(12) UK Patent Application (19) GB (11) 2503486 (13) A

(43) Date of A Publication 01.01.2014

(21) Application No: 1211485.6

(22) Date of Filing: 28.06.2012

(51) INT CL:
G06F 17/22 (2006.01)

(56) Documents Cited:
WO 2008/063797 A2 **US 5047918 A**
US 20110239192 A1 **US 20080250394 A1**

(58) Field of Search:
INT CL **G06F**
Other: **WPI, EPODOC**

(71) Applicant(s):
International Business Machines Corporation
(Incorporated in USA - New York)
New Orchard Road, Armonk, New York 10504,
United States of America

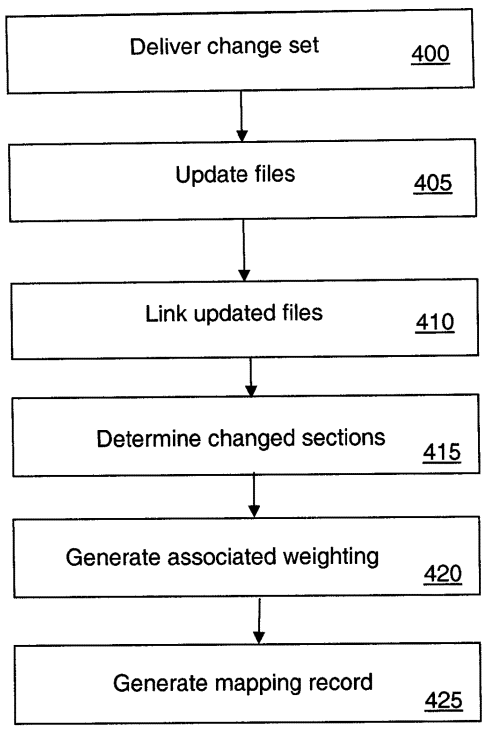
(72) Inventor(s):
Sam Marland
Peter Cullen
Yue Wang
John Duffell

(74) Agent and/or Address for Service:
IBM United Kingdom Limited
Intellectual Property Law, Hursley Park,
WINCHESTER, Hampshire, SO21 2JN,
United Kingdom

(54) Title of the Invention: **An apparatus for managing changes to one or more files**
Abstract Title: **Managing changes to files**

(57) An apparatus for managing changes to one or more files, wherein a link is associated with changes that result in an updated version of the one or more files, the apparatus comprising: a changed section determiner operable to determine which sections associated with the one or more files have changed and operable to determine metadata regarding the nature of the changes to the sections; a metrics generator operable to use the metadata to generate a first weight associated with each of the changed sections; and a mapping record generator operable to generate a mapping record for each of the changed sections, wherein a first mapping record comprises data associated with the first weight; is associated with a particular version of the one or more files; and is linked to mapping records associated with other changed sections using the link.

FIG. 4


Deliver change set 400
Update files 405
Link updated files 410
Determine changed sections 415
Generate associated weighting 420
Generate mapping record 425

