Proactive API Compatibility Testing for Mobile Applications

Sai Prakash Pathuru
University of Colorado Colorado Springs
Colorado Springs, USA
spathuru@uccs.edu

Abstract

From the past decade, compatibility testing has been a major and urgent issue[18]. Compatibility testing itself is a huge tree and API compatibility testing is a branch in that huge tree. This paper focuses on detecting API compatibility issues between two mobile applications. There are few methods to over come these compatibility issues and most of the methods are performed after development phase is done. So, small developers may find that the APIs they choose may not be compatible in the testing phase or in the deployment phase which leads to delay in product deployment and the budget exceeds. To over come this situation, we are coming with the new methodology by making this API compatibility testing Proactive, so that this tool detects API compatibility issues at the time of development phase and help the developers to over come the issue in the development phase itself.

Keywords: Proactive Compatibility Testing, Mobile Applications, API(Application Programming Interface).

ACM Reference Format:
Sai Prakash Pathuru. 2018. Proactive API Compatibility Testing for Mobile Applications. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

From the past decade, mobile device usage has been increasing rapidly and many mobile applications are coming out to the market. So, clients with ideas approach developers to build their mobile applications. Some may keep up with the clients and build all the features within the agreed time and budget, but some small developers may not complete the project within time or budget and at times like that and other various reasons, developers tend to use API of another application to acquire services.

These APIs the developers choose may or may not be accurate until unless the API is implemented and tested in their application. So, rather than finding out the API is not compatible with their application in the testing phase, we are making this API compatibility testing proactive so that the developers can find out if the API is compatible with their mobile application in the development phase itself and find if there are any other compatible issues in the implementation/development phase.

API Compatibility Testing focuses on two sides. They are:

1. API compatibility testing between two mobile applications, where these two mobile applications communicate and acquire services through API.
2. API compatibility testing between a mobile application and a mobile device to acquire services of mobile device hardware.

This paper focuses on API compatibility testing between two mobile applications, where these two mobile applications communicate and acquire services through API.

So, as the developers are acquiring some features from a third-party applications through API to keep up with the market. Overtime, considering changes in applications and API versions, using API between two different mobile applications can lead to compatibility issues.
Consequences of API compatibility issues:

1. Low performance — If the API is not properly build, it leads to compatibility issues effecting the user experience.
2. Data inconsistency — Compatibility issues can lead to data discrepancies between two applications, leading to inconsistencies in transaction record, user profiles, etc.
3. Security vulnerabilities — If the API interaction is poor, it can lead to security issues and can have high chance of exposing sensitive data.
4. Customers — Issues like low performance, data inconsistencies, and security vulnerabilities will lead to losing customers and eventually leads to bad reputation.
5. Market — As there are thousands of applications for a single genre, the will be a lot of competition between

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

the same genre of applications. So, once if the application starts gaining bad reputation, it will be very difficult for that application to bounce back in the market.

Until now, to overcome the API compatibility issues, few authors came up with different methods and tools like *How Android developers handle Evolution-induced API compatibility issues*[1], *CiD*[2], *ACID*[3], *Detecting Android API Compatibility Issues With API Differences*[4]. Most of the methods are used after completion of development/implement phase but this paper focuses on a method to make the API compatibility issues detection proactively, that is this method will run in the background at the time of developing an application so that when developing an application with API in it, this method detects if there are any API compatibility issues at the time of developing so that the developers can overcome the issues at the time of developing and deploy the product with minor issues.

This paper evaluates this method by considering an application version and an API version which has more API compatibility issues by the time of deployment from the history of that mobile application. We will develop that version of application from the start but with our API compatibility issue detection tool running in the background and see how many compatibility issue are there by the time of deployment and present a chart showing the results of developing an mobile applications without our tool and with our tool.

In summary the main contribution of this research are:

1. Can we make the API compatibility issue detection Proactive? How?
2. Can integrating an machine learning algorithm helps to improve API compatibility issue detection?
3. Overall, How can this tool help the developers?

2 Background and Related Work

What is API compatibility Testing?
API(Application Programming Interface) compatibility testing is a type of testing that ensures the compatibility and proper functioning of APIs, particularly when they interact with each other or with other components of a software system.

