Contemporary Engineering Sciences, Vol. 9, 2016, no. 12, 571 - 577
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ces.2016.6443

Code Clone Detection Technique of Android Layout Codes Using View Attribute Filtering

JaeYeon Lee

College of Informatics, Korea University, Korea

Jongwook Jeong

College of Informatics, Korea University, Korea

Hoh In

College of Informatics, Korea University, Korea

Abstract

Each of Android application's screens is composed by an XML file called "Android layout code files". Developers can draw each screen by declaring user interface components in the Android layout code file. However, Android developers tend to use the same user interface components multiple times when declaring identical components in layout code, which leads to code clone issues that reduce the maintainability of code. There are many studies and techniques that detect code clones, but there is a lack of code clone research related to Android layout code. In this paper, we propose a new technique for detecting code clones in Android layout code. Our technique uses a tree-based model for data abstraction and compares each tree node by considering visual and non-visual View attributes of the Android layout codes. We selected 12 Android open source projects and developed a code clone detection tool to check validity of our approach. Our tool uses the proposed technique and two other existing techniques for comparison. The experimental results showed that our proposed technique exhibited higher precision and recall on the projects compared to the other two techniques. The results obtained from applying our technique to open source projects show that our technique is useful for detecting code clones.

Keywords: Code clone, Android, Refactoring, Android Layout XML

1 Introduction

Android is one of the most widely used mobile operating systems since 2009, and the number of applications developed for the Android platform is rapidly increasing. The rapid growth of Android development has given rise to a lot of concerns related to software maintenance. One of the important maintainability issues, presented in an Android developer document [9], is code clones in Android layout. Due to the fact that applications in Android have identical interfaces in different screens, developers tend to use the same android layout code fragment in different files or lines, leading to code cloning issues.

Code cloning is defined as using identical or similar code fragments multiple times in several lines or files [1, 6]. Moreover, code cloning is considered as a "bad smell" in code [2], which lowers maintainability of software by forcing developers to modify the same code fragment multiple times [3-5]. Likewise, code cloning issues in Android layout also lower maintainability of applications. In order to solve code cloning issues, developers must search through each codes when finding code clones, but searching through code is very time consuming [12]. There are many previous studies that detect cloned code automatically [6]. However, these studies do not consider the characteristics of Android layout.

In this paper, we propose a new technique for detecting code clones in Android layout code by considering the characteristics of Android layout. We have used a tree-based model for abstracting Android layout XML code with Views and ViewGroups as nodes. For node comparisons, our approach checks maximum depth from leaf nodes and child nodes for comparing hierarchy of sub-trees, and types and View attributes for checking if two sub-trees are rendered identically. We classified View attributes as visual and non-visual, because not all of the View attributes influence the rendered interface.

In order to validate our approach, we have compared our technique with two different techniques. One of the techniques uses a tree-based model but does not consider the View attributes, and the other one compares each of the View attributes in raw text. Our results show that our technique has achieved higher precision than the technique that does not consider View attributes, and a higher recall than the technique comparing View attributes in raw text.

2 Background

2.1 Android layout

Android layout uses XML (Extensible Markup Language) form, for composing rendered form on the user interface screen. Android layout XML is composed of user interface components called "Views" and invisible containers called "ViewGroups" that contain Views and ViewGroups to form a hierarchy [7-8]. Each View and ViewGroup has unique Android XML attributes that declare visual

properties, behavior, and identifiers. If View attribute values that affects rendered screen are the same, either View or ViewGroup will be rendered identically. Therefore, developers are able to use same View or ViewGroup with same View attributes when drawing identical components in different parts, which brings code cloning issues. In order to solve these issues, Android supports "</include>" tag [9], which integrates same View or ViewGroup code into single file.

2.2 Tree-based comparison

In XML, elements form a hierarchy that can be abstracted into a tree model. Likewise, Android layout XML can also be abstracted into a tree model with Views and ViewGroups as tree nodes. A tree-based technique in code clone detection finds cloned fragments by abstracting code into a tree model and finding similar subtrees [10]. For the above reason, we selected a tree-based technique for data abstraction that corresponds to the hierarchy structure of XML. To find cloned fragments, we list the available sub-trees for each file and search for similar or identical sub-trees, one pair of files at a time. Deciding if two sub-trees are similar is done by checking four conditions. First, we check if the depth of two sub-trees from the deepest leaf node is the same. Second, we check the type of root nodes of the sub-trees. Third, we check if each child node of the root node has been checked as similar. Finally, we check the root nodes View attributes by the View attribute filtering technique that we have proposed. Two sub-trees are marked as similar if all four conditions are fulfilled.