GB 2503486 A

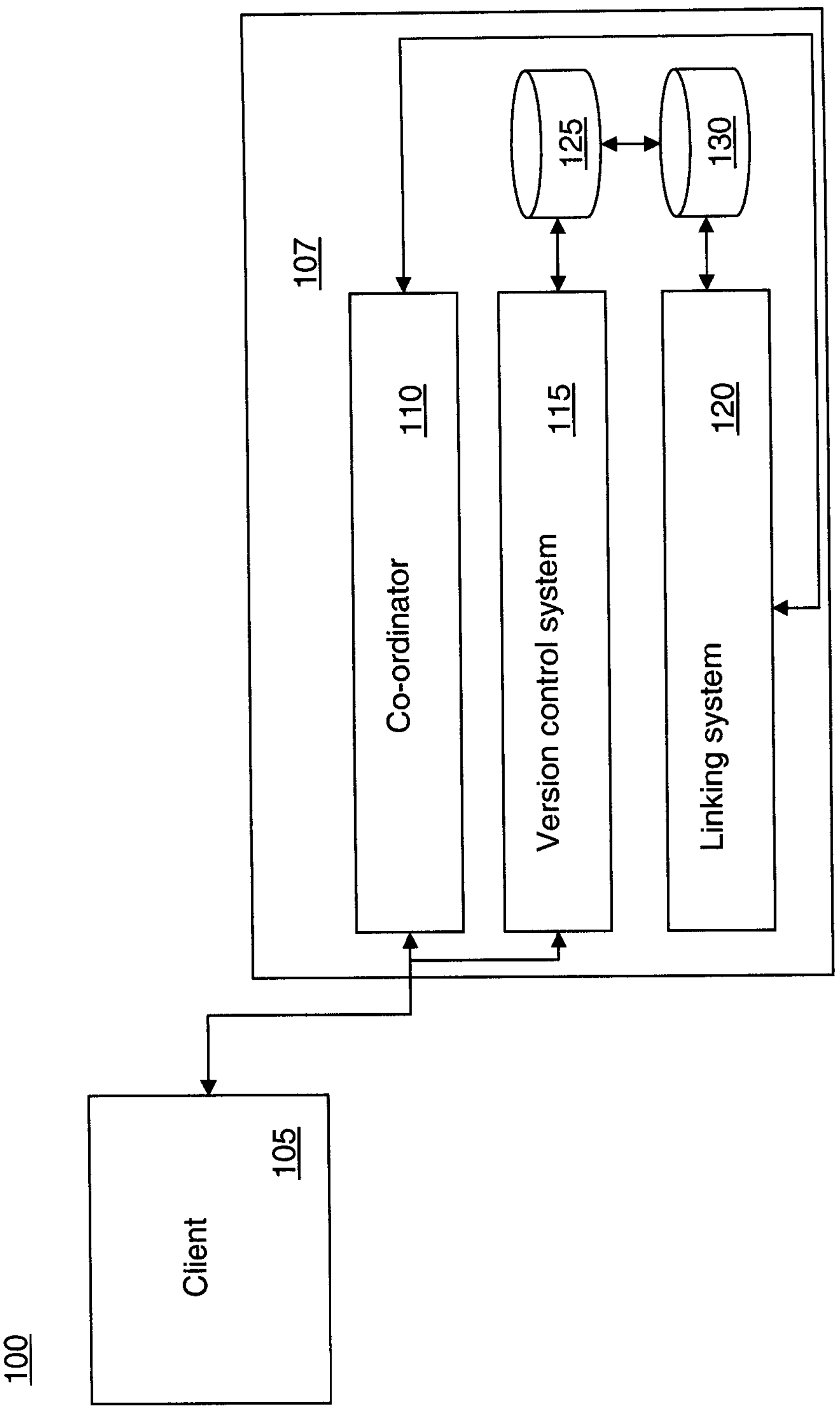

100
Client
105
107
Co-ordinator
110
Version control system
115
Linking system
120
125
130

FIG. 1

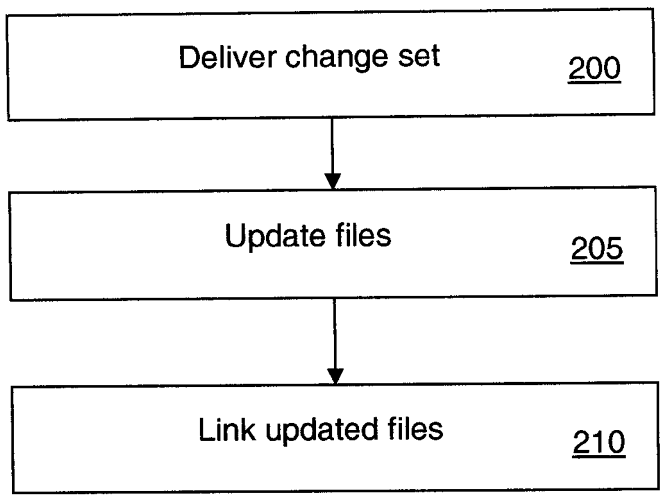

Deliver change set
200
Update files
205
Link updated files
210

FIG. 2

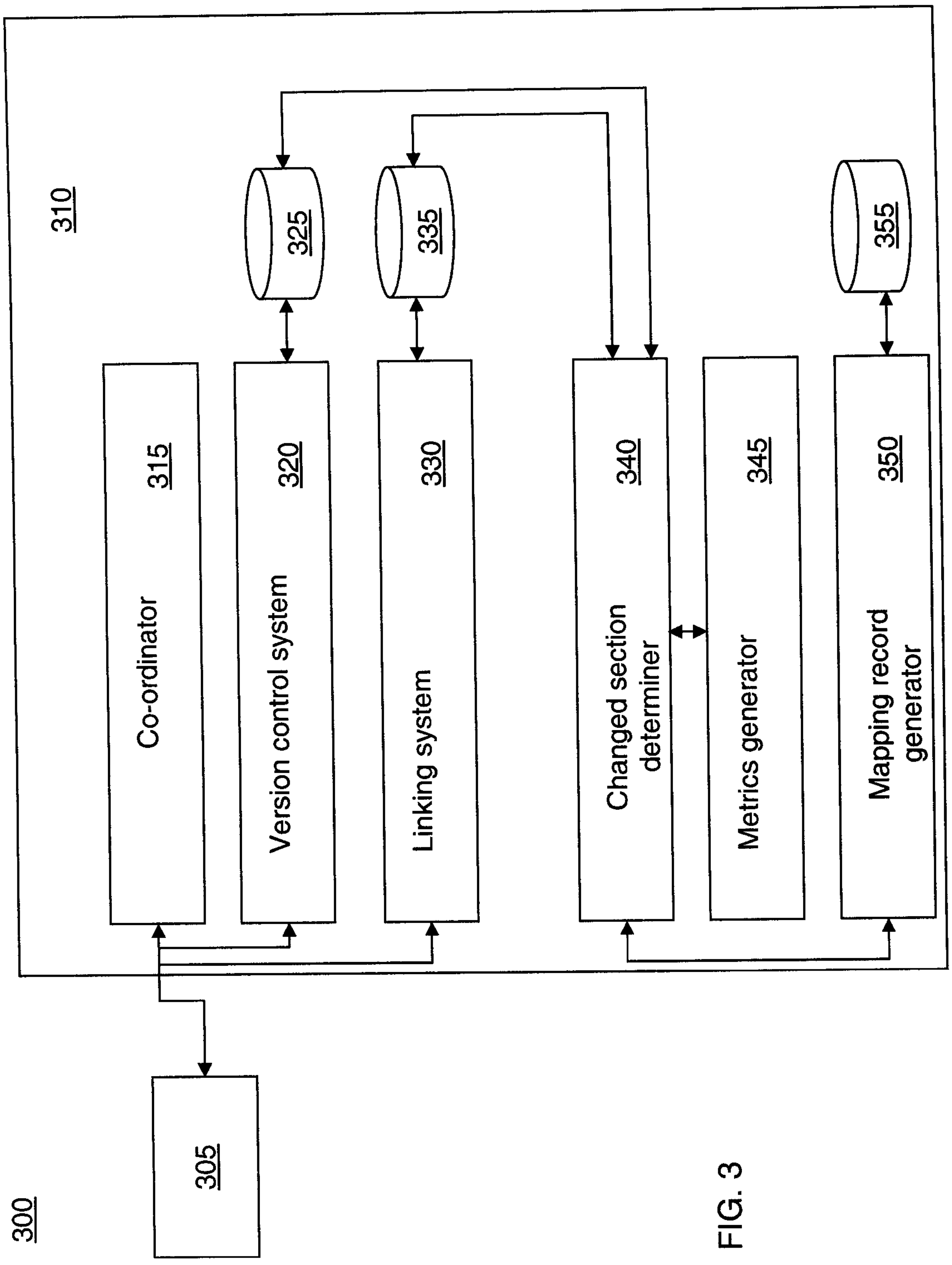
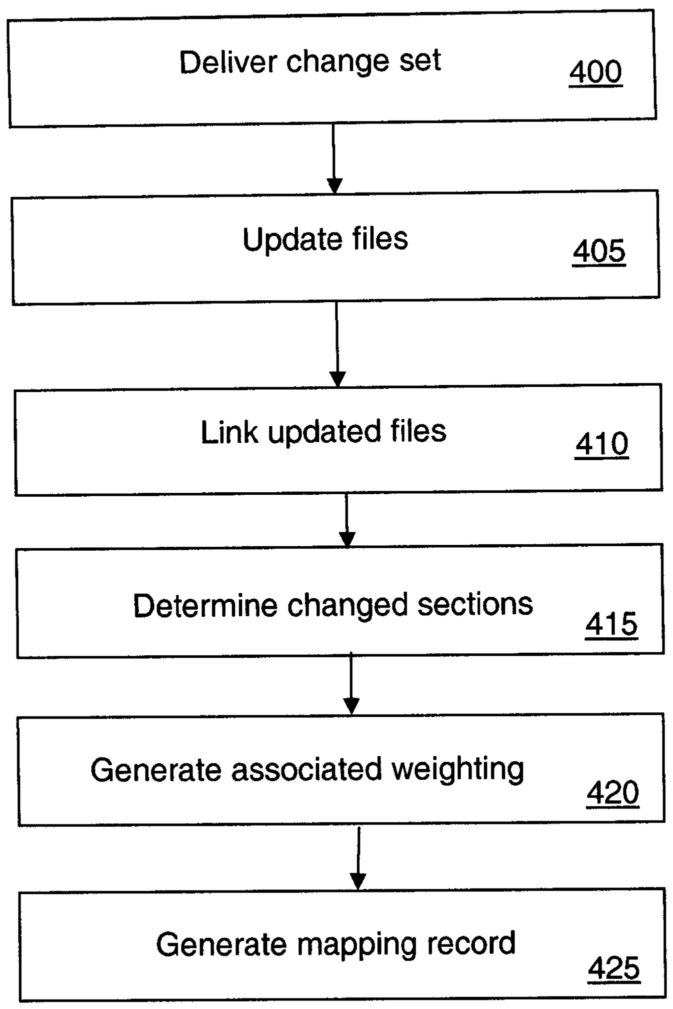

300
305
310
Co-ordinator 315
Version control system 320
325
Linking system 330
335
Changed section determiner 340
Metrics generator 345
Mapping record generator 350
355

FIG. 3



Deliver change set
400
Update files
405
Link updated files
410
Determine changed sections
415
Generate associated weighting
420
Generate mapping record
425