Why is proactive API compatibility testing important?
Intense competition in mobile application market — Though there are thousands of mobile applications in various market, most mobile users often use only about nine applications every day. Users often abandon mobile applications with serious compatibility issues, and seek alternatives with in seconds.
Frequent changes and upgrades of mobile applications and APIs — API compatibility testing often must be redone when there are new changes in mobile application or if there are any changes in API to check if the either of the new versions are compatible with each other or not.
Complex mobile user interfaces — Mobile applications have various user interfaces, such as: key, gesture, voice, and sensor. If the API interactions between our mobile application and the API is not properly build, these user interfaces may not function properly leading to unexpected compatibility issues.
Higher costs of compatibility testing — Due to hundred kinds of mobile devices, frequent changes in mobile applications and APIs, we have to spend lot of time and effort on API compatibility testing.

From the past decade, the compatibility issues between the applications has become major and urgent problem that needs to be addressed. An author, *Tarek Muhmud* published a research paper called *API Compatibility Issue Detection, Testing and Analysis for Android Apps*[5], from this paper we understood that mobile applications and APIs are frequently evolving, which means it is an never ending process, which means the API compatibility testing is also never ending process.

Many authors came up with different methods, simultaneously with the new problems but the methods they published are not proactive. These methods are applied on the code after completion of development/implement phase. Where first implementing the code and then testing it proves to be not an effective way to do it when compared to a method where first developing the test cases and then implementing the code in such a way that satisfies the test cases, this method is called Test Driven Development(TDD)[7][8].

The similar concept is being applied in this paper by making this API compatibility testing Proactive, where this tool runs in the background and actively monitors the code, dynamically generate the test cases, and the results are assigned to a predictive model to check if the current method written is viable or not. If not it should pop up a message saying so and so method can raise so and so API compatibility issues. So, to make this API compatibility testing Proactive, we integrated a machine learning algorithm in to our method, this algorithm is called Decision tree algorithm[9].

What is Decision tree algorithm?
The decision tree algorithm is a supervised machine learning algorithm used for both classification and regression tasks. It works by recursively partitioning the data set into subsets based on the values of different features. This algorithm consists of three nodes. They are:

1. Root node — In the root node, we will have a feature with the best splitting criterion.
2. Internal node — In this internal node, the data set is split into subsets based on the values of the feature and the process continues until the stopping criterion is met.
3. Leaf node — In this leaf node, a class label is assigned based on the majority class of the instances in the subset.

3 Methodology

Proactive API Compatibility Testing method is build on python programming language. In this section, we will explain how does this proactive API compatibility testing tool works, explain what to do and what happens in each step and how the outline of the code looks like for each step in this tool.

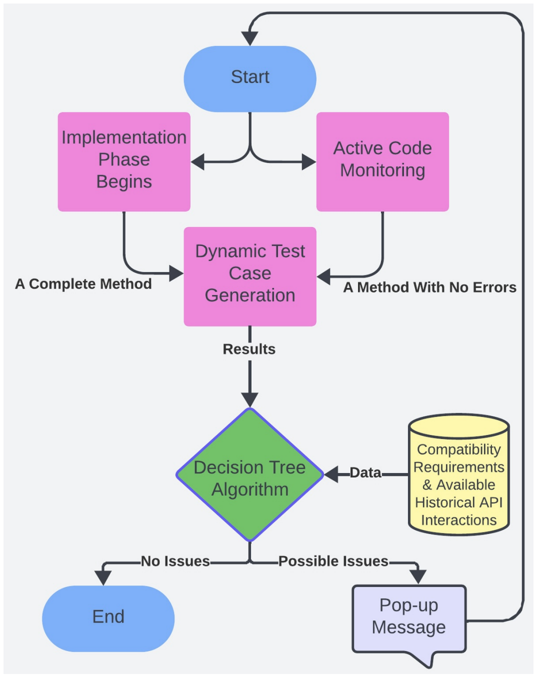


Figure 1. Flow Graph of Proactive API Compatibility Testing

3.1 How does this Proactive API compatibility testing tool works?

From figure 1, first, the developers must review all the API documentations that are to be used in their mobile application, to get better understanding of how the API works. Then, they have to set compatibility requirements according to the knowledge gained from the API documentations and according to the application's features and gather if there are any previous interactions between the API and the application for accurate outcomes. All this data must be stored before development/implementation phase and must be applied to predictive model(Decision tree algorithm).

