Method and system for monitoring quality control of an application

FIELD OF INVENTION

1. The present disclosure relates broadly, but not exclusively, to methods and systems for monitoring quality control of an application.

BACKGROUND

2. Quality assurance of a software application such as a mobile application is of fundamental importance in ensuring that a user of the mobile application can enjoy a smooth and bug-free experience. Strong quality control and maintenance ensures that a mobile application stays competitive, especially in the huge app market now prevalent and easily accessed via the internet, and app platforms such as GooglePlay™, Appstore™, Amazon™, and other similar platforms.

3. It is common for an application to experience problems such as bugs, crashes, compatibility issues, stability issues and other similar problems in its lifetime. This may be due to users of the application having new hardware or devices to run the application, or installing other new software that may have conflict with the concerned application. It may also be issues arising from an operating system or platform having difficulty running the application, for example due to new, unexpected problems from an update of the operating system, platform or the application. Information of these issues are crucial for a developer of the application to update the application and rectify the issues. However, in real life cases, platform issues are difficult to identify especially during the application development phase or testing phase, as typically such information can only be retrieved from crash reporting dashboards or via CE tickets.

4. Some developers would do weekly regression testing on their applications to ensure good stability. However, identifying an impact area of an application at which the regression testing should be focused is also challenging, as it is typically a manual process where engineers share documents with for example a quality assurance team regarding the impact areas to be tested. In the process of sharing

such information, missing or incorrect details may be shared, which causes the regression testing to be unreliable.

5. Another issue faced by an application developer is how to correctly identify an unstable feature or module in the application for testing. This is also a difficult process as misinformation may result in an incorrect feature being tested. Testing a feature of an application with a right set of devices is also very important as it ensures that the application is 100% compatible with all the devices, which reduces the chances of high severity crash issues and in turn increase the user experience. This is especially important for mobile applications as a developer would want their mobile application to run stably and smoothly on as many devices as possible. This is made even more challenging with so many mobile devices being available in the market.

6. A need therefore exists to provide methods and systems that seek to overcome or at least minimize the above mentioned challenges.

SUMMARY

7. According to a first aspect of the present disclosure, there is provided a method for monitoring quality control of an application, comprising: identifying one or more files that are affected by a proposed change initiated by a user, the proposed change being initiated by a change request, the change request being a merge request comprising data relating to the one or more files and information provided by the user relating to the proposed change, the information further indicating user details relating to the user, wherein the identification is based on the data; determining an impact area of the application that is affected by the proposed change based on the identified one or more files and the information; and merging the merge request with source code of the application based on a check whether the information is in the merge request.

8. According to a second aspect of the present disclosure, there is provided a system for monitoring quality control of an application, the system comprising: at least one processor; and at least one memory including computer program code; the at least one memory and the computer program code configured to, with the at least one processor, cause the system at least to: identify one or more files that are affected by a proposed change initiated by a user, the proposed change being initiated by a change request, the change request being a merge request comprising data relating to the one or more files and information provided by the user relating to the proposed change, the information further indicating user details relating to the user, wherein the identification is based on the data; and determine an impact area of the application that is affected by the proposed change based on the identified one or more files and the information; and merge the merge request with source code of the application based on a check whether the information is in the merge request.

BRIEF DESCRIPTION OF THE DRAWINGS

9. Embodiments and implementations are provided by way of example only, and will be better understood and readily apparent to one of ordinary skill in the art from the following written description, read in conjunction with the drawings, in which:

10. Figure 1 is a schematic diagram of a computing device. The computing device can be implemented as a system for monitoring quality control of an application, according to various embodiments of the present disclosure.

11. Figure 2 is a schematic diagram of a system for monitoring quality control of an application, according to various embodiments.

12. Figure 3 is a flow diagram illustrating a process for monitoring crash logs according to an embodiment.

13. Figure 4 is a flow diagram illustrating how a merge request (MR) is processed according to an embodiment.

14. Figure 5 is an exemplary diagram showing how a stability report may be generated according to an embodiment.

15. Figure 6 is an exemplary diagram showing how a web crawler may be utilized for monitoring quality control of an application according to an embodiment.

16. Figure 7 shows a flow diagram of how a business vertical report is generated according to an embodiment.

17. Figure 8 shows a flow diagram of how a suitable test device is determined according to an embodiment.

18. Figure 9 is a flow chart illustrating a method for monitoring quality control of an application, according to various embodiments.