FIG. 4

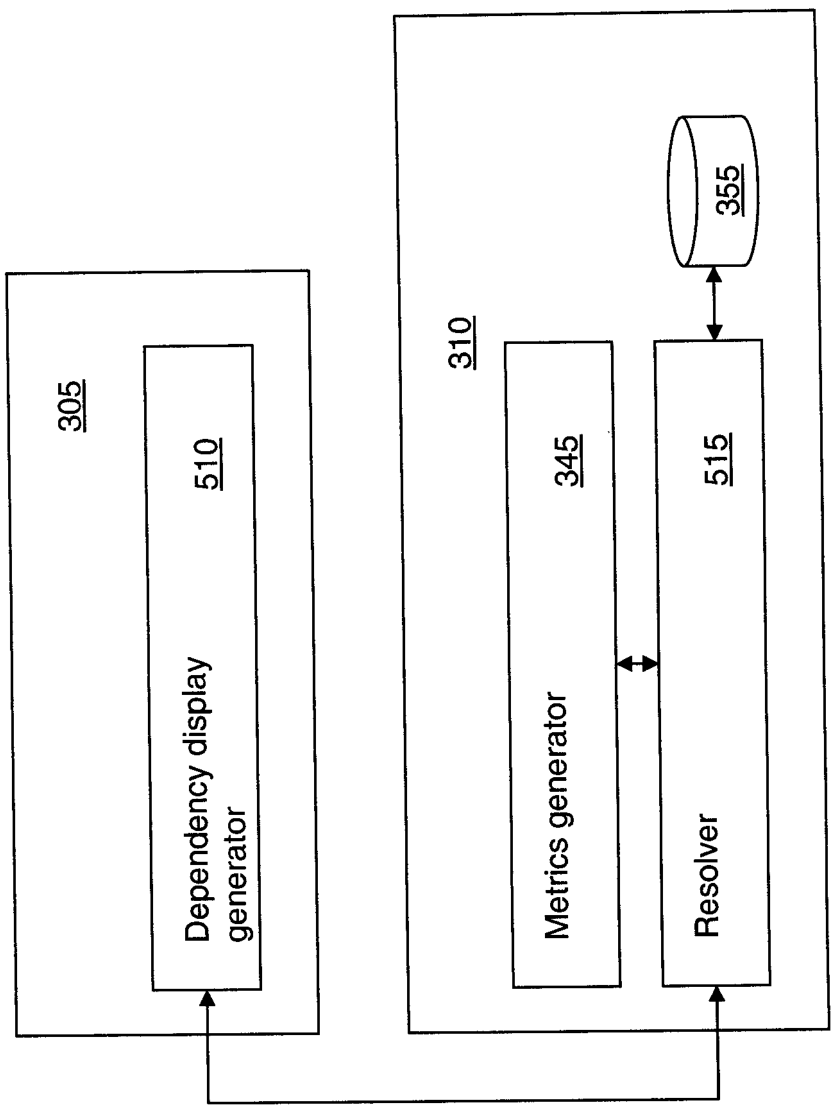
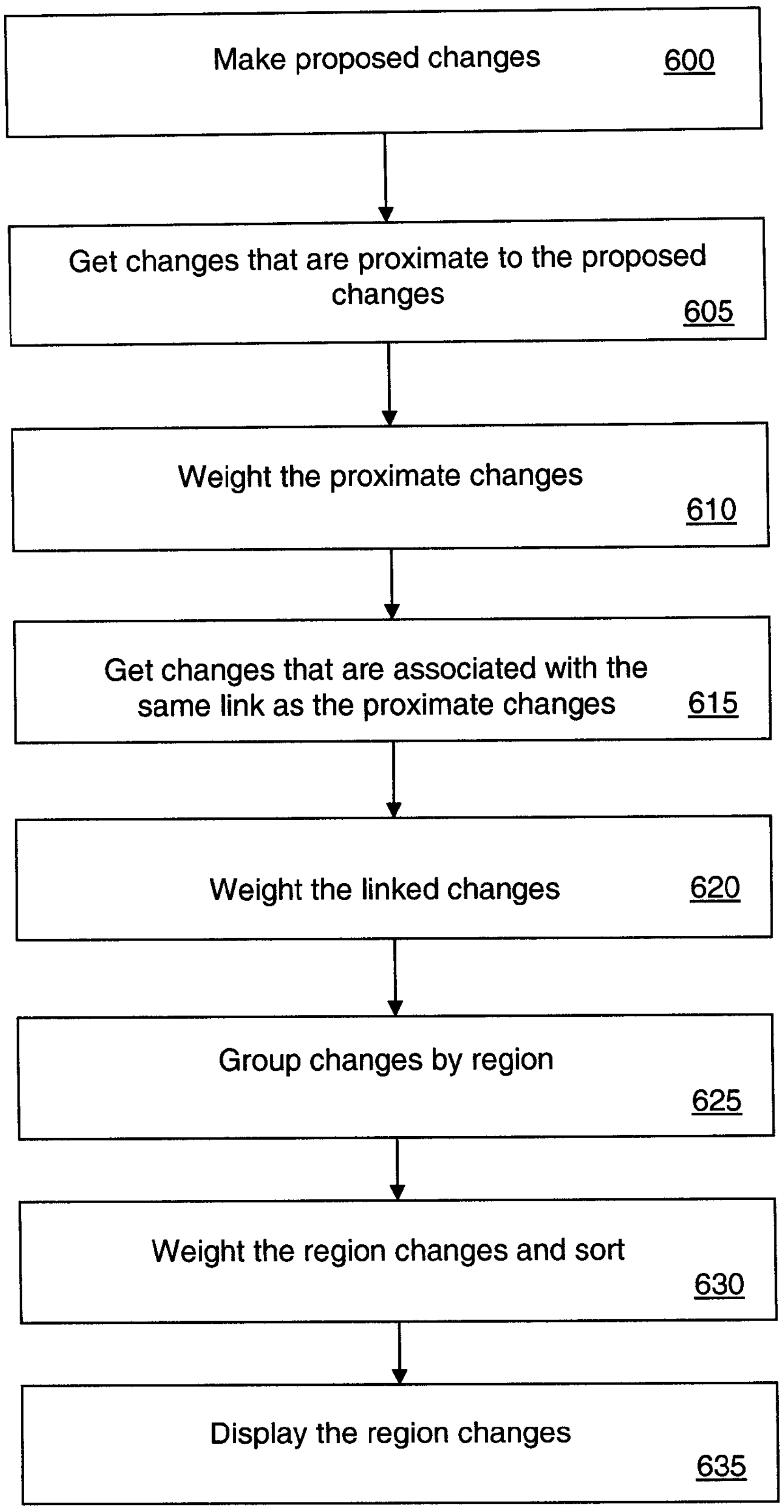
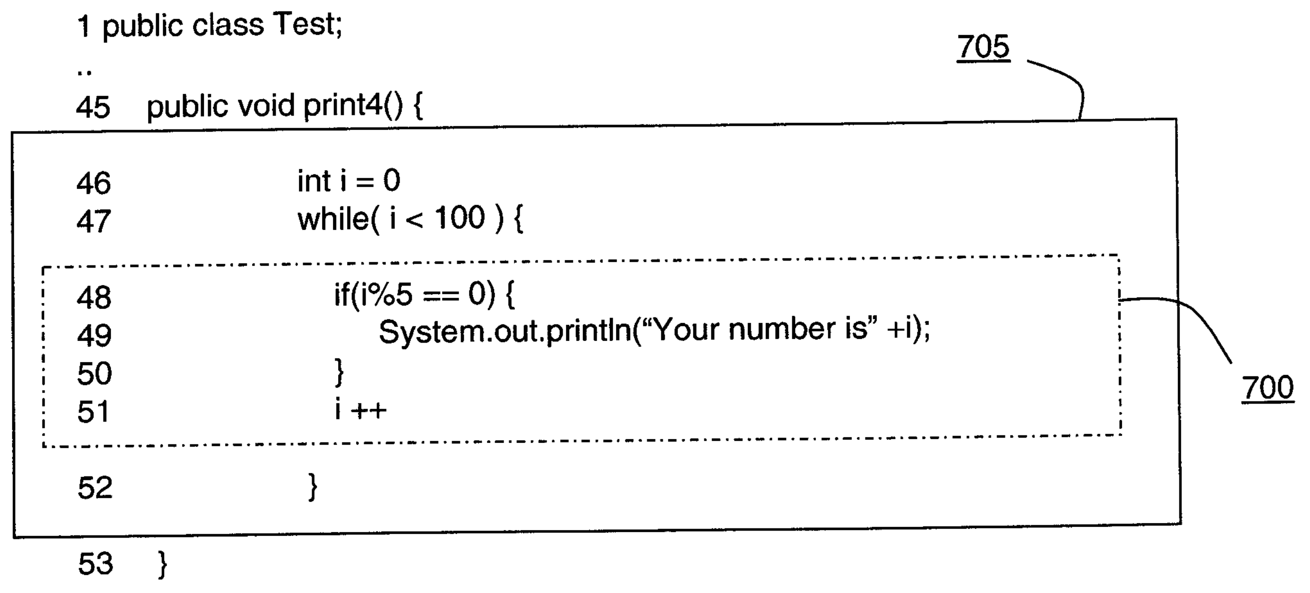

