International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-11, September 2019

# Detecting Features Inconsitency in Cross-Platform Applications

**C.Shanthi, K.Sharmila, J.Jebathangam, R.Devi**

***Abstract:*** ***The development of mobile computing has offered rise to a differing set of computing platforms. Customers utilize these distinctive stages for both personal and business exercises, for example, banking, shopping and so on. The developers of mobile computing faces many challenges which includes numerous versions of operating systems and thousands of devices which various in screen sizes.While developing mobile applications, the application engineers have to determine APIs of a home stage (e.g., Windows Phone), and consequently create forms of the application for different target stages (e.g., iOS and Android).Because of this cross platform application development is a striking issue for programming engineers who need to pitch to clients regardless of which platform they run (Windows, Mac or Linux).***

***Keywords: Mobile Application, Cross Platform, Inconsistency, Android, iOS***

## I.INTRODUCTION

Maximum number of mobile devices is operated on Android, iOS or Windows 10; these are generally known as operating systems or platforms. The three types of mobile application namely native application, web application and hybrid application. Native applications are created to aim one particular platform like android, iOS or Windows. Mobile web applications are developed for web applications to deliver pages on web browsers which are getting operated in mobile devices. Hybrid applications have been developed to aim multiple Cross platforms mobile applicationsThe major dispute in developing Hybrid/cross platform mobile applications for interoperability through multiple platforms is to retain the developed application consistent across different platforms. The next frequent issue in a cross-platform mobile application is to identify the missing features among the applications which are developed for different platforms. The application developers should check the developed mobile application on each and every platform independently and physically executing screen-by-screen assessment and detecting many inconsistencies in cross-platforms. But it is very difficult, time consuming and moreover it might have errors.

**Manuscript published on 30 September 2019.**
*Correspondence Author(s)
**Dr.C.Shanthi,** Associate Professor, School of Computing Sciences, VISTAS.
**Dr.K.Sharmila,** Associate Professor, School of Computing Sciences, VISTAS.
**J.Jebathangam,** Assistant Professor, School of Computing Sciences, VISTAS.
**R.Devi,** Assistant Professor, School of Computing Sciences, VISTAS.



## II. LITRETURE SURVEY

The problem of solving multiple user interfaces started a while

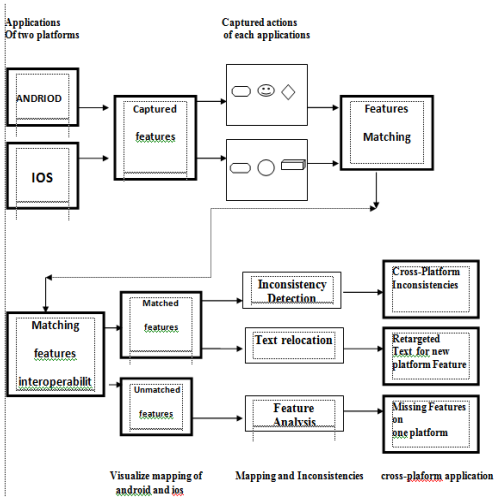


**Fig. 1. Overview of the Technique**

ago, however, to date actual insufficient effort is available for mobile devices explicitly in the literature. Mattia Fazzini and Alessandro Orso (2017) suggested DIFFDROID, an innovative method that assistances developers to discover inconsistencies in native mobile applications. DIFFDROID syndicates contribution generation and discrepancy testing to assess the performance of an application on different platforms and identifies possible inconsistencies Choudhary et al. (2014) developed a technique to examine the client-server communication and network traces of different versions of a web application to match features across platforms. A.Mesbah and M. R. Prasad (2011) found an automated solution for the problem of cross-browser compatibility testing of modern web applications as a 'functional consistency' check of web application behavior across different web browsers. Their method involves mechanically examining the assumed web application under dissimilar browser surroundings and taking the behavior as a finite-state machine and formally comparing the generated models for equivalence on a pair wise-basis and exposing any observed discrepancies. L. Wei, Y. Liu, and S.C. Cheung (2016) have reported a technique named FicFinder to detect compatibility issues in Android applications. .