19. Figure 10 is a block diagram of a computer system suitable for use as a system for monitoring quality control of an application as shown in Figures 1 to 9.

20. Skilled artisans will appreciate that elements in the figures are illustrated for simplicity and clarity and have not necessarily been depicted to scale. For example, the dimensions of some of the elements in the illustrations, block diagrams or flowcharts may be exaggerated in respect to other elements to help to improve understanding of the present embodiments.

DETAILED DESCRIPTION

21. Embodiments will be described, by way of example only, with reference to the drawings. Like reference numerals and characters in the drawings refer to like elements or equivalents.

22. Some portions of the description which follows are explicitly or implicitly presented in terms of algorithms and functional or symbolic representations of operations on data within a computer memory. These algorithmic descriptions and functional or symbolic representations are the means used by those skilled in the data processing arts to convey most effectively the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities, such as electrical, magnetic or optical signals capable of being stored, transferred, combined, compared, and otherwise manipulated.

23. Unless specifically stated otherwise, and as apparent from the following, it will be appreciated that throughout the present specification, discussions utilizing terms such as “monitoring”, “utilizing”, “retrieving”, “providing”, “generating”, “quantifying”, “calculating”, “outputting”, “optimising”, “rebuilding”, “storing”, ”mapping”, “checking”, “identifying”, “collecting”, “searching”, “conducting”, “cross-checking”, “aggregating”, “determining”, “regenerating”, “updating”, “comparing”, “adjusting”, “compiling”, “performing”, “obtaining” or the like, refer to the action and processes of a computer system, or similar electronic device, that manipulates and transforms data represented as physical quantities within the computer system into other data similarly represented as physical quantities within the computer system or other information storage, transmission or display devices.

24. In addition, the present specification also implicitly discloses a computer program, in that it would be apparent to the person skilled in the art that the individual steps

of the method described herein may be put into effect by computer code. The computer program is not intended to be limited to any particular programming language and implementation thereof. It will be appreciated that a variety of programming languages and coding thereof may be used to implement the teachings of the disclosure contained herein. Moreover, the computer program is not intended to be limited to any particular control flow. There are many other variants of the computer program, which can use different control flows without departing from the scope of the specification.

25. Furthermore, one or more of the steps of the computer program may be performed in parallel rather than sequentially. Such a computer program may be stored on any computer readable medium. The computer readable medium may include storage devices such as magnetic or optical disks, memory chips, or other storage devices suitable for interfacing with a computer. The computer readable medium may also include a hard-wired medium such as exemplified in the Internet system, or wireless medium such as exemplified in the GSM mobile telephone system. The computer program when loaded and executed on such a computer effectively results in an apparatus that implements the steps of the preferred method.

26. A quality assurance platform is a data aggregator that monitors and collects data from multiple sources, and processes the data for implementing various quality control measures for an application. The multiple sources may comprise other platforms, databases, source code repositories, or other similar sources. Such sources may include, for example, Firebase™, Android vitals™, Gitlab™, Metabase™, issue reporting forums (e.g. for Google™, Apple™, Amazon™ or other similar entities) and other similar sources. The quality assurance platform may be configured to continuously monitor and collect data from the various sources relating to problems that a concerned application may have or may be affected by such as bugs, crashes, compatibility issues, stability issues and other similar problems. The collected data may be utilized to finetune quality assurance (QA) processes for the application, and/or provide an indication on which part of an application should the QA efforts and resources be focused on. The quality assurance platform may also periodically generate reports comprising, for example, crash reports, known issues and impact area for each business vertical which help to effectively channelize the QA bandwidth. A business vertical associated with an application or feature refers

to a particular market that the application or feature serves. For example, an application or feature may be catered for customers in insurance, transport, food delivery, payment instruments, or other type of markets. The quality assurance platform may also function as and be referred to as a dashboard wherein engineers, QA team and other personnel in a development team of the concerned application may utilize for monitoring any updates regarding the application (e.g. new issues and impact area, resolved issues, regression testing results, areas of application to focus regression testing on, and other similar information).

27. An application development platform enables developers to write and develop applications for web, Android or iOS without having to worry about server-side programming. Such platforms typically also provide valuable information such as issue reports relating to a developed application that has been released to the market that can be used for updating and improving the application. Non-exhaustive examples of an application development platform include Firebase™, Back4App™ , Android Studio™, Xcode™ or Parse™.