305
Dependency display generator
510
310
Metrics generator
345
Resolver
515
355

FIG. 5



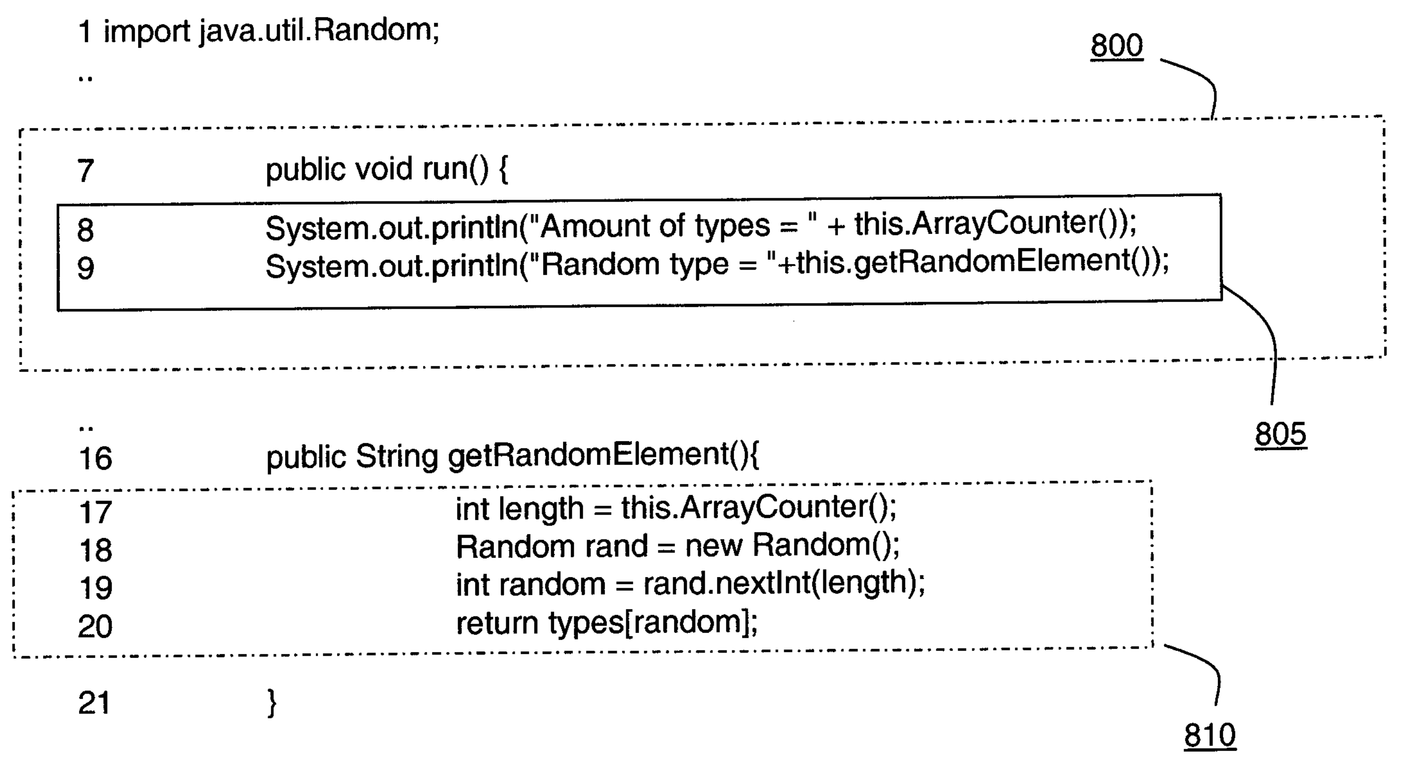
Make proposed changes 600
Get changes that are proximate to the proposed changes 605
Weight the proximate changes 610
Get changes that are associated with the same link as the proximate changes 615
Weight the linked changes 620
Group changes by region 625
Weight the region changes and sort 630
Display the region changes 635

FIG. 6



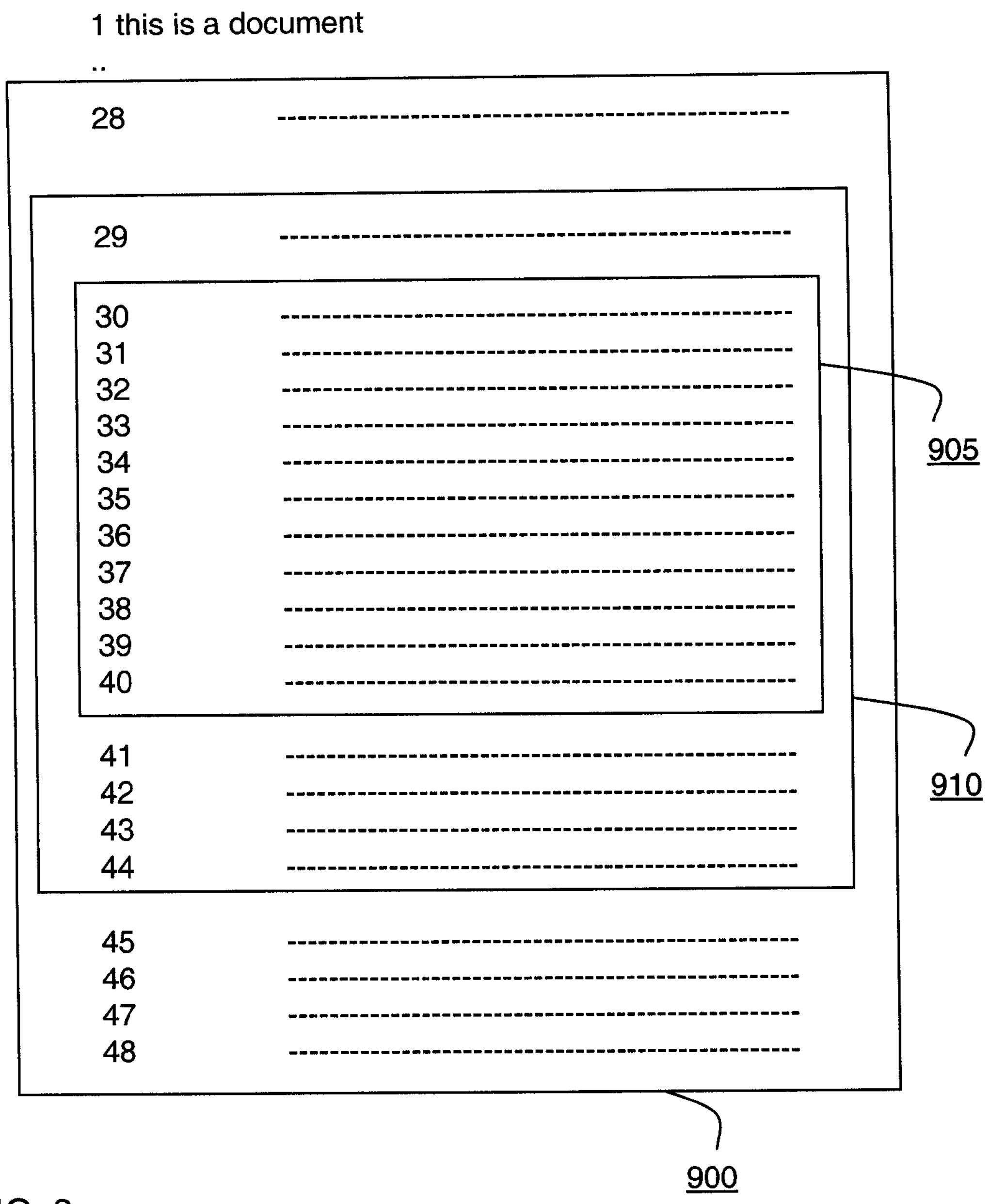
```
1 public class Test;
..
45   public void print4() {
46              int i = 0
47              while( i < 100 ) {
48                   if(i%5 == 0) {
49                        System.out.println("Your number is" +i);
50                   }
51                   i ++
52              }
53   }
```