3 View attribute filtering: Proposed approach

Android layout XML has a variety of attributes for each View or ViewGroup types. These View attributes define its behavior or visual form or identifier for external (Java code) references. In order to check two different View or ViewGroup is identically rendered, we have to compare visual View attribute values that affect rendered interface. This is because identical interface normally shares same values for visual View attributes. If visual View attributes have different values, the rendered screen will differ with one another. On the other hand, non-visual View attributes might not be the same even if two View or ViewGroup are identical. In order to detect identically rendered interface from layout code, we have to compare only visual View attributes.

We classified existing Android layout attributes from Android developer documents [7-8] into visual and non-visual. Classification of attributes was done by inspecting each attribute's role in the interface defined in developer documents [7-8]. Attributes that declare identifiers, or behaviors for user interactions, or descriptions, were classified as non-visual. Attributes that declare positions, lengths, colors, source, size, and alignment, were classified as visual. As a result, we created a classification table with non-visual and visual attribute names, as shown in Table 1. We used this table to filter unnecessary attributes, and compare only necessary attributes that affects rendered interface.

Category	View Attribute names
Non-visual View attributes	Id, contentDescription, Tag, nextFocus, focusable, clickable, longClickable, onClick, labelFor, Hint, important, For-Accessibility, TransitionName, descendantFocusability, touchScreenFocusability, transitionGroup
Visual View attributes	padding, minHeight, minWidth, layout_weight, layout_centerVertical, layout_margin, width, height, layout_gravity, background, ellipsize, textSize, textAppearance, textAlignment, Orientation, Src, color, alpha

Table 1. Visual and Non-visual View attribute list

4 Experiment

To prove the effectiveness of our approach, we selected 12 open source Android projects from GitHub [11]. We considered two conditions for selecting the projects. First, the project must contain cloned code fragments in Android layout. This is because detecting cloned code fragment is the main concern of our work. Second, the project must have more than 100 "Stars," which is the number of users who are interested at that project. This is because number of stars shows the size of community who are interested in that project, and project with large community tends to have better quality with independent, objective reviewing from people outside the project team [13]. Number of Stars, commits, and last updated date for each selected projects are listed in Table 2.

We developed a tool and counted code clone fragments for each of the three different techniques for comparison. The first one was our proposed approach that used a tree-based technique and View attribute filtering, as described in the previous section. The second technique is the text-based technique that compares code fragments by raw text. The third technique is another tree-based technique that compares only the hierarchy of the tree and does not consider View attribute values. Our tool gets Android layout code as input, which is parsed and abstracted it into tree models. The abstracted tree models are compared with one another by comparing each of the sub-trees. Sub-tree comparison is done using the three techniques mentioned above. After that, detected code fragments for each technique are counted and stored separately.

Project name	Stars	Commits	Last-updated
Ace Music player	213	3	2015-01-06
BetterBatteryStats	233	1884	2016-03-23
Duckduckgo	230	1198	2016-02-28
FBReaderJ	1028	9005	2016-01-31
GEM Music player	210	217	2016-03-20
Jams Music player	1913	79	2014-10-21
K9mail	2432	6147	2016-03-20
MPDroid	425	2594	2014-12-10
OpenStreetMap	657	1859	2013-03-13
OSMAndroid	730	29312	2016-03-26
OSMonitor	196	337	2015-11-03
Wordpress	1124	14805	2016-03-25

Table 2. Selected Android open source project list

Finally, our tool presents the number of detected code fragments for each project, with rewritten code snippets of detected cloned fragment that can be integrated into new "<include/>" tags in the Android layout XML file. The number of detected code fragments obtained from our tool for each project is shown in Table 3. In order to compare effectiveness of our proposed technique with previous code clone detection techniques, we calculated recall which shows completeness of detection, and precision which shows purity of detection. For calculating

Project name	Tree-based (View Attribute filtered)	Text-based	Tree-based (No filtering)	Oracle
Ace Music player	22	16	41	23
BetterBattery Stats	3	2	8	3
Duckduckgo	2	0	4	2
FBReaderJ	4	1	11	4
GEM Music player	3	3	9	3
Jams Music player	32	18	50	33
K9mail	2	1	12	3
MPDroid	5	4	12	5
OpenStreeMap	3	3	11	3
OSMAndroid	4	4	49	5
OSMonitor	1	0	3	1
Wordpress	10	6	38	8

Table 3. Detected code clone fragments

recall and precision from data obtained from our tool, we counted real cloned code fragments which called as oracle, by inspecting every layout files and interface rendered from layout files manually. The number of oracles for each project is given in Table 3.