28. A source code repository is a data structure that stores metadata for a set of files or a directory structure of, for example, an application. Information such as a historical record of changes made to the set of files or directory structure, a file owner of each file or code, an associated business vertical and other information relating to the files may be stored in the source code repository. The quality assurance platform may cross-check data with a source code repository of an application to identify an owner of a code, feature, or file of the application, as well as an associated business vertical and impact area relating to a crash report of the application. Non-exhaustive examples of a code control repository include Github™, GitLab™, Bitbucket™ or Jenkins™.

29. An analytics platform is a platform that enables analysis of data for application development and regression testing of an application. The analytics platform enables developers to gain insight about how good an application or feature is performing by, for example, tracking user activity to help the product and business to

serve better. For example, a new feature may be released and the analytics platform may help track how many users are visiting a particular web page in relation to this new feature. The analytics platform may also be used to assess what are the revenue impacts of a new product or feature. An important capability of an analytics platform is page view tracking, wherein the quality assurance platform can use such data to set a severity of an issue e.g. a high page view count of a webpage relating to a particular issue may indicate a high severity, and vice versa. For example, the quality assurance platform may utilize the analytics platform to analyze and assess an impact and severity of a crash that is reported in a crash log. Non-exhaustive examples of an analytics platform include Metabase™, Google Analytics™ or Crashlytics™.

30. An issue tracking program enables issues relating to an application such as crashes, bugs, and other similar issues to be tracked to facilitate agile project management and application development. The quality assurance platform may utilize the issue tracking program to identify issues relating to regression testing which can be used to set the business vertical stability. Non-exhaustive examples of an issue tracking program include Jira™, Zepel™ or Pivotal Tracker™.

31. Figure 1 illustrates a schematic diagram of a computing device 100. The device 100 can be implemented as a system for monitoring quality control of an application as discussed herein, according to embodiments of the present disclosure.

32. The device 100 at least includes a processor 102 and a memory 104. The processor 102 and the memory 104 are interconnected. The memory 104 includes computer program code (not shown in Figure 1). The memory 104 and the computer program code are configured to, with the processor 102, cause the device 100 to perform the steps for monitoring quality control of an application as described in the following paragraphs of the present disclosure.

33. Figure 2 depicts an overview of various sources of information that may be utilized by the system for monitoring quality control of an application, according to various embodiments. The system comprises a quality assurance platform 202. The quality assurance platform 202 may be configured to monitor and collect data from multiple sources, and processes the data for implementing various quality control

measures for an application. For example, the quality assurance platform 202 may monitor and collect data from an application development platform 204 which was used to develop the concerned application. The application development platform 204 may be, for example, Firebase™, Android vitals™ or other similar application development platform. The monitoring and collection of data may be based on a stream of issue reports or crash logs relating to the application (and may include issue reports relating to other applications) that is provided by the application development platform 204 or from a database of the application development platform 204. Such data may include information relating to crash reports concerning the application. For example, when the quality assurance platform 202 monitors the application development platform 204 and detects that a crash is reported, data relating to the crash report is collected and processed to determine an affected file of the reported crash, as well as an owner of the affected file. The owner of the affected file may be identified by cross-referencing with a source code repository 206 of the concerned application which typically stores fileowner details for each file of the application. The source code repository 206 may be, for example, GitLab™, GitHub™, Bitbucket™, Jenkins™ or other similar repositories. The quality assurance platform 202 may also cross-check the data with an analytics platform 208 to determine a severity level of a reported crash.

34. The quality assurance platform 202 may also utilize an issue tracking program 210 to keep track of issues relating to the application and to identify issues relating to regression testing which can be used to set the business vertical stability. The issue tracking program 210 may be, for example, Jira™, Zepel™ or Pivotal Tracker™. Further, a web crawler 212, which is a program or bot that may be programmed to systematically browse the world wide web or specific areas of the world wide web, may be used by the quality assurance platform 202 to monitor issue reporting forums (e.g. for Google™, Apple™, Amazon™ or other similar entities depending on the application) and generate reports of the monitoring results. These reports may then be analysed manually or via AI to determine information of each issue identified in the reports such as application programming interface (API) Level, device, type of issue, and other similar information relating to the identified issue.