Secondly, the developers must start the implementation/development phase once the API documentation is reviewed, and compatibility requirements are set. Once, a class or a method related to the application and API is developed, simultaneously, the code is actively monitored in the background and raise a pop-up message if there are any errors in that class/method. If there are no issues, then in the background, the test case are generated dynamically and the written code is tested and results are reported.

Once, the results are reported, these results are applied to the predictive model along with the compatibility requirements and if there are any stores interactions between API and the application in the history. These results from the test cases and the compatibility requirements are assigned to the root node of the decision tree algorithm. The, the root node divides the collected data into meaning full subsets which are called internal nodes in the decision tree algorithm. This division of subset continues until there is none to divide, then a no issues/possible issues type of result is produces in the leaf node of the decision tree algorithm.

If the result in the leaf node is "no issues", then the whole process is terminated believing that there are no API compatibility issues. If the result in the leaf node is "possible issues", then a pop-up message is displayed saying there are possible issues. If the result is "possible issues", then the developer tries to overcome those issues at the time of development/implement phase. Once, the developer attempts to change the code to overcome the issues, the whole process will run again as the code is changed. This process is iterative until the result in the leaf node changes from "possible issues" to "no issues".

3.2 What happens in each step?

3.2.1 Review API Documentation.

Developers of new application must thoroughly review all the API documentations that are used in their application before the implementation phase. Generally, an API documentation contains Introduction, Authentication, Endpoints, Request and Response formats, Error handling mechanisms. This helps the developers understand how they can use some third-party application services in their application through API. Once the developers are done going through the API documentation, the developers must feed all the knowledge they gained in the compatibility requirements document.

3.2.2 Define Compatibility Requirements.

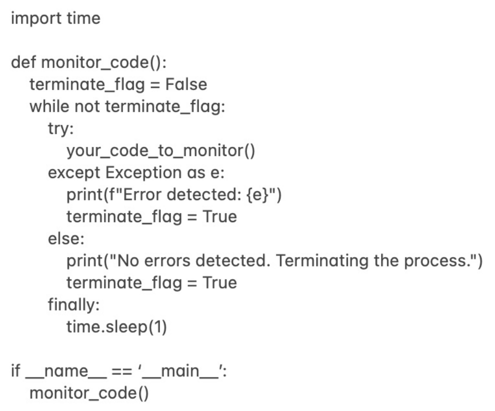
In this section, the developers must define the compatibility requirements such as minimum and maximum API versions that support their application, expected changes in the API version, and the knowledge they gained through API document before the implementation phase for their new application. All the information must be stored and must be fed to the predictive algorithm for accurate outcomes.

3.2.3 Collect Historical API Interactions.

In this section, if there are any, collect the data of previous interactions that took place between the application and the APIs used in that application, and any other issues in the application due to API. This data can also help to train the predictive model for accurate outcomes.

3.2.4 Active Code Monitoring.

In this section, when the development/implementation phase begins, the written code is being monitored and is being executed in real-time in the background and checking if there are any errors in the written code. If there are any errors, it displays the errors and once the developer overcomes those errors, again the code is monitored actively and this process is continued until there are no errors. If there are no errors then the process is terminated. The code for active code monitoring is shown in figure 2.



```
import time

def monitor_code():
   terminate_flag = False
   while not terminate_flag:
      try:
         your_code_to_monitor()
      except Exception as e:
         print(f"Error detected: {e}")
         terminate_flag = True
      else:
         print("No errors detected. Terminating the process.")
         terminate_flag = True
      finally:
         time.sleep(1)

if __name__ == '__main__':
   monitor_code()
```

Figure 2. Active code monitoring

Explanation of the code:

