ABSTRACT
The functionality of many mobile applications is dependent on various contextual, external factors. Depending on unforeseen scenarios, mobile apps can even malfunction or crash. In this paper, we have introduced MobiCoMonkey - automated tool that allows a developer to test app against custom or auto generated contextual scenarios and help detect possible bugs through the emulator. Moreover, it reports the connection between the bugs and contextual factors so that the bugs can later be reproduced. It utilizes the tools offered by Android SDK and logcat to inject events and capture traces of the app execution.

CCS CONCEPTS
• Computing methodologies → Simulation tools; • Software and its engineering → Integrated and visual development environments; Software testing and debugging; Empirical software validation;

KEYWORDS

2 BACKGROUND STUDY AND RELATED WORKS
As shown by Muccini et al. [16], mobile apps are different from traditional software and require specialized, new testing techniques. As a result, many different works were done including augmenting existing approaches and introducing novel approaches of mobile app testing. Several state of the art works focus on providing contextual testing as a cloud service [10], testing as a service [9], generating different input for Android apps [13], recording input events to later replay the same events to reproduce bugs [8], generating automatic GUI testing [1], on device bug reporting for android applications [14], generating contextual events for stress testing Android Apps [17], intent fuzzing [19] and, automatic discovering, reporting and reproducing mobile app crashes through static and dynamic analysis [15]. However, as surveyed in [12] by Vasquez et al., most of
Table 1: Event Types and Possible Values of MobiCoMonkey based on Android Guidelines [4]

<table>
<thead>
<tr>
<th>GSM Profile</th>
<th>Network Delay</th>
<th>Network Status</th>
<th>Key Events</th>
<th>User Rotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GSM</td>
<td>GSM HSDPA</td>
<td>Alphabets</td>
<td>Portrait</td>
</tr>
<tr>
<td>1</td>
<td>EDGE</td>
<td>HSCSD LTE</td>
<td>Numerals</td>
<td>Landscape</td>
</tr>
<tr>
<td>2</td>
<td>UMTS</td>
<td>GPRS EVDO</td>
<td>Symbols</td>
<td>Reverse</td>
</tr>
<tr>
<td>3</td>
<td>None</td>
<td>UMTS FULL</td>
<td>Back space</td>
<td>Landscape</td>
</tr>
<tr>
<td>4</td>
<td>EDGE</td>
<td>Delete</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

these approaches are not used in industry for various reasons; such as lack of applicability, heavy overhead for integrating in existing approach, and incompatibility with existing tools. Further, the author made several recommendations for future testing approaches such as automatic test cases that involve app, has low overhead to integrate with existing practices, and better trace-ability between test cases and features. **MobiCoMonkey** focuses on being a low overhead testing tool that utilizes non-modified Android SDK and interacts with the app in a non-obstructive manner. Therefore, it is possible to utilize it with other existing approaches. Further, it offers trace-ability between the injected contextual events and the resulting logs generated within the activities of the app. This allows the developer to zero-in on the possibly bug prone features.

3 THE MobiCoMonkey TOOL

*MobiCoMonkey* consists of several independent, configurable components. An overview of the components are discussed in subsection 3.1. Any developer can download it from GitHub [3] and use python3.6, JavaFX and pip to utilize its various components. Underneath, it utilizes various high level and low level elements of Android SDK, such as adb, emulator, shell, telnet and aapt. MobiCoMonkey is capable of injecting several low level contextual factors, which are: GSM Profile, Network Delay and Network Status. Additionally, it is capable of changing Rotation View, inserting Key Press events, and toggling Airplane Mode. The categories and values of currently supported events are provided in Table 1. It should be clarified that contextual network condition variation is only possible in emulated devices. Enabling support for such in actual android devices require high overhead modification of the Android OS and rooting. Therefore, **MobiCoMonkey** supports only android emulators as of now.

3.1 Components

The internal components of MobiCoMonkey are created using Python 3.6. These components are accessed by the MobiCoMonkey GUI to perform various types of actions. Due to various user configurable settings, MobiCoMonkey’s nature is quite flexible. The internal components are discussed as follows.

**3.1.1 App Manager:** The internal App Manager takes care of installing and uninstalling the app as required. It also exports permissions requested by the app to determine which **MobiCoMonkey** supported contextual factors will be applicable for contextual testing. Additionally, it extracts UI components of an activity by utilizing uiautomator dump when requested. To get the full list of User Interface (UI) elements, it scrolls from top to bottom of the screen until no new UI elements are found in the same activity. As a result, it becomes possible to create mapping of UI elements to each activity regardless of the screen size and view mode.

**3.1.2 Contextual Scenario Generator:** The contextual scenario generator is utilized when user defined scenarios are not provided for testing. Based on permissions, it generates random contextual scenarios based on a user provided seed value. Accordingly, the same scenarios can always be generated by utilizing the seed value. While generating, it considers minimum interval and maximum interval between contextual factor changes based on user configuration, and the total duration of each scenario. For example, if the
minimum and maximum intervals are configured to be 5 and 12.
two Network Status related events it may generate are:
NetworkStatus 0 5 hscsd
NetworkStatus 1 10 lte
Each line represents one contextual event. The first value describes
the type of contextual event, the second value is the index, the third
value is the interval after which the next Network Status event will
be applied and the fourth value is the current contextual event to
be applied. The combined duration of all contextual events for a
particular type matches the total duration of scenario. All the other
contextual scenarios are produced similarly.