FIG. 7



```
1 import java.util.Random;
..
7               public void run() {
8               System.out.println("Amount of types = " + this.ArrayCounter());
9               System.out.println("Random type = "+this.getRandomElement());
..
16              public String getRandomElement(){
17                              int length = this.ArrayCounter();
18                              Random rand = new Random();
19                              int random = rand.nextInt(length);
20                              return types[random];
21              }
```

FIG. 8



1 this is a document
..
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
905
910
900

FIG. 9

AN APPARATUS FOR MANAGING CHANGES TO ONE OR MORE FILES

FIELD OF THE INVENTION

The present invention relates to an apparatus for managing changes to one or more files.

BACKGROUND OF THE INVENTION

Currently, it is difficult to identify which parts of a file may need to be changed when an associated file is changed. For example, it is difficult to identify which parts of documentation for a product require updating when associated code is changed or vice versa. This can result in, for example, the documentation being out of date (e.g., because the documentation has not been updated to reflect changes in the code) or a scenario wherein documentation associated with the code is present in multiple locations and is not consistently updated in each of the locations.

A prior art solution is shown in figure 1, comprising a system (100) having a client computer (105) operable to interact with a server computer (107). The server computer (107) comprises a co-ordinator (110); a version control system (115) having access to a first data store (125) for storing one or more files – in the example herein, the files comprise code and associated documentation; and a linking system (120) having access to a second data store (130) for storing one or more links.

A process according to the prior art will now be described with reference to figure 2. At step 200, a user uses the client computer (105) to make changes to code and to documentation and “commits” the changes to a change management system (e.g., depicted by the server computer (107)).

The co-ordinator (110) receives the changes from the client computer (105) and co-ordinates the version control system (115) to access the files in the first data store (125) and to update (step 205) the relevant files. The co-ordinator (110) co-ordinates the linking system (120) to create (step 210) a link. In the example herein: links are created between one or more code files and one or more documentation files. The co-ordinator (110) co-ordinates the linking system (120) to create a link between the code file(s) that was changed by the user and the documentation file(s) that was changed by the user.

The linking system (120) links the files by use of a timestamp, wherein a code file and a documentation file are linked if they are committed at the same time.

Alternatively, the linking system (120) links the files if changes associated with the file are committed under the same change item (which has an identifier) wherein a change item can be used to group changes associated with a logical unit of work.

The linking system (120) stores the link in the second data store (130).

This solution is disadvantageous in that it is coarse grained – e.g., links can only be created based on an exact timestamp or a change item and further, it is not possible to easily identify which sections of the documentation relate to which sections of code.

DISCLOSURE OF THE INVENTION

According to a first aspect, there is provided an apparatus for managing changes to one or more files, wherein a link is associated with changes that result in an updated version of the one or more files, the apparatus comprising: a changed section determiner operable to determine which sections associated with the one or more files have changed and operable to determine metadata regarding the nature of the changes to the sections; a metrics generator operable to use the metadata to generate a first weight associated with each of the changed sections; and a mapping record generator operable to generate a mapping record for each of the changed sections, wherein a first mapping record comprises data associated with the first weight; is associated with a particular version of the one or more files; and is linked to mapping records associated with other changed sections using the link.

According to a second aspect, there is provided a method for managing changes to one or more files, wherein a link is associated with changes that result in an updated version of the one or more files, the method comprising the steps of: determining which sections associated with the one or more files have changed; determining metadata regarding the nature of the changes to the sections; using the metadata to generate a first weight associated with each of the changed sections; and generating a mapping record for each of the changed sections, wherein a first mapping record comprises data associated with the first weight; is associated with a particular version of the one or more files; and is linked to mapping records associated with other changed sections using the link.

According to a third aspect, there is provided a computer program comprising program code means adapted to perform all the steps of the method above when said program is run on a computer.

Advantageously, by creating and maintaining a link between e.g., code file(s) and documentation file(s), a dependency is created such that when planning any code changes, sections that require updating in the documentation can easily be found and vice versa. This allows a number of further advantages, e.g., an improvement in the quality of documentation created; reduction in cost of updating documentation due to the relevant documentation requiring changes being identified automatically and without the need to rely upon a specialist with experience in the particular area; and an improved ability to plan product changes and determine the cost of such changes.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will now be described, by way of example only, with reference to preferred embodiments thereof, as illustrated in the following drawings:

Figure 1 depicts a system according to the prior art;

Figure 2 is a flow chart showing the operational steps involved in a process according to the prior art;

Figure 3 depicts a system for generating and maintaining links according to the preferred embodiment;

Figure 4 is a flow chart showing the operational steps involved in a process for generating and maintaining links according to the preferred embodiment;

Figure 5 depicts a system for analysing links according to the preferred embodiment;

Figure 6 is a flow chart showing the operational steps involved in a process for analysing links according to the preferred embodiment; and

Figures 7-9 depict a number of changes and records.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

As will be appreciated by one skilled in the art, aspects of the present invention may be embodied as a system, method or computer program product. Accordingly, aspects of the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a "circuit," "module" or "system." Furthermore, aspects of the present invention may take the form of a computer program product embodied in one or more computer readable medium(s) having computer readable program code embodied thereon.

Any combination of one or more computer readable medium(s) may be utilized. The computer readable medium may be a computer readable signal medium or a computer readable storage medium. A computer readable storage medium may be, for example, but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, or device, or any suitable combination of the foregoing. More specific examples (a non-exhaustive list) of the computer readable storage medium would include the following: an electrical connection having one or more wires, a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fibre, a portable compact disc read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a computer readable storage medium may be any tangible medium that can contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device.

A computer readable signal medium may include a propagated data signal with computer readable program code embodied therein, for example, in baseband or as part of a carrier wave. Such a propagated signal may take any of a variety of forms, including, but not limited to, electro-magnetic, optical, or any suitable combination thereof. A computer readable signal medium may be any computer readable medium that is not a computer readable storage medium and that can communicate, propagate, or transport a program for use by or in connection with an instruction execution system, apparatus, or device.

Program code embodied on a computer readable medium may be transmitted using any appropriate medium, including but not limited to wireless, wireline, optical fibre cable, RF, etc., or any suitable combination of the foregoing.