From this experiment, we counted the number of oracles and detected cloned fragments for each project and technique for calculating recall and precision. As a result, we have obtained average recall and precision of all selected projects for each technique. Our proposed approach detected most of the oracle code clone fragments with average 95% recall, and avoided finding false positives for most of the projects with average 98% precision. On the other hand, the text-based approach detected code clone fragments without any false positives for every project, but suffered from a number of false-negatives. This is because the text-based approach strictly compares every View attribute including unnecessary attributes like identifiers and descriptions, which does not affect the rendered interface. The tree-based approach that did not consider View attributes found every oracle without missing a single one. This is because cloned codes must have same tree structure. However, it suffered from false-positives more than the number of oracles. The reason for false-positives is mainly from unconsidered attribute values that affect the rendered interface. From this experiment, we observed

that our technique got relatively higher precision and recall overall for the selected projects compared to two other techniques, indicating that our approach holds promise to accurately detect cloned fragments in Android layout.

5 Conclusion

In this paper, we presented code clone detection technique for Android layout codes. It is a tree-based technique with View attribute filtering that filters visual attributes that affect the rendered interface, and non-visual attributes that have no effect on the rendered interface. From code clone detection tool we developed, our approach detected code clones in selected open source projects with relatively higher recall and precision compared to a text-based approach and a tree-based approach that did not consider View attributes. From this result, we could conclude that considering Android layout View attributes improves accuracy by lowering both false-positives and false-negatives when finding code clones in Android layout. With our proposed approach, we can detect code clones in Android layout code and help refactoring it into <include/> tags. However, our approach still has limitations, such as consideration of change in outer XML files, identically rendered attribute values and consideration of custom View attributes. Our future work will solve these issues for better code clone detection.

Acknowledgements. This research was supported by the Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF), funded by the Ministry of Science, ICT & Future Planning (2012M3C4A7033345), and the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2016-R0992-15-1011) supervised by the IITP (Institute for Information & Communications Technology Promotion)

References

[1] M. Kim, V. Sazawal, D. Notkin and G. Murphy, An empirical study of code clone genealogies, *ACM SIGSOFT Software Engineering Notes*, **30** (2005), 187-196. http://dx.doi.org/10.1145/1095430.1081737

[2] M. Fowler, *Refactoring: Improving the Design of Existing Code*, 11th European Conf. Jyväskylä, Finland, 1997.

[3] G.M.K. Selim, L. Barbour, W. Shang, B. Adams, A.E. Hassan and Y. Zou, Studying the impact of clones on software defects, *17th IEEE Working Conf. Reverse Engineering (WCRE)*, (2010), 13-21.
http://dx.doi.org/10.1109/wcre.2010.11

[4] M. Mondal, M.S. Rahman, R.K. Saha, C.K. Roy, J. Krinke and K.A. Schneider, An empirical study of the impacts of clones in software maintenance, *19th Int'l Conf. Program Comprehension (ICPC)*, (2011), 242-245. http://dx.doi.org/10.1109/icpc.2011.14

[5] C. Boldyreff and K. Richard, Reverse engineering to achieve maintainable WWW sites, *8th IEEE Working Conf. on Reverse Engineering (WCRE)*, (2001), 249-257. http://dx.doi.org/10.1109/wcre.2001.957829

[6] R. Koschke, Survey of research on software clones, *Dagstuhl Seminar Schloss*, Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[7] Google developers, Android View, http://developer.android.com/intl/ko/reference/android/view/View.html

[8] Google developers, Android ViewGroup, http://developer.android.com/intl/ko/reference/android/view/ViewGroup.html

[9] Reusing layouts, http://developer.android.com/intl/ko/training/improving-layouts/reusing-layouts.html

[10] K.R. Chanchal, J.R. Cordy and R. Koschke, Comparison and evaluation of code clone detection techniques and tools: A qualitative approach, *Science of Computer Programming*, **74** (2009), 470-495. http://dx.doi.org/10.1016/j.scico.2009.02.007

[11] GitHub, https://github.com/

[12] D. Ekwa and M.P. Robillard, Tracking code clones in evolving software, *29th IEEE Int'l Conf. Software Engineering (ICSE)*, (2007), 158-167. http://dx.doi.org/10.1109/icse.2007.90

[13] M. Aberdour, Achieving quality in open-source software, *IEEE Software*, **24** (2007), 58-64. http://dx.doi.org/10.1109/ms.2007.2

Received: April 7, 2016; Published: May 26, 2016