3.1.3 Executor: This can utilize various approaches for context-
tual testing. For example, it can run in guided approach, where
it will run only selected activity screens of an app. Otherwise, it
will iterate through all the activities of the provided app. Before
executing any contextual events from the scenario, it starts logcat
service in the Emulator with a clean state. Next, for each activity it
runs the contextual scenario events. Consequently, each executed
event are saved in a log file for later analysis. To illustrate, a sample
log can be:
03-19 00:36:35 NetworkStatus 0 8 lte
03-19 00:36:43 NetworkStatus 1 5 gsm
03-19 00:37:05 UserRotation 0 8 ROTATION_REVERSE_POTRAIT
The generated log file follows a structure similar to logs generated
by logcat[5]. The executor can also additionally run an optional
text input field testing thread. Utilizing a top-down approach for
each activity, it inputs random text characters in text input fields.
Additionally, it stores progress of text fields covered while doing
so. Therefore, even if the view is suddenly rotated, it can continue
inserting text input sequentially. The executor furthermore watches
for fatal status in case of app crash. In such case, it stops execution
and saves available logs.

3.1.4 Log Analyzer: It combines logs generated by the logcat
evaluator and MobiCoMonkey executor. The logs are sorted based
on temporal ascending order. From each generated log, it
extracts the logs of the app at Warning, Error and Fatal level.
Generally, these logs are produced by either the system and/or
the developer as part of debugging. Warning is defined as a non-
obstructive malfunction which takes place when something does
not execute properly. Next, Failure or Error is found when an
internal activity crashes, while the app resumes execution. For
example, android.view.WindowLeaked is found when a dialog is
dismissed improperly. Lastly, Fatal is found when the app crashes
fatally and forfeits execution.

3.1.5 GUI Components. For ease of use, several Graphical User
Interface components are prepared as part of MobiCoMonkey using
JavaFX. These components are namely MobiCoMonkey GUI,
Contextual Scenario Creator, and Result Viewer as shown in Figure
1, 2 and 3 respectively.
Activities related to utilizing MobiCoMonkey can be categorized
to three segments, which are Setup & Configuration, Execution,
and Result Analysis. These are described in sub sections 3.2, 3.3,
and 3.4 respectively.

3.2 Setup & Configuration
To start, a developer has to download and follow instructions from
the GitHub [3] repository to initially set configuration values of
MobiCoMonkey in a config file. It is assumed that the user already
has Android SDK, Android Virtual Device and Android Emula-
tor installed in system. The configuration file is utilized by the
MobiCoMonkey GUI after user browses and selects the address of
the config file. Next, the user can utilize the Event Generator Win-
dow to create a custom scenario of Contextual Events. For each
entry of contextual events, the user has to insert the type of event,
the nature of the event and the duration of the event. For example,
if the user wants to add a Network Status of GSM stage, he simply
has to select it from the drop-down menu and input the delay before
the next event. While saving the file, the generator will convert it
into a Comma Separated Value(CSV) format that can be easily read
by the MobiCoMonkey Executor module. The user can also simply
ignore the Event Generator module and click on 'Run Test'. In such
case, the executor module will generate contextual scenarios based
on a user defined seed value automatically and will inject those for
a prefixed amount of time.
3.3 Execution

After clicking on Run Test at first, the app manager module will check for existing installation of the app installer file (apk) after initiating the Emulator and will install a fresh version if necessary. Second, the app manager will extract several meta-data related to the app, such as requested permissions and name of activities included in app. Based on extracted permissions, the executor will filter the events in the provided scenario. For example, an app that does not request permission to access the Internet or any data network might not be affected by a sudden contextual change in Network Status or Network Speed. Third, based on the extracted names of activities, the executor will extract further information related to User Interface, such as available text input fields. As a result, it is possible to traverse through activities in a top-down approach, while inserting textual elements in each input fields.

During execution, it creates detailed log similar to logcat[5] detail format. This log contains information related to android activities, UI components, and injected events. Additionally, it also utilizes the Android internal logcat to monitor the errors, warnings and fatal errors. All these information are stored continuously to prevent loss of data. Furthermore, in case of a fatal error it immediately stops execution and stops collecting unnecessary data.

3.4 Result Viewing

Using the MobiCoMonkey Result Viewer as shown in Figure 3, the information are categorized to Warning, Error, and Fatal levels. It then allows further filtering to android activities screens so that user is able to view details only from the necessary activities. By navigating to the appropriate activity, the user is able to view the events that occurred during testing. With it, the time-based adjacent, injected contextual events are provided so that user may deduce whether the derived execution event is influenced by the injected contextual events.

4 CONCLUSION

The strength of MobiCoMonkey lies in its simple, modular approach for handling different tasks of contextual mobile testing. An inexperienced user without any idea about contextual testing can use it similar to Monkey, where it injects random contextual events to an app. On the other hand, an experienced user can design custom contextual scenarios while utilizing any other kind of testing, such as UI testing or Unit testing. Furthermore, it allows the user to utilize it without modifying any existing tools in use or modifying the applications. As a result, it can be integrated as part of agile software methodology with very little overhead. In future, MobiCoMonkey can be extended for further analysis of app behavior so that causation and correlation can be derived from injected contextual events and observed behavior of the app. Additional contextual factors, such as GPS, Accelerometer and Bluetooth can be considered for future scope extension.

REFERENCES