Computer program code for carrying out operations for aspects of the present invention may be written in any combination of one or more programming languages, including an object oriented programming language such as Java (Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates), Smalltalk, C++ or the like and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider).

Aspects of the present invention are described below with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer readable medium that can direct a computer, other programmable data processing apparatus, or other devices to function in a particular manner, such that the instructions stored in the computer readable medium produce an article of manufacture including instructions which implement the function/act specified in the flowchart and/or block diagram block or blocks.

The computer program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other devices to cause a series of operational steps to be performed on the computer, other programmable apparatus or other devices to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

A preferred embodiment will now be described with reference to the figures.

A system (300) for generating links is shown in figure 3 and comprises a client computer (305) operable to interact with a server computer (310). The server computer (310) comprises a co-ordinator (315); a version control system (320) having access to a third data store (325) for storing one or more files – in the example herein, the files comprise code and associated documentation but it should be understood that the files can comprise any number of artefacts (e.g., code only; documentation only); and a linking system (330) having access to a fourth data store (335) for storing one or more links.

The server computer (310) also comprises a changed section determiner (340) operable to access the third data store (325); the fourth data store (335); a metrics generator (345) and a mapping record generator (350). The mapping record generator (350) is operable to access a fifth data store (355) for storing one or more mapping records.

A process for generating links according to the preferred embodiment will now be described with reference to figure 4.

A user uses the client computer (305) to make changes to at least one of: code and documentation. In the example herein, the changes are associated with a change item which has an identifier e.g., a work item that is associated with a changed function in the code (e.g., a work item 6000 is associated with adding a new function to enable secure access to a web site; a work item 5000 is associated with adding a new method to an interface) or an entry in a change set which is typically smaller in scale than a work item. A work item may comprise a plurality of change sets. In the example herein, the change item comprises a work item.

At step 400, in the example herein, the user uses the client computer (305) to send data comprising a changed code file (having an identifier); a changed documentation file (having an identifier) (alternatively, file identifiers associated with the changed files; and the changed information and/or associated offset data) and an associated work item identifier to the server computer (310).

In the example herein, the co-ordinator (315) receives the data and passes the data to the version control system (320).

The version control system (320) uses file identifiers associated with the changed code file and the changed documentation file in order to access the files in the third data store (325) and find server-held versions of the files.

The version control system (320) compares the changed code file and the changed documentation file with the server-held versions in order to determine the changes that the user has made (e.g., using a diff tool).

Thereafter, at step 405, the version control system (320) updates the server-held versions in the third data store (325) and stores the changed files as new versions. Preferably, the original server-held versions of the files are also maintained. It should be understood that determining the changes and updating the files can be carried out in a number of ways.

At step 410, the co-ordinator (315) invokes the linking system (330) to create a link between the code file that was changed by the user and the documentation file that was changed by the user under the work item. Note that a link is associated with changes that result in updated versions of associated files.

The linking system (330) stores the link (e.g., using a link identifier) in the fourth data store (335).

At step 415, the changed section determiner (340) is operable to access the third data store (325) and the fourth data store (335) in order to determine which sections of the code file(s) and the documentation file(s) were changed by the user. The changed section determiner (340) is also operable to determine further data regarding the changes (e.g., size, complexity etc.).

At step 420, the metrics generator (345) generates weightings associated with the link between the code file and the documentation file that have been changed under the work item – in more detail, the metrics generator (345) generates a weighting associated with each changed section of the code file(s) and the documentation file(s). The metrics generator (345) communicates with the changed section determiner (340) in order to obtain further data associated with the code changes (e.g., size; complexity; code coverage analysis) and to obtain further data associated with the documentation changes (e.g., size; complexity; nature of changes (e.g., change in tense; change in values)). The metrics generator (345) uses the further data in order to create the weighting.

At step 425, the mapping record generator (350) is operable to generate a set of mapping records associated with the link -- the set of mapping records associated with the link comprises the same identifier. The mapping record generator (350) generates a mapping record for each changed section comprising a weighting associated with the changed section; an identifier and a time stamp. The mapping record generator (350) stores mapping records in the fifth data store (355).

Advantageously, a set of mapping records associate a section(s) of code that is changed and a section(s) of documentation that is changed under the same work item, wherein the changes results in a new version of associated files.

A worked example of figure 4 will now be described.

At step 400, in the example herein, the user uses the client computer (305) to send data comprising a changed code file; a changed documentation file and an associated work item identifier (e.g., 4000) to the server computer (310).

Example changes to the code file and the documentation file are shown below:

Changes_1

Code file version (ID 1234) – line 3-10
Documentation file version (ID 5678) – line 60-80

The co-ordinator (315) receives the data and passes the data to the version control system (320).

The version control system (320) uses file identifiers (e.g., “1234” and “5678”) in order to find server-held versions of the files and determines that a section comprising lines 3-10 have changed in the code file and that a section comprising lines 60-80 have changed in the documentation file. It should be understood that changes may not necessarily be made to every line of a section (e.g., if a user has made changes to lines 3 to 5 and lines 8-10 of a file, lines 3-10 may be treated as a single changed section).

Thereafter, at step 405, the version control system (320) updates the server-held versions in the third data store (325). In the example herein, the original versions (termed herein, version 1) of the files and the updated versions (termed herein, version 2) of the files are stored in the third data store (325).

At step 410, the co-ordinator (315) invokes the linking system (330) to create a link between the code file that was changed by the user and the documentation file that was changed by the user under the work item. Note that a link is associated with changes that result in an updated version of a file.

The linking system (330) stores the link (e.g., using a link identifier) in the fourth data store (335). An example of a link is shown below:

Link_1

Link ID = 4321 (Code file version = 1234; Documentation file version = 5678; Work_item = 4000)

At step 415, the changed section determiner (340) accesses the third data store (325) and the fourth data store (335) and determines that a section comprising lines 3-10 have changed in the code file and that a section comprising lines 60-80 have changed in the documentation file.

The changed section determiner (340) also determines further data regarding the changes.

In one example, further data comprising a size of the change in relation to a total size of a file is determined (e.g., by dividing the number of lines changed by the total number of lines in the file).

In another example, further data comprising complexity of a change is estimated. In one example, a programming language associated with a code file is identified and thereafter, e.g., for complied languages such as Java (Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates, a compiler is executed in order to obtain a parse tree for an updated version of the code file and a preceding version of the code file. Nodes of each parse tree are iterated in order to determine differences – e.g., a number of nodes in the new tree that are not identical to ones in the previous tree could be counted. In another example, a cyclomatic complexity of each of an updated version of the code file and a preceding version of the code file is calculated.

At step 420, the metrics generator (345) generates weightings associated with the link (Link_1) between the code file and the documentation file that have been changed under the work item. The metrics generator (345) communicates with the changed section determiner (340) in order to further obtain data associated with the code changes and further data associated with the documentation changes in order to create a weighting. It should be understood that any amount of further data regarding the changes can be taken into account in order to generate a weight and any number of weighting schemes can be used.

In the example herein, weights for each changed section associated with the link are generated using further data associated with size as shown below, wherein in the example, a greater weight (e.g., 20) is given to a change having a larger size:

Weights_1

Code file version (ID 1234) – java line 3-10; weight = 7
Documentation file version (ID 5678) – java line 60-80; weight = 20

The weight is a relative measure of how important a changed section is – e.g., if a changed section has a higher weight, a user would want to be alerted to it in preference to a changed section having a lower weight.

