A Feature-Oriented Mobile Software Development Framework to Resolve the Device Fragmentation Phenomenon for Application Developers in the Mobile Software Ecosystem

Younghun Han[1(✉)], Gyeongmin Go[1], Sungwon Kang[1], and Heuijin Lee[2]

[1] Department of Computer Science, KAIST, Deajeon, Republic of Korea
{younghun.han, imarch, sungwon.kang}@kaist.ac.kr
[2] Samsung Electronics, Suwon, Republic of Korea
koslee@kaist.ac.kr

Abstract. In the current mobile software environment, the device fragmentation phenomenon causes a serious problem to the mobile software ecosystem stakeholders. Since mobile manufacturers make various differentiated hardware components for product differentiation around strategically selected open platforms, a huge number of devices are produced each year. Since the application developers have to verify manually whether the developed application is compatible with specific devices, a tremendous burden is put on the application developers. To solve this problem, we propose a feature-oriented mobile software development framework and implement as part of it an automated tool for compatibility verification. To evaluate our framework, we conduct a case study with 10 devices and 21 features from the real world. The result of the case study indicates that a significant effort reduction can be achieved by using our framework.

Keywords: Mobile software ecosystem · Device fragmentation phenomenon · Feature model · Android

1 Introduction

"A Software ECOsystem (SECO) is the interaction of a set of actors on top of a common technological platform that results in a number of software solutions or services" [1]. A Mobile Software ECOsystem (MSECO) is a software ecosystem that consists of a set of actors where the actors interact with each other through a common technological platform that enables a number of mobile applications to simultaneously run on mobile devices such as smart phone, tablet and smart watch [2].

In the past, the mobile manufacturers developed mobile software using in-house platforms whereas today the majority of manufacturers develop mobile software using open platforms such as Android. Since mobile manufacturers develop various differentiated hardware components for device differentiation around strategically selected open platforms, there are a huge number of devices [3] in the current mobile software environment. This is called the *device fragmentation phenomenon* [4].

Y. Zhang et al. (Eds.): CloudComp 2015, LNICST 167, pp. 189–199, 2016.
DOI: 10.1007/978-3-319-38904-2_20

According to the survey in [3], conducted by Open Signal, as of 2013, there exist approximately 682,000 Android devices. These devices are classified into 11,868 species by various criteria, such as manufacturer, version of API, display resolution etc. Moreover, since the number of mobile devices increase steadily every year, devices fragmentation phenomenon will be worse in 2015. The device fragmentation phenomenon affects four stakeholders in the software ecosystem: the platform providers, the end-users, the application developers, and the market service providers. Among them, especially, the application developers will have difficulty in verifying compatibility whether developed application will run on specific devices.

To relieve the application developers of the burden arising from the device fragmentation phenomenon, we propose the *Feature-Oriented Mobile Software (FOMS) development framework* and implement as part of the framework a tool that automates compatibility verification. Our framework is the result of viewing the device fragmentation phenomenon from the perspective of application developers. It is composed of the domain part and the application part. In the domain part, domain experts collect the features of devices and analyze it using the software product line approach. In the application part, developers take advantage of the artifacts constructed by domain experts for application development. To evaluate our framework, we conducted a case study with 10 devices and 21 features used in real world. It shows that a significant efforts reduction is possible by using our framework rather than the traditional approach.

This paper is organized as follows. In Sect. 2, we introduce background for understanding our framework. In Sect. 3, we describe the FOMS development framework for resolving the device fragmentation phenomenon from the perspective of application developers. In Sect. 4, we introduce our tool for automation of compatibility verification. In Sect. 5, we conduct a case study. In Sect. 6, we discuss related works dealing with the device fragmentation phenomenon. Lastly, we conclude with our contributions and future works in Sect. 7.

2 Background

In this section, we introduce the device fragmentation phenomenon, the concept of feature model and a classification of mobile device features.

2.1 The Device Fragmentation Phenomenon

The stakeholders of the mobile software ecosystem are the platform provider, the end-user, the application developer and the market service provider. To these stakeholders the device fragmentation phenomenon causes the following problems. First, the platform provider has difficulty in managing platform evolution since various versions of a platform exist. Second, the end-user has difficulty in finding an application that is compatible with the user device. Third, the application developer has difficulty in testing developed software because they do not know the compatible target devices. Lastly, the market service provider has difficulty in providing exact application since they do not know precisely what features each device has.