## III. DETECTING FEATURES INCONSITENCY

This technique recognizes and coordinates the highlights of a cross-platform mobile application by examining the customer server correspondence that happens when the mobile application is utilized on the distinctive platforms. At a higher level, the technique operates in four major steps: (1) to verify the traces of the system contact among the client and server of various platform-specific versions of a cross platform mobile applications, (2) to find each traces and unique states in the model as a sequence , (3) to recognize a division of these traces as feature instantiations, (4) to match the feature sets recognized for each and every platform-specific version of the cross platform mobile application which is used to recognize 78 matched and lost features ,and different functionality of applications across versions, (5) to create a image of the different models, finding the detected data inconsistencies.The Fig 1 shows the overview of the featCHECK. It captures the activities of the Android and iOS application running on two unique stages and various Shapes have been captured for the two platforms discretely. It can be an obviously expressed model or can be the runtime trace of the mobile application. At this point, reasonable captured data is then compared over numerous platforms and can prompt both matched and unmatched behavior over the platforms. Which matches the number of edges and unique states in the shapes of the mapping features.

### *A. featCHECK*

The featCHECKis used for exactly identifying matched and un-matched features finding in cross platform mobile application. Figure 1 gives a high-level vision of the technique used, featCHECK. The first and foremost step of featCHECK is to group a set of system level traces for both mobile application versions. The resulting feature mapping has been developed by the premise of trace-sets. The fundamental process of mapping features generally independent and based on trace collection. There are three principal stages, reflecting the three difficulties discussed which has been given below. The primary step, the system traces are found to recognize request which are examples of the same action. In this stage, all requests are preoccupied and mapped onto a little alphabet of actions. In the next step, the unique traces from each and every platform are bundled and canonicalzed into a center arrangement of traces. In the third step, the canonicalzed traces from the cross platform mobile applications are looked at against each other to locate mapping between features.

### *B. Algorithm: 1*

I/p: T: Locate traces
O/p: CA: Collection of actions
Begin
K ←TraceSimplify(T)
Level 1 Collection
Level 1Collection ←SimpleCollection(T; url-path equals)
Level 2 Collection
Level 2 Collection ← {}
TD← {TanimotoDistance(k1; k2) | k1; k2∈ K}
underCollection← split(Level 1Collection; size = = 1)
overCollection← split(Level 1Collection; size > 1)
L2Collection: add(AggloCollection(underCollection;TD; (<; t1)))
For eachc∈overCluster do
L2Collection: add(AggloCollection(c; TD; (>; t2))
return L2Collection

### *C. Algorithm: 2*

Trace Simplify
I/p : T: Set of cross platform traces = {T1, T2, .., Tn}
O/p: W: Set of words tuple sequences = {w1, w2, ..,wm}
Begin
W ← ( )
For each T ∈ T do
For each<request, reaction>∈ T do
while isRedirect(reaction.code) do
reaction ← followRedirect(reaction)
if isCodeOrData(reaction.type) then
w ← getWordsws(request.path, request.qs)
W.add(w)
Return w.

In iOS and Android have diverse User Interface (UI) components; mapping process is expected to discover identical gadgets. The Graphical User Interface (GUI) components that exist for both local Android and iPhone platforms are distinguished based on the dissimilarity and similarities on both the platforms. These UI identical mappings are made freely accessible. The relation returns 1 if both components are viewed as mapped and 0 if not.The traces and feature mapping stages which yields individual models with an arrangement of registered secondary mapping properties for their unique states and limits (FMIG and FMAG). This algorithm works based on the following assumptions (1) the application model begins with first edge that prompts an initial state and (2) competently, the two models begin with a similar starting states and edge Pairs of iPhone and Android. In order to discover the edge-pairs, all the outgoing iPhone edges are acquired. The feature mapping illustrates the matching problem which has the extreme weighted bipartite coordinating (MWBM) issue. It gives a bipartite chart G = (V: E) where V and E represents Vertices and Edges respectively. The information for states and edges are utilized as a part of this phase to outline feature of two models. The Fig 2 demonstrates unmatched features from both the platforms