At step 425, the mapping record generator (350) is operable to generate one or more mapping records associated with the link – examples are shown below:

Mapping record ID = 123
Code file version ID = 1234
Region start offset = 3
Region end offset = 10

Timestamp = 12/4/12 07:00

Weight = 7

Mapping record ID = 123

Documentation file version ID = 5678

Region start offset = 60

Region end offset = 80

Timestamp = 12/4/12 07:05

Weight = 20

Advantageously, a set of mapping records is associated with a changed section(s) to code and a changed section(s) to documentation associated with the same work item and associated with a link to a particular version of associated files.

It should be understood that if changes associated with a link are received and if such changes have associated lines that overlap (wherein the overlap can have a preferred size) , such changes can be merged for the purposes of detailing in a mapping record. Alternatively, mapping records can be created for each of the overlapping changes.

In the example herein, there are further changes (step 400) associated with the same work item (ID= 4000).

Example changes to the code file and the documentation file are shown below – note that the file versions have different identifiers from those denoted in Changes_1 as the further changes being made result in a different version of the files:

Changes_2

Code file version (ID 123456) – line 3-8

Documentation file version (ID 5678910) - line 20-23

As above, at step 405, the version control system (320) updates the server-held versions in the third data store (325). In the example herein, version 2 and the updated versions (termed herein, version 3) of the files are stored in the third data store (325). Further, version 1 is also maintained.

At step 410, the co-ordinator (315) invokes the linking system (330) to create a second link which is stored by the linking system (330) in the fourth data store (335). An example of a link is shown below – note that as Changes_2 results in updated versions of the associated files, Link_2 has a different Link ID from those denoted in Link_1:

Link_2

Link ID = 4444 (Code file version = 123456; Documentation file version = 5678910; Work_item = 4000)

At step 415, the changed section determiner (340) accesses the third data store (325) and the fourth data store (335) and determines that lines 3-8 have changed in the code file and lines 20-23 have changed in the documentation file. The changed section determiner (340) also determines further data regarding size of the changes.

At step 420, the metrics generator (345) generates weightings associated with the link (Link_2) by using the further data regarding the changes determined by the changed section determiner (340).

In the example herein, weights for each changed section associated with the link are generated using further data associated with size as shown below, wherein in the example, a greater weight (e.g., 5) is given to a change having a larger size:

Weights_2

Code file version (ID 123456) – line 3-8; weight = 5
Documentation file version (ID 5678910) – line 20-23; weight = 3

At step 425, the mapping record generator (350) is operable to generate one or more mapping records associated with the link – note that the ID (e.g., 1234) of the mapping records associated with Changes_2 is different from the ID (e.g., 123) of the mapping records associated with Changes_1 to reflect the different resulting versions of the associated files – examples are shown below:

Mapping record ID = 1234
Code file ID = 123456
Region start offset = 3
Region end offset = 8
Timestamp = 12/4/12 09:00
Weight = 5

Mapping record ID = 1234
Documentation file = ID 5678910
Region start offset = 20
Region end offset = 23
Timestamp = 12/4/12 09:05
Weight = 3

From the above examples, it can be seen that when changes are made to a code file and to a documentation file and result in updated versions of the files, such changes can be linked under a set of mapping records. This allows a user to have an overview of the impact on one file of changes made to another file. Further, each changed section is advantageously weighted such that a user can understand the relative importance/significance of the changed section on each of the impacted sections and focus his/her attention appropriately, thus ignoring a majority of irrelevant results – further, such weightings can be used in another process of the preferred embodiment for analysing links as will be detailed below.

In an example, when a user activates a preferred implementation in a code file, the preferred implementation generates a call hierarchy using e.g., static analysis, code coverage to scan through each of the changes to the code file found, checking for links in e.g., a documentation file. The preferred implementation renders a tree for the hierarchy in conjunction with the further data regarding the changes and the tree is augmented with the documentation file as a child node of a parent node representing the code file. Preferably, if the user activates the child node, the documentation file is opened in an appropriate editor.

A system and process for analysing links will now be described.

It should be understood that when changing e.g., code, a developer can carry out manual impact analysis on the code in order to find out which other artefacts may be affected by the code changes. If another artefact, e.g., associated documentation is required to be changed, a developer may not necessarily remember to update the documents or indeed know which parts of the documentation require updating.

Advantageously, the system and process for analysing links according to the preferred embodiment aids with the problem above in that, in response to making a change to a file, a user is able to review sections of the file and/or sections of other files which may require updating also.

With reference to figure 5, a more detailed diagram of the client computer (305) and the server computer (310) is shown. The client computer (305) comprises a dependency display generator (510) – the further functions of the client computer (305) depicted in figure 3 have been omitted for clarity. The server computer (310) comprises the metrics generator (345) and a resolver (515) operable to access the fifth data store (355) for storing one or more mapping records – the further functions of the server computer (310) depicted in figure 3 have been omitted for clarity.

A process for analysing links according to the preferred embodiment will now be described with reference to figure 6.

A user uses the client computer (305) to make (step 600) changes (termed herein, proposed changes) to code only and the user sends data comprising the changes and the associated work item identifier to the server computer (310).

Example changes to the code file are shown below:

Changes_3

(1) Code file version (ID trunk/Test/src/com/test/Test.java) – line 48-51
(2) Code file version (ID trunk/Test/src/com/test/Types.java) – line 7-9

(3) Code file version (ID trunk/Test/src/com/test/Types.java) – line 17-20

The first proposed change (700) is shown in figure 7; the second proposed change (800) is shown in figure 8; and the third proposed change (810) is shown in figure 8.

The version control system (320) updates the server-held versions in the third data store (325); the co-ordinator (315) invokes the linking system (330) to create a link between the changed sections of the code file (further details of the link are omitted herein for clarity) which is stored by the linking system (330) in the fourth data store (335). The changed section determiner (340) accesses the third data store (325) and the fourth data store (335) in order to determine which sections have changes and determine further data regarding the changes.

The metrics generator (345) optionally generates weightings associated with the link by using the further data regarding the changes determined by the changed section determiner (340) and the mapping record generator (350) is operable to generate mapping records associated with the link. If weightings are not generated, a default value of one (1) is used in the algorithms that follow.

Examples of the mapping records are shown below wherein a greater weight is given to more significant (e.g., not in terms of size, but complexity, user configured data etc.) changes:

Proposed change mapping record 1
Mapping record ID = ABC
Code file ID = trunk/Test/src/com/test/Test.java
Region start offset = 48
Region end offset = 51
Timestamp = 20/4/12 09:00
Weight = 30

Proposed change mapping record 2
Mapping record ID = ABC
Code file ID = trunk/Test/src/com/test/Types.java
Region start offset = 7

Region end offset = 9

Timestamp = 20/4/12 10:00

Weight = 1

Proposed Change mapping record 1

Mapping record ID = ABC

Code file ID = trunk/Test/src/com/test/Types.java

Region start offset = 17

Region end offset = 20

Timestamp = 20/4/12 11:30

Weight = 5

Alternatively, the user e.g.., selects areas of the code to which he/she wishes to make proposed changes and the mapping record generator (350) uses the selections to generate dummy proposed change mapping records each having a default weight of 1.

Alternatively, the user e.g., calls the preferred embodiment by activating an element associated with a code file by selecting a method from a call hierarchy and requesting a list of callers associated with the selected method. In addition to returning the callers, the mapping record generator (350) generates one or dummy proposed change mapping records each having a default weight of 1 associated with an area of code associated with the element.

At step 605, the resolver (515) accesses the fifth data store (355) and runs one or more queries in order to retrieve mapping records associated with previously made changes that are proximate to the proposed changes.