2.2 Feature Model

Feature model was first introduced in the feature oriented domain analysis (FODA) [5] method in 1990 by Kang et al. Since then, feature model has been widely used to model relationships between features including commonality and variability of features by many Software Product Line Engineering [6] researchers. In the early stage of software product line development, the feature modeling technique is very instrumental in identifying and analyzing reusable parts. There are many kinds of feature models including FODA, FORM [7], Cardinality-based FM [8], etc. Among them, we will use the basic feature model in [9], which includes mandatory, optional, OR and alternative feature and constraints (i.e. the require constraint and the exclude constraint).

2.3 Classification of Mobile Device Features

Lee and Kang [2] classified features in mobile devices into four groups: platform features, manufacturer features, regional features and differentiated features. A platform feature is a feature distributed by a platform vendor. Examples of a platform feature are GPS, Bluetooth and LTE. A manufacturer feature is a feature made by the mobile manufacturer to compete with other mobile companies. A regional feature is a feature used for a specific country or a network operator. Lastly, a differentiated feature is a unique feature for product differentiation. Examples of a differentiated feature are IrDA, S-pen and Finger scan. In this paper, the domain experts will construct a feature model using a classification method and the basic feature model technique.

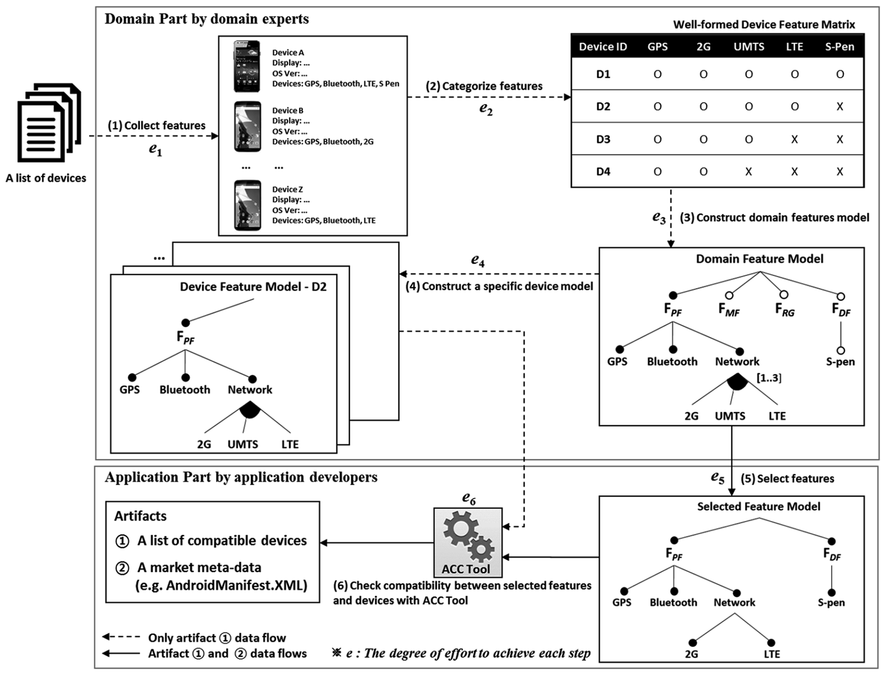
3 A Feature-Oriented Mobile Software (FOMS) Development Framework

In this section, Sect. 3.1 describes our FOMS development framework for resolving the device fragmentation phenomenon and Sect. 3.2 explains the process of the framework with a simple example.

3.1 Methodology

Figure 1 shows our FOMS development framework. It consists of two parts: the domain part and the application part. In the domain part, the domain experts such as software product line experts and platform vendors are in charge of data collections and construction of a feature model. In the application part, the application developers derive useful information for application development from the constructed feature model.

The FOMS development framework has six steps. In steps (1) and (2), device information is gathered from a specific source such as a description file, which describes device information in a format specified by the market service provider.