35. Figure 3 is a flow diagram 300 illustrating a process for monitoring crash logs according to an embodiment. At step 302, the quality assurance platform 202 monitors a crash log for reports or notifications relating to issues of the concerned application such as a failure to function properly. The crash log may be a stream of reports from an application development platform 204 or from a database of the application development platform. The monitoring may include crash logs from one or more databases, e,g, the quality assurance platform monitors one or more databases for a notification relating to a failure of the application to function properly.

36. At step 304, the crash log is processed to identify a notifications or report relating to failure of the application to function properly. The identified notification or report may comprise information indicating details of the failure and indicating a file and an owner of the file that is related to the failure. Information of the file and owner of the file may be cross-checked with a source code repository 206 relating to the application. At step 306, it is determined whether the file owner details are available in the source code repository 206. If the details are not available, the process proceeds to step 308 wherein the failure is reported by the quality assurance platform 202 as a platform issue (e.g. an issue relating to the application development platform 204) with the application development platform team indicated as the owner, and the report is saved in a database 310 of the quality assurance platform 202. If the file owner details are available, the process proceeds to step 312 where it is determined whether active session details relating to the failure are available in the source code repository 206. If there is no session information found relating to the failure, the process proceeds to step 314 where the failure is reported by the quality assurance platform 202 with a default severity level. The quality assurance platform 202 then saves details of the reported failure into the database 310. On the other hand, if session details are found in the source code repository 206 at step 312, the process proceeds to step 314 where the failure is reported with a severity level based on the active session details found in the source code repository 206. For example, information of the active session may be checked with an analytics platform 208 to determine the appropriate severity level for the failure. The quality assurance platform 202 then saves details of the reported failure into the database 310.

37. Further, such monitoring can be performed by the quality assurance platform 202 on a periodic basis, and the quality assurance platform 202 may generate, for each period of monitoring, a report documenting any notification relating to failure of the application to function properly found during the period of monitoring. Advantageously, this process enables identification of any issues relating to an application at a very early stage, without waiting for a crash threshold before the development and QA team of the application takes action.

38. Figure 4 is a flow diagram 400 illustrating how a merge request (MR) is processed by the quality assurance platform 202 according to an embodiment. A MR may be a change request that is generated to initiate a proposed change relating to an application. For example, the proposed change may be a change in code and/or change in one or more files of the application. The proposed change may be in response to an issue that is found from the monitoring of notification of failures in the process 300, or made during regression testing of the application, or made as a possible improvement to the application, or other similar reasons. At step 402, a MR or change request is generated for the source code repository 206 of the application. At step 404, the MR request is approved by the source code repository, and the proposed change relating to the application as indicated in the MR or change request is due to be merged with source code of the application that is stored in the source code repository 206. At step 406, an impacted business vertical and an owner of the MR are determined. The change request comprises data relating to one or more files and information relating to the proposed change. The quality assurance platform 202 identifies one or more files that are affected by the proposed change based on the data, and the impacted business vertical is then identified based on the identified one or more files. The owner of the MR or change request is the person who is responsible for generating the MR or change request, or may also be referred to as a user e.g. a user of the quality assurance platform 202.

39. At step 408, an impact area is determined. The impact area is an area of the application that is affected by the proposed change, and the determination is based on the identified one or more files, the impacted business vertical and the infor-

mation relating to the change request. The information relating to the change request is to be provided by the MR owner, for example in a form of a short description explaining the proposed change. At step 410, it is determined whether the information from the MR owner is available in the change request. If this information is not available, the MR request is not allowed to proceed (e.g. the proposed change is not allowed to be merged with the source code of the application in the source code repository 206) and the MR request is reverted back to the source code repository 206, wherein the MR owner is provided a chance to input the information. If the information from the MR owner is found to be available at step 410, the MR or change request is merged with the source code of the application. A report documenting details of the MR or change request is also generated and then stored in database 412 of the quality assurance platform. The report may be stored in a respective category based on the impacted business vertical and owner or user details of the MR or change request. Advantageously, this enables accountability for any source code changes, and such changes can be accurately tracked and recorded.

40. Figure 5 is an exemplary diagram 500 showing how a stability report may be generated according to an embodiment. At step 504, reports stored in database 502 of the quality assurance platform 202, and regression issues that are captured and logged in the issue tracking program 210, are both compiled. The reports stored in the database 502 may, for example, relate to issues found during monitoring and data collection as illustrated in Fig. 2, and may comprise reports that are generated and stored during processes 300 and 400. Regression issues that are captured and logged in the issue tracking program 210 may be in the form of reports that are generated and stored during regression testing of the application.