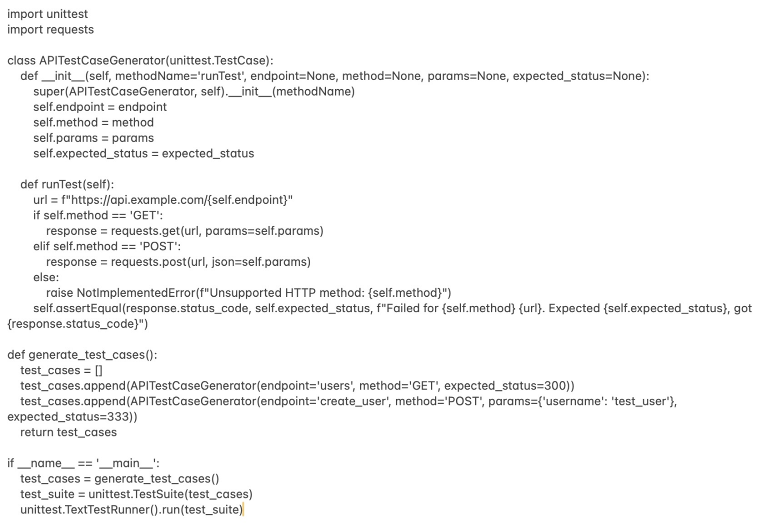
1. One way to approach this is by using a combination of try and except blocks to catch exceptions and a loop to continuously monitor the code. Additionally, we used a flag to indicate whether the process should terminate or not.
2. The monitorcode function contains a loop that continuously executes the yourcodetomonitor function.
3. If an exception occurs during the execution of the monitored code, it catches the exception, prints the error message, and sets the terminateflag to false. If no error occurs, it prints a message and sets the terminateflag to true. The loop then exits, terminating the process.

3.2.5 Dynamic Test Case Generation.

Once a method/class is written without any errors, simultaneously, along with the development/implementation phase, the test cases for that code is generated dynamically and is executed and tested in the background in real-time. Once all the test cases are passed, the result is assigned to a predictive model. The code for generating test cases dynamically is shown in figure 3.

Explanation of the code:

1. Generating python test cases dynamically for API compatibility involves dynamically creating tests that cover a range of scenarios to ensure the compatibility of an API.
2. We used unittest[12][13] module to dynamically generate test cases for API compatibility. We also used requests library for making HTTP requests.



```
import unittest
import requests

class APITestCaseGenerator(unittest.TestCase):
   def __init__(self, methodName='runTest', endpoint=None, method=None, params=None, expected_status=None):
      super(APITestCaseGenerator, self).__init__(methodName)
      self.endpoint = endpoint
      self.method = method
      self.params = params
      self.expected_status = expected_status

   def runTest(self):
      url = f"https://api.example.com/{self.endpoint}"
      if self.method == 'GET':
         response = requests.get(url, params=self.params)
      elif self.method == 'POST':
         response = requests.post(url, json=self.params)
      else:
         raise NotImplementedError(f"Unsupported HTTP method: {self.method}")
      self.assertEqual(response.status_code, self.expected_status, f"Failed for {self.method} {url}. Expected {self.expected_status}, got
{response.status_code}")

def generate_test_cases():
   test_cases = []
   test_cases.append(APITestCaseGenerator(endpoint='users', method='GET', expected_status=300))
   test_cases.append(APITestCaseGenerator(endpoint='create_user', method='POST', params={'username': 'test_user'},
expected_status=333))
   return test_cases

if __name__ == '__main__':
   test_cases = generate_test_cases()
   test_suite = unittest.TestSuite(test_cases)
   unittest.TextTestRunner().run(test_suite)
```