Well-formed Device Feature Matrix

Device ID	GPS	2G	UMTS	LTE	S-Pen
D1	O	O	O	O	O
D2	O	O	O	O	X
D3	O	O	O	X	X
D4	O	O	X	X	X

Domain Part by domain experts
A list of devices
(1) Collect features
e_1
Device A
Display: ...
OS Ver: ...
Devices: GPS, Bluetooth, LTE, S Pen
Device B
Display: ...
OS Ver: ...
Devices: GPS, Bluetooth, 2G
Device Z
Display: ...
OS Ver: ...
Devices: GPS, Bluetooth, LTE
(2) Categorize features
e_2
e_3
(3) Construct domain features model
Domain Feature Model
F_{PF}
F_{MF}
F_{RG}
F_{DF}
GPS
Bluetooth
Network
S-pen
[1..3]
2G
UMTS
LTE
e_4
(4) Construct a specific device model
Device Feature Model - D2
Application Part by application developers
e_5
(5) Select features
Selected Feature Model
e_6
ACC Tool
(6) Check compatibility between selected features and devices with ACC Tool
Artifacts
① A list of compatible devices
② A market meta-data (e.g. AndroidManifest.XML)
Only artifact ① data flow
Artifact ① and ② data flows
※ e : The degree of effort to achieve each step

Fig. 1. The procedure of the FOMS development framework

To extract the features of a device, the domain experts can take advantage of description file. Each piece of device information should be given in a well-organized form, such as matrix, table or database.

In steps (3) and (4), the domain experts construct the feature model from the Device Feature Matrix. Since the feature model has a hierarchical tree structure, the domain expert should know the characteristics of each feature of a device. All of the features are categorized into the four groups using the classification method explained in Sect. 2.3. As a result, two feature models are constructed: Domain Feature Model and Device Feature Model. The domain feature model represents all device information while the device feature model represents feature information of a specific device.

In step (5), the application developers can select various features in the domain feature model following the application specification of what the developer should develop.

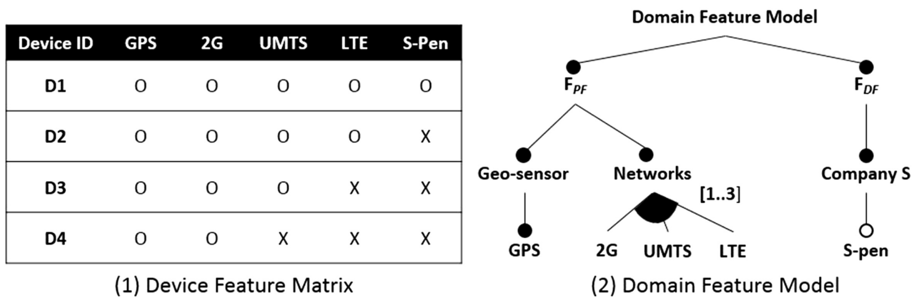
In step (6), compatibility between the selected feature model and each device feature model should be checked. To that end, the application developer can use the ACC (Automated Compatibility Calculation) tool that we developed as part of the FOMS development framework, which automatically check compatibility between them.

At the end of the process, the framework generates two artifacts: a list of compatible devices and a market meta-data. The market meta-data will take advantage of the fragments of AndroidManifest.xml for checking compatibility between developed application and specific devices in Google Play Store.

3.2 A Conceptual Example for the FOMS Development Framework

An example of a well-organized device feature matrix is given in Fig. 2(1). Since GPS is a kind of geographical sensor, the parent feature of GPS can be named as "Geo-sensor". In addition, since all devices have the GPS feature, it should be a mandatory feature. The features 2G, UMTS, and LTE are network features. So their parent feature can be named as "Networks", which can have at least one and at most three sub-features.

The domain feature model in Fig. 2(2) shows the feature model constructed from Fig. 2(1). To model the observations in the previous paragraph that the "Networks" feature is made up of three features (i.e. 2G, UMTS and LTE), Fig. 2(2) uses the *inclusive_or* constraint, in this case with the cardinality of *inclusive-or* of "from 1 to 3". The categories "Geo-sensor" and "Networks" can be classified into Platform Feature by [2]. On the other hand, since S-Pen is a feature for product differentiation by a Company S, it can also be classified into Differentiated Feature by [2]. The domain experts should name the parent feature of S-Pen as Company S and this feature is an optional feature.