41. At step 506, a weekly delta is determined, which may be a measure of how areas of the application that are affected by issues have reduced, increased or changed on a weekly basis, or how the code of the application has changed on a weekly basis due to, for example, bug fixes or feature integration. At step 508, a stability report is generated, the stability report indicating a measure of how stable the application is, which is based on the determined weekly delta. The stability report enables a development team to have a measure for how stable an application is, based on changes in quantity of issues faced by the application overtime. It will be

appreciated that, instead of being determined weekly, the delta may also be determined daily, monthly, biweekly, or other similar periods.

42. Figure 6 is an exemplary diagram 600 showing how a web crawler may be utilized for monitoring quality control of an application according to an embodiment. At step 602, a web crawler 212, which is a program or bot that may be programmed to systematically browse the world wide web or specific areas of the world wide web, may be used by the quality assurance platform 202 to monitor issue reporting forums (e.g. for Google™, Apple™, Amazon™ or other similar entities depending on the application) for any reported issues relating to the application. At step 604, it is determined whether platform details relating to reported issues found by the web crawler 212 can be extracted. If it is determined that the extraction is not possible (e.g. no platform details were indicated in the report issue, or the details are not clearly indicated), the process proceeds to step 606 where a manual check is conducted to identify the platform related to the reported issue. The results of the manual check are compiled into a report and stored in database 608 of the quality assurance platform 202. At step 610, copy of the report is also sent to a QA team for the application.

43. On the other hand, if it is determined that the platform details can be extracted, the process proceeds to step 612 where it is determined whether an automated extraction of information should be triggered to analyze and compile all the information that the web crawler 212 has obtained. If it is determined that automation should not be used, the process proceeds to manual checking at step 606 as previously explained. Otherwise, automated extraction is triggered and the extracted information is compiled into a report and stored in the database 608, wherein a copy of the report is also sent to the QA team for the application at step 610. Advantageously, having a manual check provides a contingency in case automated extraction of platform details from issues found by the web crawler is not possible.

44. Figure 7 shows a flow diagram 700 of how a business vertical report is generated according to an embodiment. A business vertical report provides an insight to the quality as well as stability of an application or feature that is associated with a business vertical. The QA team can realign their bandwidth to focus their efforts

efficiently based on the business vertical report. For example, if the business vertical report indicates that issues relating to an associated application or feature is relatively low or zero, then the QA team can channelize bandwidth to test new features or other quality improvement aspects for the application, rather than release build testing. Through the monitoring and information collection process described in figures 2-6, database 702 of the quality assurance platform 202 would have stored various information and reports thereof in relation to identified issues or changes made to the concerned application, for example in relation to change logs 704 captured for change requests, business vertical crashes 706 and platform issues 708 identified by web crawl. These information may be compiled in process 710 to generate a business vertical report.

45. Testing a new or changed feature with the right set of devices is very important as it ensures that the application is compatible with all of them which reduces the chances of high severity crash issues and in turn improves the user experience. Figure 8 shows a flow diagram 800 of how a suitable test device is determined according to an embodiment. At step 802, device details such as device model number, hardware specifications, operating system details, software specifications, or other similar information may be extracted from processing one or more comma-separated values (CSV) files that are stored in the application development platform 204. The application development platform may be, for example, Firebase™. As there is no regional splitting of devices for Firebase™, the device details contained in the CSV files from Firebase™ are not segregated into regions. At step 804, analytics platform 208 may be queried for further information relating to the devices identified in step 802. For example, the analytics platform 208 may be Metabase™ which has detailed reports about device locations. Therefore, such information can be used to group the devices identified in step 802 regionally.

46. With the combined information obtained from steps 802 and 804, a report may be generated at step 806 by the quality assurance platform. The information can be used to determine one or more suitable devices that will be used for testing a new or changed feature, and/or for regression testing of an impact area of the application. For example, one or more devices may be identified based on a determined impact area, and a report may be generated indicating the identified devices. Regression testing may then be performed for the determined impact area for the

application on the identified devices. Any issues that may occur during the regression testing may then be documented and generated as a report.

