instance, the suggestion list can be prioritized in which the most frequently used APIs are located in the top.

The advanced-recommendation engine is responsible for implementing the recommendation by means of the advanced techniques which have been briefly summarized in the previous section. The most popular techniques proposed by academic publications relate to the application of data mining and machine learning. Therefore, the successful implementation of the component depends on the learnt programming patterns or code snippets. These resources are designed to be collected in the training layer of the tool framework.

• The UI layer contains the user interface of a tool visible to the developers. The core component is the recommendation trigger which usually exhibits as a button or a context menu integrated in the development environment. There are other two UI components related to the recommendation trigger in order to satisfy the recommendation requirements. The search panel can be regarded as an independent constituent of the IDE from which the code searching mechanism can be applied. The trigger then can be

activated from the search panel after a developer typed in the querying statement in the searching scenario. The editor extension is designed as a upgrading to the programming editor by enhancing the ability of code recommendation especially under the basic recommendation and the advanced recommendation scenarios. For example, the activation of different kinds of recommendation can be supported by specific shortcuts. In addition, the recommendation results can be presented in more intuitive manner rather than a plain list.

• The training layer is not included in the IDE plugin scope however is also an important constituent of the tool framework. It is responsible for learning knowledge from open resources and organizing the learnt patterns or snippets into a knowledge base. The knowledge trainer adopts the techniques such as data mining and machine learning and takes effect by user request or even in a sustained manner. The learning objectives can be the online web resources, e.g., programs from GitHub and discussion threads from StackOverflow, or organizational resources, e.g., the applications developed beforehand by the same developer or by the other development teams.

5.2 Customization of the tool framework

A toolsmith can preserve all the components in tool framework to construct a comprehensive recommendation tool. He can also derive a simple one according to the requirements from developers or organizations by making decision of the inclusion of specific components. Once the customization towards the framework is performed, the preserved components constitute the tool architecture specifying the development goal.

The customization towards the components of the tool framework is performed in two aspects. On one hand, the existence of the specific components can be determined. Toolsmith can decide whether or not to remove a component according to the recommendation requirements. On the other hand, the implementation of a specific component can be determined. Toolsmith can make the decision which kinds of techniques to be applied to make the component effective.

The customization process can be performed towards all the layers as listed in the following perspectives.

• Customization to the UI layer. The constituent of the user interface can be determined depending on the developers' requirements for code recommendation, i.e., which recommendation scenarios are required during their programming process. For example, the search panel is included when a developer anticipates to acquire related information or resources by constructing a query statement. The panel is removed if the requirement is not considered. On the other hand, if the basic recommendation and the advanced recommendation are both involved, how to trigger the suitable recommendation action should be considered and configured. For instance, a developer can configure the tool to activate the corresponding recommendation actions through clicking specific buttons or menus. In addition, the presentation styles of the recommendation results can also be configured.

• Customization to the engine layer. The four interface components in the upper sub-layer are determined according to the recommendation objectives. The signatures of the interfaces can be further decided to ensure the correct communication between the UI layer and the engine layer. The components in the lower sub-layer are determined according to the recommendation mechanisms developers select. In theory, all of the interface components could apply the basic recommendation or the advanced recommendation. In particular, the basic recommendation engine is always included in almost all of the modern IDEs. The optimizer engine, however, is adopted when required. In addition, the underlying techniques applied to implement the advanced recommendation engine should be determined before the tool is constructed.

• Customization to the context layer. The inclusion of the components in the context layer is decided from the decision transferred from the higher layer or delivered directly from the tool users. For example, when the recommendation technique does not take the context into consideration, the context recognizer is meaningless and can be removed. Furthermore, the context monitor is included when an instant recommendation feature is required. In addition, the techniques supporting the recognizing and

the monitoring can be chosen by the tool developers. Under the circumstances, how to capture the programming context accurately and efficiently may be a core concern, and how to define the proper rules to activate the instant recommendation from the context monitor should also be carefully considered.

• Customization to the training layer. The training layer can be configured to decide how to learn the knowledge and from which the knowledge can be learnt. When an organization decides its dependent resources, the origination of the application of machine-learning and data-mining can be clarified. On the other hand, the selected techniques of machine-learning or data-mining determines what kind of data structures can be formed to store the learnt programming patterns or code snippets.

However, the customization is not performed independently in separate layer. There exists customization dependencies between the different layers. For example, the implementation of code search mechanism involves the search panel in the UI layer, the code search interface and the search engine in the engine layer. The existence of these components is usually bound together. Therefore, the actual customization process can be performed from top to bottom, i.e., customizing the UI layer first then the customization decision is propagated to the other layers to determine the related components correspondingly.

6 Conclusion

In Android development, code recommendation has been widely accepted. Existing modern IDEs provide basic recommendation ability mainly based on the static type system. Meanwhile, academic researchers in this field continue to offer advanced solutions to improve the quality of recommendations. In this paper, we characterize the mechanisms of selected publications based on a multidimensional categorization framework, and then classify the mechanisms into three categories. A qualitative review towards the application of typical code recommendations for different recommendation requirements was performed. Results show that state-of-the-art code recommendations can benefit developers but still have deficiencies in meeting specific development scenarios. Furthermore, we indicate the improvement directions from the method aspect as well as the tool aspect. A set of improvement suggestions are proposed to alleviate the deficiencies and to enhance the recommendation ability. The tool framework, which is composed of a set of components, helps toolsmith to organize the three recommendation categories in a single architecture.