Device ID	GPS	2G	UMTS	LTE	S-Pen
D1	O	O	O	O	O
D2	O	O	O	O	X
D3	O	O	O	X	X
D4	O	O	X	X	X

Fig. 2. Device feature matrix and domain feature model

After the domain feature model is constructed, the developers can select the features in Fig. 2(2). In our example, if a developer selects four features, for example GPS, 2G, UMTS, and LTE, which can be described as:

Table 1. Compatibility calculation for the example in Table 1 and C_e

Devices	Platform feature				Differentiated feature	Compatibility calculation with C_e
	GPS	2G	UMTS	LTE	S-Pen	
D1	O	O	O	O	O	**O**
D2	O	O	O	O	X	**O**
D3	O	O	O	X	X	**X**[*]
D4	O	O	X	X	X	**X**[**]

[*]LTE is not supported by D3
[**]UMTS and LTE are not supported by D4

$$C_e = \{\text{GPS},\ \text{2G},\ \text{UMTS},\ \text{LTE}\}$$

To support development, the framework produces two pieces of information: a list of compatible devices and a market meta-data. Table 1 shows that D1 and D2 are compatible while D3 and D4 are not. In addition, we will describe a detailed description of the market meta-data in Sect. 5.

4 The Automation of Compatibility Calculation

As part of the FOMS development framework, we implemented a tool ACC and posted it at http://salab-intra.kaist.ac.kr/FOMS. The language used for the server side of ACC is PHP. ACC consists of two components: one for domain feature model configuration and the other for application development support. Developers can get application development support by making a device configuration for an application in the former component after pressing 'submit' button. The application development support component consists of three sections: a list of compatible devices, a list of incompatible devices and auto-generated market meta-data, which is applied to AndroidMenifest.xml. With the result component, the developers can notify which devices are compatible with the application. Moreover, the developers can paste auto-generated XML code fragments directly into the development asset, i.e. AndroidManifest.xml.

The ACC tool uses the set operation *union* ($\cup$) for implementing compatibility calculation as in the formula F1. As an example for using ACC, if a set of features in device D (i.e. Device Feature Model) is compatible for given selected features C (i.e. Selected Feature Model), then C must be subset of D and F1 should be satisfied.

$$D \cup C = D \tag{F1}$$

To implement the compatibility calculation, the feature set of device information is represented as a binary scheme as exemplified in Fig. 3. A bit location represents a feature. The value indicates existence or non-existence of a feature, with "1" representing that the device or configuration have the feature and "0" representing that the device or configuration does not have the feature. For example, in Fig. 3, ①, ②, ③ and ④ each represents a feature. The value 1001 means that features 1 and 4 exist but features 2 and 3 do not.

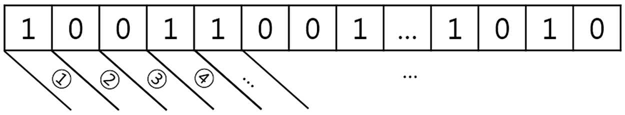


Fig. 3. A binary scheme for representing device information

For compatibility calculation, the condition whether the given configuration satisfies a compatibility F1 or not, can be defined as F2.

$$(d + c) \oplus d = 0 \tag{F2}$$

where **d** is a binary scheme for a specific device **D**, and **c** is the configuration for device **C** given by the developer.

When OR operation is applied to **d** and **c**, if **C** is subset of **D**, then it will produce value **d**. Since XOR gives 0, if two given operands have the same value, the result is 0. The meaning of the result with 0 from this calculation is that the device and given configuration is compatible. Table 2 shows an example of compatibility calculation. Suppose that the domain has four features, each named as "Feature *n*". Then if the developer gives device configuration with the value 1101 (Features 1, 2, and 4), the compatibility calculation results become as in Table 2, which shows that devices A and B are compatible with the configuration that comes from the user. However, it also shows that C and D are incompatible.

Table 2. Device compatibility calculation

Device name	Device A	Device B	Device C	Device D
d	0xF (1111)	0xD (1101)	0x9 (1001)	0x2 (0010)
c	0xD (1101)	0xD (1101)	0xD (1101)	0xD (1101)
Calculation	1111 + 1101 ⊕ 1111 —— 0000	1101 + 1101 ⊕ 1101 —— 0000	1001 + 1101 ⊕ 1101 —— 0100	0010 + 1101 ⊕ 1111 —— 0010
Result	Compatible	Compatible	Incompatible	Incompatible

5 A Case Study

For the device feature tree, we used a well-formed table, the appendix table in [2], as an input. We classified 21 features and represented its actual values in binary. The identified features were 21, and we limited the number of devices to 10. Table 3 shows the result of classifications.