47. Figure 9 is a flow chart 900 illustrating a method for monitoring quality control of an application, according to various embodiments. At step 902, one or more files that are affected by a proposed change initiated by a user are identified, the proposed change being initiated by a change request, the change request comprising data relating to the one or more files and information relating to the proposed change, wherein the identification is based on the data. At step 904, an area of the application that is affected by the proposed change is determined based on the identified one or more files and the information.

48. Figure 10 is a block diagram of a computer system suitable for use as a system for monitoring quality control of an application as shown in Figures 1 to 9. The following description of the computer system / computing device 1000 is provided by way of example only and is not intended to be limiting.

49. As shown in Figure 1000, the example computing device 1000 includes a processor 1004 for executing software routines. Although a single processor is shown for the sake of clarity, the computing device 1000 may also include a multi-processor system. The processor 1004 is connected to a communication infrastructure 1006 for communication with other components of the computing device 1000. The communication infrastructure 1006 may include, for example, a communications bus, cross-bar, or network.

50. The computing device 1000 further includes a main memory 1008, such as a random access memory (RAM), and a secondary memory 1010. The secondary memory 1010 may include, for example, a hard disk drive 1012 and/or a removable storage drive 1014, which may include a magnetic tape drive, an optical disk drive, or the like. The removable storage drive 1014 reads from and/or writes to a removable storage unit 1018 in a well-known manner. The removable storage unit 1018 may include a magnetic tape, optical disk, or the like, which is read by and written to by removable storage drive 1014. As will be appreciated by persons skilled in the relevant art(s), the removable storage unit 1018 includes a computer readable storage medium having stored therein computer executable program code instructions and/or data.

51. In an alternative implementation, the secondary memory 1010 may additionally or alternatively include other similar means for allowing computer programs or other instructions to be loaded into the computing device 1000. Such means can include, for example, a removable storage unit 1022 and an interface 1020. Examples of a removable storage unit 1022 and interface 1020 include a removable memory chip (such as an EPROM or PROM) and associated socket, and other removable storage units 1022 and interfaces 1020 which allow software and data to be transferred from the removable storage unit 1022 to the computer system 1000.

52. The computing device 1000 also includes at least one communication interface 1424. The communication interface 1024 allows software and data to be transferred between computing device 1000 and external devices via a communication path 1026. In various embodiments, the communication interface 1024 permits data to be transferred between the computing device 1000 and a data communication network, such as a public data or private data communication network. The communication interface 1024 may be used to exchange data between different computing devices 1000 which such computing devices 1000 form part an interconnected computer network. Examples of a communication interface 1024 can include a modem, a network interface (such as an Ethernet card), a communication port, an antenna with associated circuitry and the like. The communication interface 1024 may be wired or may be wireless. Software and data transferred via the communication interface 1024 are in the form of signals which can be electronic, electromagnetic, optical or other signals capable of being received by communication interface 1024. These signals are provided to the communication interface via the communication path 1026.

53. Optionally, the computing device 1000 further includes a display interface 1002 which performs operations for rendering images to an associated display 1030 and an audio interface 1032 for performing operations for playing audio content via associated speaker(s) 1034.

54. As used herein, the term "computer program product" may refer, in part, to removable storage unit 1018, removable storage unit 1022, a hard disk installed in hard disk drive 1012, or a carrier wave carrying software over communication path 1026

(wireless link or cable) to communication interface 1024. Computer readable storage media refers to any non-transitory tangible storage medium that provides recorded instructions and/or data to the computing device 1000 for execution and/or processing. Examples of such storage media include floppy disks, magnetic tape, CD-ROM, DVD, Blu-rayTM Disc, a hard disk drive, a ROM or integrated circuit, USB memory, a magneto-optical disk, or a computer readable card such as a PCMCIA card and the like, whether or not such devices are internal or external of the computing device 1000. Examples of transitory or non-tangible computer readable transmission media that may also participate in the provision of software, application programs, instructions and/or data to the computing device 1000 include radio or infra-red transmission channels as well as a network connection to another computer or networked device, and the Internet or Intranets including e-mail transmissions and information recorded on Websites and the like.

55. The computer programs (also called computer program code) are stored in main memory 1008 and/or secondary memory 1010. Computer programs can also be received via the communication interface 1024. Such computer programs, when executed, enable the computing device 1000 to perform one or more features of embodiments discussed herein. In various embodiments, the computer programs, when executed, enable the processor 1004 to perform features of the above-described embodiments. Accordingly, such computer programs represent controllers of the computer system 1000.