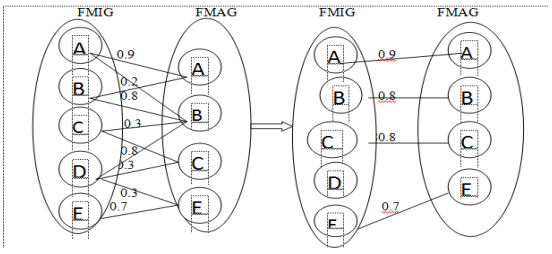


**Fig..2. Bipartite graphs of features mapping**

This issue is to find out similar of maximum weight wherever the weight of matching M is given by w(M) =Pe2M w(e).

The definition of the MWBM is utilized in Hungarian Algorithm. The edge E operates amongst iOS and ANDROID and signifies the opportunity of similar and comparing features. The weight on an edge denotes the profits of matching both the features, the possibility is that they are surely right matches. On the other side, features A-E from the iOS Platform (FMIG) are connected with features 1-4 from the Android platform (FMAG) through edges, labeled for each pair.On the right side of the figure 3 is the answer for the MWBM problem where only the edges adding to the extreme profit are engaged. This process of matching is the last and final consequence of the calculation and gives a summary of matched features, which is [(A, B, C, D, E); (A, B, C, E)] for the representation.. Table I shows the collective numeral of 'Unique States', 'Edges', User Interface essentials aimed at all the circumstances operating on every Android and iPhone application, developed by featCHECK. The last splinter of the table 1exhibits the quantity of states which are unique and the result demonstrates that featCHECK can differentiate the new constraints of a specified iPhone and Android application-pair and creates a couples of situations both in iPhone applications and Android applications wherever the quantity of physical noteworthy exclusive states which do not accurately synchronize the quantity of states which are exceptional and poised through the vibrant analyzer. This is primarily approach now receipts into account the type of the class (both chat in Android and missing chat in iOS) in major a unique state and thus divided for different views.

The Table above makes to evaluate a step by step process, in finding the traces of the applications which makes the task easy to find the data inconsistencies in given 25 open-sources cross platform applications.

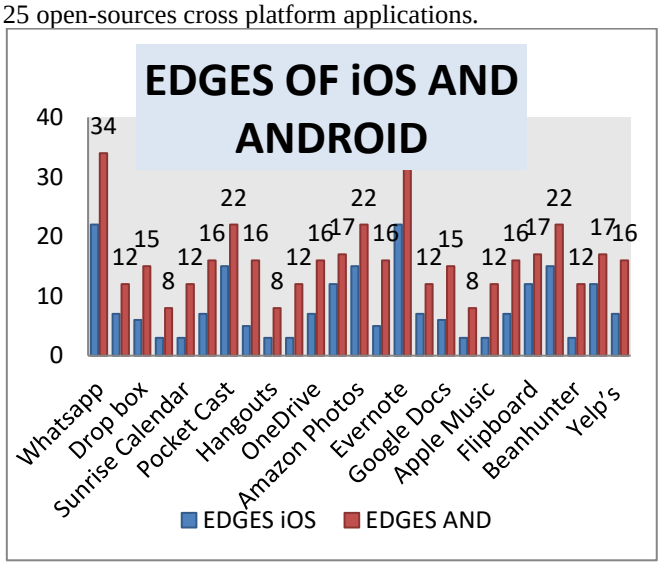


**Fig. 3. Graph Representations of Edges**

Fig 3 shows the graphical representation of the edges extracted from the applications of iOS and Android.

