**Appendix E -Metrics used in Android apps reliability studies**

|  | **Purpose of measures** | **Attributes / Factors studied (as stated in the study)** | **Measures/Metrics** |
| --- | --- | --- | --- |
| S2, S14 | Analyse relationship between robustness/responsiveness errors and app popularity | robustness | # crashes per app |
| responsiveness | ANR (Application Not Responding) error count per app |
| popularity | app rating, app download count |
| S4, S28 | Analyse availability of a smartphone | aging | Free physical memory, heap memory usage, CPU usage |
| responsiveness | app response time/responsiveness (estimated based on its launch time) |
| availability | Sojourn time of the states (*Up*, *Degradation*, *Rejuvenation*, *Crash*) of the OS and applications |
| S7 | Provide a quantitative relationship between the mobile app store quality req. and the ISO/IEC 25010 quality characteristics. | Importance of app store quality requirement (reflects the needs of the customer) | DQW: Demanded Quality Weight (based on the frequency that each quality requirement may have on the analysed app stores), DQRW: Demanded Quality Relative Weight (the overall importance of the quality req. relative to the weights of all other quality requirements |
| Association of above and the quality characteristics in ISO/IEC 25010 | RV: Relationship Value ( the association between the demanded quality req. and the quality characteristics in ISO/IEC 25010), QCW: Quality Characteristic Weight (given by the sum of all RVs between each demanded quality req. and the analysed quality characteristic, multiplied by their corresponding DQRW), QCRW: Quality Characteristic Relative Weight (the weight of the quality characteristic relative to the weights of other characteristics) |
| S9 | Mining non-functional requirements from app reviews | Non-functional requirements from user perspective | TF: relative Term Frequency( the term frequency weight of a word), TF-IDF: Term frequency-inverse document frequency |
| S10 | Characterizing emergency update patterns | Update patterns | Emergency ratio of an update (Ui) : the ratio between the lifetime of the Ui-1 update and the median lifetime of all the updates for that app, lifetime of emergency updates: the time difference (in days) between the update date of the update Ui and the update of the next update U*i+1*, the ratio of negative reviews (reviews with one or two star rating) for an app. |
| S13 | Explore user experience through code quality metrics | User experience | *Unchecked Bundles*: # Bundles in each app, # Checked Bundles in each app, # Unchecked Bundles in each app*, Token Exceptions*: # potential bad token exception, *Fragments*: # fragments in each app, *Separation of View and Controller* *of MVC* : # views in controllers, # views not in controllers, the % of views defined outside of controllers |
| S15, S36 | Construct usage independent quality metric | usage independent quality metric for apps | *Release.Date*: Start Date for the release, *Exceptions*: # exceptions, *New.Users:* # new users, *Release.Duration*: Effective duration of the release, *Usage.Intensity*: Average time on site per user, *Usage,Frequency*: # visits per user |
| S16, S19, S72 | Measure the declared SDK versions and their consistency with API calls | SDK versions | *minSdkVersion*: the minimum platform API level required for an app to run, *targetSdkVersion*: the platform API level that an app targets at., *maxSdkVersion*: the maximum platform API level on which an app can run |
| API levels | *minLevel* : minLevel of an app is the maximum of all its valid API calls' corresponding minLevel values, *maxLevel* : maxLevel of an app is the minimum of all valid API calls' corresponding maxLevel values |
| S20 | Analyse the relationship between source code changes and uncaught exceptions | Size of source code | *TotalLOC*: # lines of normal and exceptional code for each version |
| Robustness | *Uncaught Flow*: # exception flows that leave the bounds of the system without being handled, *Relative UncaughtFlow*: the relative change in the number of uncaught flows between pairs of versions |
| Change impact | *EHChurnedLOC*: # lines of exceptional code added or changed between a pair of versions, *InterfaceAdded, InterfaceChanged, InterfaceRemoved*: respectively, the # methods that added, changed or removed exceptional interfaces between a pair of versions*, InterfaceTypeAdded, InterfaceTypeChanged, InterfaceTypeRemoved*: respectively, # exception types that were added, changed, or removed from exceptional interfaces, *ClassChurned*: # classes that have an exception interface or catch block added, changed, or removed between a pair of versions, *MethodChurned*: # of methods that have an interface or catch block added, changed, or removed between a pair of versions, *NormalChurnedLOC*: the sum of the added and changed lines of normal code for a pair of versions. |
| S21 | Automatically Identify incorrect permission |  | *Damerau-Levenshtein distance: measures the similarity between two input strings.* |
| S23 | To analyse how different types of bugs occur together and if fixing them affects the remaining bugs. | Reliability, Security, Maintainability, Performance | *# reliability bugs* (code errors that prevent the program from operating as intended ), # security bugs(problems in the code that could be exploited by a bad actor to compromise the *security* of the application) and # maintainability bugs (code smells) as reported by SonarCloud,  Measures after fixing bugs : # fixed bugs per app, # cases no new bugs appeared, # cases more than one bug disappeared, # cases where new bugs appeared, # cases where same type of bugs appeared, # cases where different type of bugs appeared, average CPU and memory usage before and after fixing each bug |
| S25 | Analyse the relationship between the app development approach and its user-perceived quality. | performance, usability, security or reliability from user perspective | *Complaint Density(qc)= ,* where *qc* is a quality concern ( i.e., performance, usability, security or reliability) |