Figure 3. Dynamic test case generation

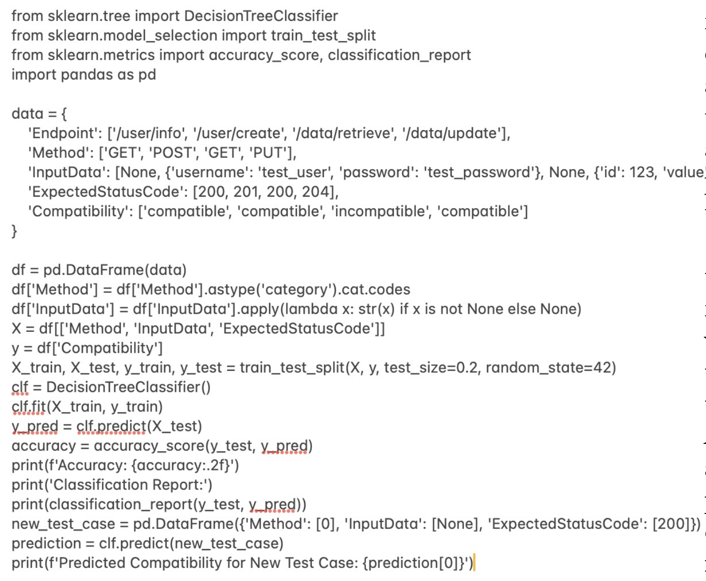
3. The APITestCaseGenerator class is a subclass of unittest.TestCase. It takes parameters like endpoints, method, params, and expectedstatus to dynamically create test cases for different API scenarios.
4. The generatetestcases function creates a list of test cases by instantiating APITestCaseGenerator with different parameters.
5. The generated test cases cover various API scenarios, such as different HTTP methods, endpoints, and parameters.
6. The test suite is created using unittest.TestSuite and then execute with unittest.TextTestRunner function.

3.2.6 Predictive model.

A predictive model is used in this tool to recognise patterns from the written code, test cases, by making sure it satisfies all the compatibility requirements, and check if the interaction between the application and the API is smooth or not. A machine learning algorithm called decision tree algorithm is used in this tool. Where in this algorithm we have three nodes, they are: root node, internal node, and leaf node. In root node, the raw data is fed, from figure 1, the raw data is the result of test cases, compatibility requirements, historical API interaction. Where this data is divided in meaningful subsets which are called internal nodes and this division of subset is a continues process until there none to divide. Once, it reaches the point where there is none to divide, then based on subsets, and requirements, it gives an yes/no type result in the leaf node. The code for predictive model is shown in figure 4.

Explanation of the code:

1. Scikit-learn is a popular python library for machine learning, including decision trees[10][11].



```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd

data = {
    'Endpoint': ['/user/info', '/user/create', '/data/retrieve', '/data/update'],
    'Method': ['GET', 'POST', 'GET', 'PUT'],
    'InputData': [None, {'username': 'test_user', 'password': 'test_password'}, None, {'id': 123, 'value': 42}],
    'ExpectedStatusCode': [200, 201, 200, 204],
    'Compatibility': ['compatible', 'compatible', 'incompatible', 'compatible']
}

df = pd.DataFrame(data)
df['Method'] = df['Method'].astype('category').cat.codes
df['InputData'] = df['InputData'].apply(lambda x: str(x) if x is not None else None)
X = df[['Method', 'InputData', 'ExpectedStatusCode']]
y = df['Compatibility']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print('Classification Report:')
print(classification_report(y_test, y_pred))
new_test_case = pd.DataFrame({'Method': [0], 'InputData': [None], 'ExpectedStatusCode': [200]})
prediction = clf.predict(new_test_case)
print(f'Predicted Compatibility for New Test Case: {prediction[0]}')
```

Figure 4. Predictive model

2. We create a sample data set representing API test cases with features such as methods, input data, expected status code, and compatibility.
3. The categorical data is converted into training and testing sets. The data set is split into training and testing sets.
4. A decision tree classifier is created and trained on the training set. The model is evaluated on the test set using accuracy and a classification report.
5. Finally, we used the trained model to predict the compatibility of a new test case.

4 Evaluation

To evaluate our tool, we have taken 2 mobile applications as a case study, in this 2 mobile applications, one is main applications and other one is the applications that provide services to the main application through API. Further, we will discuss a scenarios:

1. A navigation application using a weather API from other mobile application, to acquire weather conditions at the current location.

4.1 Case Study: Navigation Application

This navigation application is a road map, which helps the user travel from current location to desired location. As there are hundreds of applications for navigating, this mobile application decided to also display the weather conditions in the current location and the pinpointed destination location to attract customers.

So, to achieve this weather conditions requirement, developers performed traditional process, where the developers reviewed the API documentation and implemented the API in the development phase. Once the development phase is completed, then they performed regression testing on entire application to check if the application behaviour is satisfying the expected outcome or not. Initial they found that there are 8 compatibility issues due to API. Which lead them to put more time and efforts to overcome the issues, which lead to delay the deployment phase.