56. Software may be stored in a computer program product and loaded into the computing device 1000 using the removable storage drive 1014, the hard disk drive 1012, or the interface 1020. Alternatively, the computer program product may be downloaded to the computer system 1000 over the communications path 1026. The software, when executed by the processor 1004, causes the computing device 1000 to perform functions of embodiments described herein.

57. It is to be understood that the embodiment of Figure 10 is presented merely by way of example. Therefore, in some embodiments one or more features of the computing device 1000 may be omitted. Also, in some embodiments, one or more features of the computing device 1000 may be combined together. Additionally, in some embodiments, one or more features of the computing device 1000 may be split into one or more component parts.

58. It will be appreciated by a person skilled in the art that numerous variations and/or modifications may be made to the present disclosure as shown in the specific embodiments without departing from the scope of the specification as broadly described. The present embodiments are, therefore, to be considered in all respects to be illustrative and not restrictive.

CLAIMS

What is claimed is:

1. A method for monitoring quality control of an application, comprising:

identifying one or more files that are affected by a proposed change initiated by a user, the proposed change being initiated by a change request, the change request being a merge request comprising data relating to the one or more files and information provided by the user relating to the proposed change, the information further indicating user details relating to the user, wherein the identification is based on the data;

determining an impact area of the application that is affected by the proposed change based on the identified one or more files and the information; and

merging the merge request with source code of the application based on a check whether the information is in the merge request.

2. The method according to claim 1, wherein the step of determining the impact area comprises:

identifying an impacted business vertical based on the identified 1 or more files, and determining the impact area based on the impacted business vertical.

3. The method according to claim 2, the method further comprising:

generating a report relating to the change request, and

storing the report in a dashboard database based on the impacted business vertical and the user details.

4. The method according to claim 1, further comprising:

monitoring one or more databases for a notification relating to failure of the application to function properly, the notification comprising information indicating details of the failure and indicating a file and an owner of the file that is related to the failure;

determining a severity level of the failure by checking the information with an analytics platform; and

reporting the notification and determined severity level to one or more users.

5. The method according to claim 4, wherein the monitoring is done on a periodic basis, further comprising:

generating, for a period of monitoring, a report documenting any notification relating to failure of the application to function properly found during the period of monitoring.

6. The method according to claim 1, further comprising:

identifying one or more devices based on the determined impact area, and

generating a report indicating the identified devices.

7. The method according to claim 6, further comprising:

performing regression testing for the determined impact area for the application on the identified devices, and

generating a report documenting any issues during the regression testing.

8. A system for monitoring quality control of an application, the system comprising:

at least one processor; and

at least one memory including computer program code;

the at least one memory and the computer program code configured to, with the at least one processor, cause the system at least to:

identify one or more files that are affected by a proposed change initiated by a user, the proposed change being initiated by a change request, the change request being a merge request comprising data relating to the one or more files and information provided by the

user relating to the proposed change. the information further indicating user details relating to the user, wherein the identification is based on the data;

determine an impact area of the application that is affected by the proposed change based on the identified one or more files and the information; and

merge the merge request with source code of the application based on a check whether the information is in the merge request.

9. The system according to claim 8, further configured to:

identify an impacted business vertical based on the identified 1 or more files, and determining the impact area based on the impacted business vertical.

10. The system according to claim 9, the system further configured to:

generate a report relating to the change request, and

store the report in a database based on the impacted business vertical and the user details.

11. The system according to claim 8, further configured to:

monitor one or more databases for a notification relating to failure of the application to function properly, the notification comprising information indicating details of the failure and indicating a file and an owner of the file that is related to the failure;

determine a severity level of the failure by checking the information with an analytics platform; and

report the notification and determined severity level to one or more users.

12. The system according to claim 11, wherein the monitoring is done on a periodic basis, the system further configured to:

generate, for a period of monitoring, a report documenting any notification relating to failure of the application to function properly found during the period of monitoring.

13. The system according to claim 8, further configured to:

identify one or more devices based on the determined impact area, and

generate a report indicating the identified devices.

14. The system according to claim 13, further comprising:

perform regression testing for the determined impact area for the application on the identified devices, and

generate a report documenting any issues during the regression testing.