| S27 | analyze the availability and user experience to determine the optimal rejuvenation time interval. | Availability  User Experience | *App response time,* sojourn time of the states (*Sleep*, *Active*, *Young*, *Old*, *Recovering*, *Rebirth*) |
| S38 | Relationship between usage of Android abstractions and uncaught exceptions | Size of source code | *NClasses****: #*** classes of each version that extend Android-specific abstractions (*Activity* , *Service* , *AsyncTask* and *BroadcastReceiver*), *NMethodCall****: #***  method calls related to each abstraction, *NTry: #*  try blocks defined in the scope of classes that extend Android-specific abstractions, *DensityTry****:*** is defined as *NTry* divided by *NClasses.* |
| Robustness | *Uncaught Flow (as S20)* |
| Change impact | *EHChurnedLOC* *(as S20)* |
| S39 | Evaluate adequacy of DroidEH for improving robustness | Robustness/fault tolerance | *# crashes, # uncaught exceptions* |
| S41 | To analyse how complementary user feedback and testing tools are. | User perspective quality | % of issues reported in both reviews and stack traces , % of issues reported only in user reviews , % of issues reported only in stack traces , where *L:* # links detected between a review and a stack trace, *S* : # stack traces that was not possible to link any reviews, *R* : # reviews that were not linked to any stack traces and *T* : total # unique issues reported for a given app( T=L+S+R) |
| S45 | To understand the characteristics of exception fixes by developer |  | *Issue Duration*: how long the developers took to fix the issue, *Changed Code Lines*: # code lines the developers changed to fix this issue, *Issue Closing Rate*: i.e., how many issues have been closed. |
| S46 | Analyse API misuse rate |  | *Misuse Rate:* # misused API from the used APIs*, Misuse Frequency:* Total # misused APIs from the total # used Android APIs |
| S50 | Analyse frequently recurring exceptions to prioritize maintenance tasks |  | *Total # exceptions, # unique exceptions, # occurrences of the same exception, # occurrences of the same exception from the same class* |
| S56 | Undocumented exceptions |  | PCSU ( i.e., P ∩ C ∩ S ∩ U) : pairs of API methods and exceptions that appear in the three sets, App Programs (P), Crashes (C), and API Source Code (S), are **not** in the API reference documentation (U). (if an exception belongs only to one set (e.g. PU = P∩U, CU = C∩U ), the evidence that the exception should be documented is less strong) |
| S57 | Analyzed how the ratings of app correlated with the  fault- and change-proneness of the APIs such app relied upon | Change-proneness | changes in API: # method changes, # changes in method signatures, # changes to the set of exceptions thrown by methods |
| Fault-proneness | # bugs/crashes fixed in the APIs |
| 61 | Analyse correlation /association between method calls and exceptions |  | # usages of API method (*m)*, # usages of API exception (*E)*, # times that exception E occurs on code snippets that contains method (m) |
| 66 | Exception fault localization | to rank lines of application source code with respect to their  relevance to that fault based on test case traces | *Value pattern score* (*V* *P*(*l*)): For each variable in the application’s source code and each test case, a sequence of pairs of < number; value > (i.e., value pattern)is recorded. Each line of code is marked as unrelated to an exception if and only if all the variables used in that line have been marked as unrelated variables. The value of *V P*(*l*) would be 1 for related lines, and 0 for others.  *Backward static slicing score (Slice(l)):* The static slice of line *l* is the set of program’s lines of code that may affect the values of variables in line *l*. This score is used to highlight related lines to the faulty line.  Test case score : Execution frequency of each line in the failed and successful test cases. |
| 68 | Analyse root causes of ANR and SNR events | Unresponsiveness / User experience | # ANR, SNR events, Timestamp of those events, Call stacks of set of app and system service processes (e.g., SystemServer and MediaServer etc.), Feature info. of call stacks: {CPU usage, memory usage, Java functions, native libraries, kernel functions, process names, # locks, length of the wait-for graph} *similar-stack analysis:* similarity between feature vectors of ANR and SNR events. |
| 73 |  | User experience/reliability | App launch time |
| 74 |  | User experience/reliability | Memory usage |
| 77 | Predict crashing releases | Complexity | *Cyclomatic*: # branching paths within code in all the source code files in a release |
| Time | *PreDays*: # of days since the previous release |
| Code | *LA*: # lines added in a release, *LD*: # lines deleted in a release, *SIZE:* total # lines of code in the current release, *SAME*: # source code files that are modified by both the current and the previous release, *CUR\_file*: # source code files in the current release, *PREV\_file:* # modified source code files in the previous release. |
| Diffusion | *Top\_NS:* # unique subsystems changed between two releases, *Bottom\_NS*: # unique subsystems changed between two releases, NF: # unique files that have changed between two releases, *File\_entropy*: Distribution of modified files across the release, *Churn\_entropy*: Distribution of modified code across the application |
| Commit | *NC:* # commits, *NFC:*# commits which fix bugs |
| Text | *Fuzzy\_score:* Fuzzy set scores of commit logs, *NB\_score*: Naive Bayes scores of commit logs, *NBM\_score*: Naive Bayes Multinomial scores of commit logs, *DMN\_score*: Discriminative naive Bayes Multinomial scores of commit logs, COMP\_score: Complement naive Bayes scores of commit logs |
|  |  |  |  |
| s82 |  | Responsiveness and user experience | *fluency, user experience degree(USD), UI response times, soujan time* |
| s83 | Locate crashing faults |  | *Similarity between two Crashes* |
| s8*5* | API compatibility |  | *minSdk, maxSdk, targetSdk* |
| s87 | Recommend Stack Overflow Posts for Fixing Runtime Exceptions |  | *similarity score: similarity between the*  *developer’s buggy code snippet and a SO post’s question code*  *snippet* |