We had taken the API version and the mobile version that has 8 API compatibility issues. Now as our tool started running in the background, we again developed that mobile version with our procedure. We first, reviewed the API documentation and set compatibility requirements according to the API documentation and their application requirement. As we already have the interactions between API and the application, we acquired that and stored along with the compatibility requirements. Now, we started development phase of that version of the application, and we found few errors from the code and later some test cases are generated and results are produced. We then applied the results and the data we stored at the beginning on the predictive model. Once, the predictive model is processed, the predictive model was able to detect some patterns which leads to compatibility issues. So,we are able to overcome those issues in the development phase. Then, we performed regression testing to check the whole application behaviour and we found there are 3 API compatibility issues by the time of deployment phase.

4.2 Results

The results are represented in bar chart below in figure 5. From the figure 5, we can see the comparison of the Navigation application developed without our tool and with our tool. When the navigation application is developed without

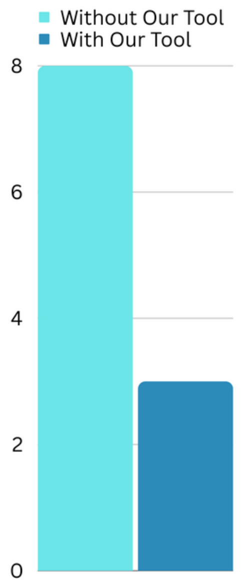


Figure 5. Comparison

our tool, by the time of deployment phase there are 8 API compatibility issues in the application. After developing the same version of the application with our tool running in the background, there were only 3 API compatibility issues by the time of deployment phase. From these results, we can understand that making the API compatibility testing proactive is much better than traditional way that is first completely developing the application with APIs and then testing it.

5 Limitations

There are few limitation for this tool as we used predictive model and other steps mentioned in the methodology section. They are:

1. As the predictive model requires data to train the model, if there are no historical API interactions between the API and the application, the results may not be accurate.
2. If any other steps in this tool are not executed properly like missing data, the results may not be accurate as that data is fed to the predictive model for accurate outcomes.
3. As mentioned earlier, this tool for now only focuses on detecting the compatibility issues between two applications when communicated through API.

6 Conclusion

From the results, we can say that making API compatibility testing proactive can give better outcomes when compared to first developing and then testing the API compatibility.

1. Can we make the API compatibility issue detection Proactive? How? Yes, by integrating the predictive model and the data to feed the model and the whole process running in the background during the development phase can make this API compatibility issue detection proactive.
2. Can integrating an machine learning algorithm helps to improve API compatibility issue detection? Yes, as we have enhanced machine learning algorithms, and to make this API compatibility testing proactive with accurate outcomes, integrating machine learning algorithm helps to improve API compatibility issue detection.
3. Overall, How can this tool help the developers? As this tool can detect API compatibility issues proactively, where developers can over come the API compatibility issues at the development phase, rather than finding them in the testing phase or deployment phase, we can say this tool can hugely help the developers.

7 Future Work

As for now, this tool is limited to API compatibility testing between two applications, in future we also want to implement the other side of API compatibility testing mentioned in the Introduction, which is API compatibility between mobile application and mobile devices to acquire the device hardware services.

In whole compatibility testing tree, API compatibility testing is a branch of that tree. As there has been increase in diverse mobile devices and UIs, in future we would like to improve our tool and make all the compatibility testing proactive.

References

[1] Hao Xia, Yuan Zhang, Yingtian Zhou, Xiaoting Chen, Yang Wang, Xiangyu Zhang, Shuaishuai Cui, Geng Hong, Xiaohan Zhang, Min Yang, and Zhemin Yang. 2020. How Android developers handle evolution-induced API compatibility issues: a large-scale study. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20). Association for Computing Machinery, New York, NY, USA, 886–898. https://doi.org/10.1145/3377811.3380357

[2] Li Li, Tegawendé F. Bissyandé, Haoyu Wang, and Jacques Klein. 2018. CiD: automating the detection of API-related compatibility issues in Android apps. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018). Association for Computing Machinery, New York, NY, USA, 153–163. https://doi.org/10.1145/3213846.3213857