There are 21 identified features, and we limited the number of devices to 10. There are three identified categories: Platform Feature (PF), Manufacturer Feature (MF), and Differentiated Feature (DF). PF is further classified into four categories: Graphic Resolution with five features, Geographic Devices with 2 features, Network with three features and Wireless Network with three features. MF is classified into one category, touch panel. Since we identified the touch panels that are included in all devices but the type of touch panel differs from manufacturer to manufacturer, those features are

Table 3. Classification of 21 features and their binary values

Category	Classification	Features	V: hex value (true)	S: count of shift
Development information	*API version*	Version 4	0x4	19
		Version 3	0x2	
		Version 2	0x1	
Platform feature	*Graphic resolution*	HD	0x10	14
		WXGA	0x8	
		WVGA	0x4	
		WQVGA	0x2	
		QVGA	0x1	
	Geo-sensor devices	GPS	0x2	12
		Accelerometer	0x1	
	Networks	2G	0x4	9
		UMTS	0x2	
		LTE	0x1	
	Wireless network	NFC	0x0	2
		IrDa	0x4	
		Bluetooth	0x2	
Manufacturer feature	*Touch*	Common touch	0x1	5
		TouchWiz	0x4	
		Optimus UI	0x2	
		Sense UI	0x1	
Differentiated feature	*Company S*	S Pen	0x2	0
	Company L	Back hold	0x1	

Table 4. Device binary code for compatibility verification

Device ID	D1	D2	D3	D4	D5
Code	0x1FFF14	0x1FFF1C	0x1FFF9E	0x1FFF9A	0x1FFF55
Device ID	D6	D7	D8	D9	D10
Code	0x1FFF90	0x1FFF14	0x046404	0x04FC30	0x1FF29E

classified into Manufacturer Feature. DF is classified into 2 company names with 1 feature for each. Furthermore, for the market meta-data auto-generation, we added a new category “Development Information”.

For convenience, we divided features by category and added the count of shift to left that is required to represent the actual value. The formula for getting actual decimal value is as follows:

$$R = V \cdot 2^S$$

In the case of the HD graphic resolution feature, the hex value for it is as follows:

$$0x10 \cdot 2^{14} = 0x10 \ll 14 = 0x40000$$

With this feature value, we represented 10 devices as a sequence of 21 bits as shown in Table 4. We named each device with alphabet D and a number, i.e. D1 through D10. One of the data we used is bit representation for D1, which is 0x1FFF14. This means that API versions are compatible with 2 $\sim$ 4 (first three bits from left is 111) and all graphic resolutions are covered. However, it also says that the "S-Pen" feature and the "Back hold" feature are unusable on this device because the last two bits are 00.

We configured the domain feature model with the following feature set:

$$\begin{gathered} E = \{\text{ API Version 4, HD, WVGA, GPS,} \\ \text{Accelerometer, Common touch panel, Bluetooth }\}. \end{gathered}$$

The result showed that compatible devices for this configuration are D1, D2, D3, D5 and D7. In addition, the auto-generated *SDK version* tag and set of *permission* tags of device usage for a market meta-data, AndroidManifest.xml. Developer should copy & paste to the AndroidManifest.XML which belongs to developing application. To evaluate our approach, the required efforts for both the traditional approach and the FOMS development framework should be measured quantitatively for comparison.

The traditional approach requires developers to verify compatibility and write market meta-data in manual. If changes occur to an application, developers should verify it with *n* devices. Also they should verify compatibility and apply changes to market meta-data *n* times. Let define the required efforts for verifying compatibility to be v and those for giving changes on market meta-data to be δw. Moreover, if there are *m* comparisons, the overall efforts become *m* times bigger. The total required efforts are as in F3.

$$E_t = m \cdot \sum_{i=1}^{n} (v_i + \delta w_i) \tag{F3}$$

To measure the performance of our FOMS development framework, let us define the efforts required for step *n* as e_n. Steps 1 $\sim$ 4 are required only once even if there are many devices. Therefore, they are independent from *m*. For step 5, there can be *m* configurations. In the case of step 6, the compatibility verification work is automated to ACC. Therefore, e_6 is ignored. Thus, the formula for the overall required efforts becomes F4.

$$E_{FOMS} = \sum_{i=1}^{4} e_i + m \cdot e_5 \tag{F4}$$

To show the usefulness of our approach, the result of F4 should be smaller than F3. F5 shows the interaction formula between the traditional approach and FOMS.

$$\sum_{i=1}^{4} e_i + m \cdot e_5 < m \cdot \sum_{j=1}^{n} (v_j + \delta w_j) \tag{F5}$$

However, in the perspective of application developers, the required efforts for steps $1 \sim 4$ are constants. Therefore, e_5 are independent from the number of devices *n*. F6 shows F5 in the big-O notation.

$$O(m) < O(m \cdot n) \tag{F6}$$

6 Related Works

As the works that address the device fragmentation phenomenon, there are only a few methods developed in the past and they all approached it from the perspectives of the platform provider and the market service provider.