**Table- I: Traces and Actions of iOS and Android**

| S.No | Name of cross platform apps | Type | EDGES | | UNIQUE STATES | |
|---|---|---|---|---|---|---|
| | | | iOS | AND | iOS | AND |
| 1 | Whatsapp | Chat | 22 | 34 | 14 | 17 |
| 2 | Facebook messenger | Messaging | 7 | 12 | 4 | 11 |
| 3 | Drop box | Upload files | 6 | 15 | 5 | 13 |
| 4 | Wunderlist | List and Notes | 3 | 8 | 5 | 11 |
| 5 | Sunrise Calendar | Calendar | 3 | 12 | 7 | 8 |
| 7 | Microsoft Word | Documents | 7 | 16 | 5 | 14 |
| 8 | Spotify | Music | 12 | 17 | 9 | 10 |
| 9 | Pocket Cast | Service apps | 15 | 22 | 8 | 9 |
| 10 | IF | Update Photos | 5 | 16 | 5 | 12 |
| 11 | Hangouts | Access files | 3 | 8 | 5 | 11 |
| 12 | Google Drive | changes to a file | 3 | 12 | 7 | 8 |
| 13 | OneDrive | Word or Excel | 7 | 16 | 5 | 14 |
| 14 | Google Photos | photo apps | 12 | 17 | 9 | 10 |
| 15 | Amazon Photos | Compression | 15 | 22 | 8 | 9 |
| 16 | Google Keep | reminders | 5 | 16 | 5 | 12 |
| 17 | Evernote | Upload documents | 22 | 34 | 14 | 17 |
| 18 | Google Calendar | Upcoming agenda | 7 | 12 | 4 | 11 |
| 19 | Google Docs | Microsoft's DOCX | 6 | 15 | 5 | 13 |
| 20 | Google Play Music | Online radio | 3 | 8 | 5 | 11 |
| 21 | Apple Music | Music catalog | 3 | 12 | 7 | 8 |
| 22 | News Republic | Latest headlines | 7 | 16 | 5 | 14 |
| 23 | Flipboard | News Republic | 12 | 17 | 9 | 10 |
| 24 | Four Square | Local reviews | 15 | 22 | 8 | 9 |
| 25 | Bean hunter | Find cafes | 3 | 12 | 7 | 8 |

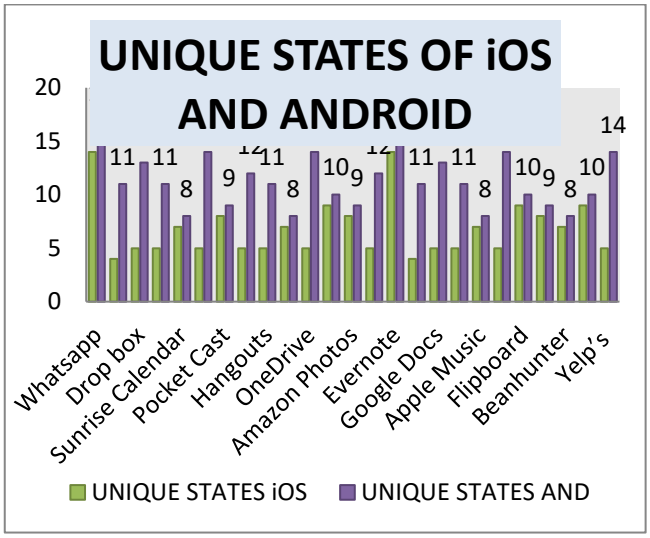


**Fig. 4 Graph Representations of Unique States**

The Fig 4 shows the Unique States evaluated from the given 25 open-source cross platform application. The graph clearly shows the difference in the states of both iOS and Android.

**Table- II: Bug Severity Descriptions**

| Severity | Description | level |
|---|---|---|
| Critical | no loss in features, no work-around | 5 |
| Major | no loss in features, with possible work-around | 4 |
| Normal | Makes a task difficult to use data | 3 |
| Minor | Functionality not affecting, behavior is not expected | 2 |
| Trivial | Functionality not affecting, surface problem | 1 |

Table II gives the descriptions about the bug severity. The majority of the false encouraging points in the detailed data inconsistencies are expected to the UI configuration state which has been implemented differently on both the platforms.