[3] Tarek Mahmud, Meiru Che, and Guowei Yang. 2022. ACID: an API compatibility issue detector for Android apps. In Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings (ICSE '22). Association for Computing Machinery, New York, NY, USA, 1–5. https://doi.org/10.1145/3510454.3516854

[4] T. Mahmud, M. Che and G. Yang, "Detecting Android API Compatibility Issues With API Differences," in IEEE Transactions on Software Engineering, vol. 49, no. 7, pp. 3857-3871, July 2023, doi: 10.1109/TSE.2023.3274153.

[5] T. Mahmud, "API Compatibility Issue Detection, Testing and Analysis for Android Apps," 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 1061-1063, doi: 10.1109/ASE51524.2021.9678812.

[6] Tarek Mahmud, Meiru Che, and Guowei Yang. 2022. Android API Field Evolution and Its Induced Compatibility Issues. In Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '22). Association for Computing Machinery, New York, NY, USA, 34–44. https://doi.org/10.1145/3544902.3546242

[7] Chetan Desai, David Janzen, and Kyle Savage. 2008. A survey of evidence for test-driven development in academia. SIGCSE Bull. 40, 2 (June 2008), 97–101. https://doi.org/10.1145/1383602.138

[8] Boby George and Laurie Williams. 2003. An initial investigation of test driven development in industry. In Proceedings of the 2003 ACM symposium on Applied computing (SAC '03). Association for Computing Machinery, New York, NY, USA, 1135–1139. https://doi.org/10.1145/952532.952753

[9] Song, Yan-Yan, and Ying Lu. "Decision tree methods: applications for classification and prediction." Shanghai archives of psychiatry vol. 27,2 (2015): 130-5. doi:10.11919/j.issn.1002-0829.215044

[10] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, pp.2825-2830.

[11] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J. and Layton, R., 2013. API design for machine learning software: experiences from the scikit-learn project. arXiv preprint arXiv:1309.0238.

[12] Lukasczyk, S., Kroiß, F., Fraser, G. (2020). Automated Unit Test Generation for Python. In: Aleti, A., Panichella, A. (eds) Search-Based Software Engineering. SSBSE 2020. Lecture Notes in Computer Science(), vol 12420. Springer, Cham. https://doi.org/10.1007/978-3-030-59762-7-2

[13] Xiaoyu Sun, Xiao Chen, Yanjie Zhao, Pei Liu, John Grundy, and Li Li. 2023. Mining Android API Usage to Generate Unit Test Cases for Pinpointing Compatibility Issues. In Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22). Association for Computing Machinery, New York, NY, USA, Article 70, 1–13. https://doi.org/10.1145/3551349.3561151

[14] Pei Liu, Yanjie Zhao, Haipeng Cai, Mattia Fazzini, John Grundy, and Li Li. 2022. Automatically detecting API-induced compatibility issues in Android apps: a comparative analysis (replicability study). In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2022). Association for Computing Machinery, New York, NY, USA, 617–628. https://doi.org/10.1145/3533767.3534407

[15] S. Scalabrino, G. Bavota, M. Linares-Vásquez, M. Lanza and R. Oliveto, "Data-Driven Solutions to Detect API Compatibility Issues in Android: An Empirical Study," 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), Montreal, QC, Canada, 2019, pp. 288-298, doi: 10.1109/MSR.2019.00055.

[16] Jezek, K., Dietrich, J. and Brada, P., 2015. How java apis break–an empirical study. Information and Software Technology, 65, pp.129-146.

[17] Scalabrino, S., Bavota, G., Linares-Vásquez, M., Piantadosi, V., Lanza, M. and Oliveto, R., 2020. API compatibility issues in Android: Causes and effectiveness of data-driven detection techniques. Empirical Software Engineering, 25(6), pp.5006-5046.

[18] T. Zhang, J. Gao, J. Cheng and T. Uehara, "Compatibility Testing Service for Mobile Applications," 2015 IEEE Symposium on Service-Oriented System Engineering, San Francisco, CA, USA, 2015, pp. 179-186, doi: 10.1109/SOSE.2015.35.