As for the future work, we plan to carry out further research based on the improvement suggestions we concluded for recommendation facilities. It may point to a smart IDE which provides more automatic features for Android developers. In the era of big data, the capability of the features can be enhanced by mining patterns from the large scale of development data which can be named as "big code" in a similar way. The big code covers the programming information from the spatial dimension as well as the time dimension. The spatial dimension involves the program codes of existing applications. Recommendations then can be derived in a more accurate level by mining and learning the programming patterns from the applications with similar contexts. The data of the time dimension originates from the programming history of a developer. The addition, deletion and modification of classes, methods or other kinds of resources represent the decisions of a developer during the development or maintenance phase. Therefore, the experience could also be learnt and recommended in a similar working scenario.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant No. 61402113).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Wu J, Shen L, Guo W, et al. How is code recommendation applied in Android development: a qualitative review. In: Proceedings of the International Conference on Software Analysis, Testing and Evolution, Kunming, 2016. 30–35
2 Su X, Khoshgoftaar T M. A survey of collaborative filtering techniques. Adv Artif Intell, 2009, 4: 2
3 Thung F, Wang S, Lo D, et al. Automatic recommendation of API methods from feature requests. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, Silicon Valley, 2013. 290–300

4 Jiang H, Nie L, Sun Z, et al. Rosf: leveraging information retrieval and supervised learning for recommending code snippets. IEEE Trans Serv Comput, 2016, 9: 1–13
5 Nie L, Jiang H, Ren Z, et al. Query expansion based on crowd knowledge for code search. IEEE Trans Serv Comput, 2016, 9: 771–783
6 Gu X, Zhang H, Zhang D, et al. Deep API learning. In: Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering, Seattle, 2016. 631–642
7 Omar C, Yoon Y S, LaToza T D, et al. Active code completion. In: Proceedings of the International Conference on Software Engineering, Zurich, 2012. 859–869
8 Schafer M, Sridharan M, Dolby J, et al. Effective Smart Completion for JavaScript. Technical Report RC25359. 2013
9 Omori T, Kuwabara H, Maruyama K. Improving code completion based on repetitive code completion operations. Inf Media Tech, 2015, 10: 210–225
10 Zhang C, Yang J, Zhang Y, et al. Automatic parameter recommendation for practical API usage. In: Proceedings of the International Conference on Software Engineering, Zurich, 2012. 826–836
11 Li L, Bissyandé T F, Klein J, et al. Parameter values of Android APIs: a preliminary study on 100000 apps. In: Proceedings of International Conference on Software Analysis, Evolution, and Reengineering, Osaka, 2016. 1: 584–588
12 Raychev V, Vechev M, Yahav E. Code completion with statistical language models. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, Edinburgh, 2014. 49: 419–428
13 Asaduzzaman M, Roy C K, Monir S, et al. Exploring API method parameter recommendations. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution, Bremen, 2015. 271–280
14 Nguyen A T, Nguyen T T, Nguyen H A, et al. Graph-based pattern-oriented, context-sensitive source code completion. In: Proceedings of the International Conference on Software Engineering, Zurich, 2012. 69–79
15 Heinemann L, Bauer V, Herrmannsdoerfer M, et al. Identifier-based context-dependent api method recommendation. In: Proceedings of the European Conference on Software Maintenance and Reengineering, Szeged, 2012. 31–40
16 Asaduzzaman M, Roy C K, Schneider K A, et al. Cscc: simple, efficient, context sensitive code completion. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution, Victoria, 2014. 71–80
17 Holmes R, Murphy G C. Using structural context to recommend source code examples. In: Proceeding of the IEEE International Conference on Software Engineering, Saint Louis, 2005. 117–125
18 Proksch S, Lerch J, Mezini M. Intelligent code completion with Bayesian networks. ACM Trans Softw Eng Methodol, 2015, 25: 3
19 Nguyen T T, Pham H V, Vu P M, et al. Recommending API usages for mobile apps with hidden markov model. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, Lincoln, 2015. 795–800
20 Bruch M, Monperrus M, Mezini M. Learning from examples to improve code completion systems. In: Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Amsterdam, 2009. 213–222
21 Amann S, Proksch S, Mezini M. Method-call recommendations from implicit developer feedback. In: Proceedings of the International Workshop on CrowdSourcing in Software Engineering, Hyderabad, 2014. 5–6
22 Pham H V, Vu P M, Nguyen T T. Learning API usages from bytecode: a statistical approach. In: Proceedings of the International Conference on Software Engineering, Austin, 2016. 416–427
23 McDonnell T, Ray B, Kim M. An empirical study of API stability and adoption in the android ecosystem. In: Proceedings of the IEEE International Conference on Software Maintenance, Eindhoven, 2013. 70–79
24 Linares-Vásquez M, Bavota G, Bernal-Cárdenas C, et al. API change and fault proneness: a threat to the success of Android apps. In: Proceedings of the Joint Meeting on Foundations of Software Engineering, Saint Petersburg, 2013. 477–487
25 Almeida M, Bilal M, Blackburn J, et al. An empirical study of android alarm usage for application scheduling. In: Proceedings of the International Conference on Passive and Active Network Measurement, Heraklion, 2016. 373–384
26 Lin Y, Cosmin R, Danny D. Retrofitting concurrency for Android applications through refactoring. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, HongKong, 2014. 341–352
27 Rountev A, Yan D. Static reference analysis for GUI objects in Android software. In: Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization, Orlando, 2014. 143–153
28 Ko D, Ma K, Park S, et al. API document quality for resolving deprecated APIs. In: Proceedingso of the Asia-Pacific Software Engineering Conference, Jeju, 2014. 2: 27–30