## IV. CONCLUSION

The need for cross platform mobile application development has become an undeniably regular industry practice. In order to make the developed application consistent across multiple platforms is very important. So there is need for mobile developers and analyzers to test the application's consistency and guarantee that the behavior of the application is same across multiple platforms. In this work, the implemented featCHECK technique consequently identifies and visualizes inconsistencies among iOS and Android versions which have same mobile applications

The performance valuation on 25 application couples demonstrates that the GUI based perfect could offer a good solution and also it maps the application-pairs.

## REFERENCES


1. Signal, O. (2013). Android fragmentation visualized. Retrieved from opensignal. com: http://opensignal. com/reports/fragmentation-2013.
2. Martin, W., Sarro, F., Jia, Y., Zhang, Y., & Harman, M. (2017). A survey of app store analysis for software engineering. IEEE transactions on software engineering, 43(9), 817-847.
3. Williamson, L. (2012). A mobile application development primer. A guide for enterprise teams working on mobile application projects. IBM Whitepaper.
4. Hu, H., Wang, S., Bezemer, C. P., & Hassan, A. E. (2018). Studying the consistency of star ratings and reviews of popular free hybrid Android and iOS apps. Empirical Software Engineering, 1-26. Calabash. http://calaba.sh/.
5. Android Market Stats. http://www.appbrain.com/stats/.
6. Tolk, A., & Muguira, J. A. (2003, September). The levels of conceptual interoperability model. In Proceedings of the 2003 fall simulation interoperability workshop (Vol. 7, pp. 1-11). Citeseer.
7. Guédria, W., Naudet, Y., & Chen, D. (2008, November). Interoperability maturity models–survey and comparison–. In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems" (pp. 273-282). Springer, Berlin, Heidelberg.
8. Mesbah, A., & Prasad, M. R. (2011, May). Automated cross-browser compatibility testing. In Proceedings of the 33rd International Conference on Software Engineering (pp. 561-570). ACM.
9. Choudhary, S. R., Prasad, M. R., & Orso, A. (2013, May). X-PERT: accurate identification of cross-browser issues in web applications. In Software Engineering (ICSE), 2013 35th International Conference on (pp. 702-711). IEEE.
10. Roy Choudhary, S., Prasad, M. R., & Orso, A. (2014, July). Cross-platform feature matching for web applications. In Proceedings of the 2014 International Symposium on Software Testing and Analysis (pp. 82-92). ACM.
11. Griebe, T., Hesenius, M., & Gruhn, V. (2015, September). Towards automated UI-tests for sensor-based mobile applications. In International Conference on Intelligent Software Methodologies, Tools, and Techniques (pp. 3-17). Springer, Cham.
12. Joorabchi, M. E., Ali, M., & Mesbah, A. (2015, November). Detecting inconsistencies in multi-platform mobile apps. In Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on (pp. 450-460). IEEE.
13. Wei, L., Liu, Y., & Cheung, S. C. (2016, August). Taming Android fragmentation: Characterizing and detecting compatibility issues for Android apps. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (pp. 226-237). ACM.
14. Kang, Y., Zhou, Y., Gao, M., Sun, Y., & Lyu, M. R. (2016, October). Experience report: Detecting poor-responsive ui in android applications. In Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on (pp. 490-501). IEEE.
15. Menegassi, A. A., & Endo, A. T. (2016, October). An evaluation of automated tests for hybrid mobile applications. In Computing Conference (CLEI), 2016 XLII Latin American (pp. 1-11). IEEE.
16. Fazzini, M., & Orso, A. (2017, October). Automated cross-platform inconsistency detection for mobile apps. In Automated Software Engineering (ASE), 2017 32nd IEEE/ACM International Conference on (pp. 308-318). IEEE.