Google Android is an open source platform that is based on the embedded Linux and a modified JAVA virtual machine [10]. In the case of open platform, due to the features arbitrarily appended by the device manufacturer, it is hard to avoid device fragmentation [2]. Therefore, to solve this problem, Google distributes both the device compatibility policy [11] and the application filtering policy [12]. However, the traditional platform-centric method handles only the features of platform vendors without considering unique features of devices, such as a device-manufacturer's feature and a device-differential feature [2]. To solve this problem, [2] proposed a device-centric method that considers all features of existing devices. The above methods tried to solve device fragmentation phenomenon from the market service perspective. However, in order to eliminate the problem of device fragmentation phenomenon in a fundamental way, its solution should be considered from the development perspective, as was done in this paper, since application developers do not know target devices and will develop software that runs on the reference terminal such as Google Nexus 5, Nexus 7 and Nexus S.

7 Conclusion

The FOMS development framework relieves the application developers of the burden of manually verifying device compatibility that arises from the device fragmentation phenomenon by providing a systematic process and the automated compatibility calculation tool ACC. According to the process, a domain feature model is first constructed and device feature models that can be commonly used by all application developers are constructed from it. Then the developer of a specific application can efficiently select from the domain feature model a feature model corresponding to the application and use the tool to produce a list of compatible devices automatically from the selected feature model and the device feature models.

To evaluate our approach, we conducted a case study with 10 devices and 21 features. We also presented evaluation formulas and showed the proposed framework is more efficient for mobile application development than the traditional approach by resolving device fragmentation phenomenon from the perspective of application developers.

For future works, we plan to integrate our framework into an existing IDE such as Eclipse and RmCRC IDE [13]. Furthermore, we are going to resolve device fragmentation phenomenon also from the viewpoints of the end-user and the platform provider.

Acknowledgments. This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015-H8501-15-1015) supervised by the IITP (Institute for Information & communications Technology Promotion).

References

1. Manikas, K., Hansen, K.M.: Software ecosystems–a systematic literature review. J. Syst. Softw. **86**, 1294–1306 (2013)
2. Lee, H., Kang, S.W.: An efficient application-device matching method for the mobile software ecosystem. In: 2014 21st Asia-Pacific Software Engineering Conference (APSEC), pp. 175–182. IEEE (2014)
3. Android Fragmentation Report, July 2013. http://opensignal.com/reports/fragmentation-2013/
4. Park, J.-H., Park, Y.B., Ham, H.K.: Fragmentation problem in android. In: 2013 International Conference on Information Science and Applications (ICISA), pp. 1–4. IEEE (2013)
5. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. DTIC Document (1990)
6. Böckle, G., van der Linden, F.J., Pohl, K.: Software Product Line Engineering: Foundations. Principles and Techniques. Springer Science & Business Media, Heidelberg (2005)
7. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: a feature-oriented reuse method with domain-specific reference architectures. Ann. Softw. Eng. **5**, 143–168 (1998)
8. Czarnecki, K., Kim, C.H.P.: Cardinality-based feature modeling and constraints: a progress report. In: International Workshop on Software Factories, pp. 16–20 (2005)
9. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: a literature review. Inf. Syst. **35**, 615–636 (2010)
10. Butler, M.: Android: changing the mobile landscape. IEEE Pervasive Comput. **10**, 4–7 (2011)
11. Device Compatibility. http://developer.android.com/guide/practices/compatibility.html
12. Filters on Google Play. http://developer.android.com/google/play/filters.html
13. Nguyen, H.-Q., Nguyen, T.-D., Pham, P.-H., Pham, X.-Q., Alsaffar, A.A., Huh, E.-N.: An efficient platform for mobile application development on cloud environments. In: International Conference on Computer Applications and Information Processing Technology (2015)