In one example, the resolver (515) runs one or more queries to obtain mapping records associated with changes that are located in a section that is within 50% of the size of the proposed change. For example, a proposed change from lines 20-30 would match proximate changes comprising lines 15-35 and lines 25-26 but not lines 20-22.

In a first query associated with the first proposed change comprising "select all mapping records where file id= trunk/Test/src/com/test/Test.java; start offset is in the range 46-50; end offset is in the range 49-55", the following mapping record is returned:

Proximate change mapping record 1
Mapping record ID = XYZ
Code file ID = trunk/Test/src/com/test/Test.java
Region start offset = 46
Region end offset = 52
Timestamp = 18/4/12 10:00
Weight = 43

In a second query associated with the second proposed change comprising "select all mapping records where file id= trunk/Test/src/com/test/Types.java; start offset is in the range 5-9; end offset is in the range 7-11", the following mapping record is returned:

Proximate change mapping record 2
Mapping record ID = XYZ
Code file ID = trunk/Test/src/com/test/Types.java
Region start offset = 8
Region end offset = 9
Timestamp = 17/4/12 11:30
Weight = 13

The fifth data store (355) also comprises the following record having the same mapping record ID as the two mapping records above:

Change mapping record 1
Mapping record ID = XYZ
Document file ID = file id: trunk/Docs/src/Installation.xml
Region start offset = 28
Region end offset = 48
Timestamp = 17/4/12 12:00
Weight = 34

Proximate change mapping record 1 (700) is shown in figure 7; Proximate change mapping record 2 (805) is shown in figure 8; and Change mapping record 1 (900) is shown in figure 9.

In a third query comprising "select all mapping records where file id= trunk/Test/src/com/test/Types.java; start offset is in the range 15-19; end offset is in the range 18-22", no mapping records (including Change mapping record 1) are returned.

At step 610, the metrics generator (345) assigns a mapping record associated with a proximate change a context specific weight which represents the significance of the mapping record in comparison with a proposed change. A context specific weight is based on for example:

- a weight of a proposed change – if a proposed change to a section is significant, by using a weight of the proposed change, significance can also be placed on proximate (previous) changes that have been made to the section since a weight of a proximate change is also a factor in determining a context specific weight – see below
- a weight of a proximate change - if a proximate change to a section is significant, by using a weight of the proximate change, significance can also be placed on a proposed change that has been made to the section since a weight of a proposed change is also a factor in determining a context specific weight
- overlap between a section associated with the proposed change and a section associated with the proximate change – this results in a mapping record associated with a proximate change having a more significant overlap with a proposed change to be treated as more significant.

In an example, a context specific weight is obtained by summing a weight of a proposed change and a weight of a proximate change and multiplying the sum by a percentage overlap between the changes as a proportion of a total number of lines associated with the changes:

context specific weight = proposed weight + returned weight * (overlapSize/(sum(sizes))

For example, for the first proposed change and the associated first proximate change, a context specific weight for the mapping record of the first proximate change is calculated as follows:

Proximate change mapping record 1 context specific weight:

30 + 43 * (4 / (4+7)) = 26.55

For the second proposed change and the associated second proximate change, a context specific weight for the mapping record of the second proximate change is calculated as follows:

Proximate change mapping record 2 context specific weight:

1 + 13 * (2 / 3+2) = 5.6

Context specific weights can be used to decide which proximate changes to treat as the most important ones.

Preferably, if multiple proximate changes are returned, proximate changes with relatively low context specific weights can be filtered out. Such low context specific weights can indicate a relatively low weight of a proposed change; a weight of a proximate change and/or percentage overlap.

At step 615, the resolver (515) executes a query in order to obtain each mapping record associated with the same mapping ID as the returned proximate change mapping records. This allows mapping records associated with the same link as that associated with the mapping records of the proximate changes to be determined such that a user has an expanded overview of changes that may be related to the proposed changes (wherein such changes may not necessarily be proximate to the proposed changes but are related to the proximate changes). For example, the resolver (515) issues a query comprising “SELECT all where mapping ID = XYZ” resulting in a return of: Proximate change mapping record 1; Proximate change mapping record 2 and Change mapping record 1.

At step 620, the metrics generator (345) weights one or more mapping records associated with the link with an importance weight which represents the significance of the one or mapping records with the link in comparison with a proposed change.

An importance weight is calculated using a multiplication of a weight of a mapping record associated with the link and the context specific weight – using the weight associated with the mapping record associated with the link allows importance to be placed on changes that are more significant over changes which are more trivial.

Note that as weights of proximate changes have already been used as a factor in deriving a context specific weight, more significance is placed on such proximate changes over changes that are associated with the proximate changes under the same link.

The importance weight is used to carry through data about which changes associated with the link are more significant such that relevant results can be displayed at a later stage to the user (this step will be discussed in more detail herein).

An importance weight is calculated for each mapping record associated with the link against each context specific weight -- this is because proximate changes associated with the same mapping record ID represents a set of related changes and significance should be given to this.

Examples of importance weights are shown below:

Proximate change mapping record 1 importance weights:

43 * 26.55 = 1141.7
43 * 5.6 = 240.8

Proximate change mapping record 2 importance weights:

34 * 26.55 = 902.7
34 * 5.6 = 190.4

Change mapping record 1 importance weights:

13 * 26.55 = 345.2
13 * 5.6 = 72.8

It should be understood that in the example above, based on the proposed changes, proximate changes committed under the same work item have been retrieved. Equally, proximate changes committed under different work items can be retrieved and the above process would also be applied to such changes.

At step 625, the resolver (515) generates one or more region records. In the example herein, a region record comprises an aggregation of proximate changes and further changes wherein such changes overlap. In an example, the resolver (515) generates the one or more region records in accordance with one or more rules (e.g., wherein a rule defines that if a start offset and an end offset of a changed section is within 50% proximity of another changed section, associated mapping records will be merged into a single region record).

The creation of one or more region records allows for a simplification of display of results as if such changes were each displayed separately, this can result in several results associated with the same or a similar section causing problems for the user e.g., making the results difficult for the user to interpret; the most significant results may not be readily apparent.

In the example herein, as Proximate change mapping record 1, Proximate change mapping record 2 and Change mapping record 1 are associated with different files (note the different file IDs), no overlap exists and thus, the resolver (515) generates a region record associated with each of the mapping records wherein:

Region record 1 is associated with Proximate change mapping record 1
Region record 2 is associated with Proximate change mapping record 2
Region record 3 is associated with Change mapping record 1

In another example, based on a proposed change, a proximate change associated with a further link is found and its associated mapping record is shown below:

Proximate Change mapping record 3
Mapping record ID = 145
Code file ID = trunk/Docs/src/Installation.xml
Region start offset = 30

Region end offset = 40

Timestamp = 20/4/12 16:30

Weight = 20

Proximate Change mapping record 3 (905) is shown in figure 9.

In the another example herein, a section (namely, lines 30-40) associated with the Proximate Change mapping record 3 overlaps entirely with a section (namely, lines 28-48) associated with Change mapping record 1. In response, at step 625, the resolver (515) generates a region record (Region record 4) wherein:

Region record 4 is associated with Proximate Change mapping record 3 and Change mapping record 1

Preferably, the resolver (515) populates the region record with an average start offset and an average end offset based on an average of the start offsets and the end offsets of the associated mapping records.

At step 630, the metrics generator (345) populates the region record with a region importance weight which is the sum of each of the importance weights of the mapping records associated with the region record. This allows more significant region records to be highlighted.

In the remainder of the example below, Region record 1; Region record 2 and Region record 4 will be used and representations are shown below:

Region record 1

Region record ID = 888

Code file ID = trunk/Test/src/com/test/Test.java

Average region start offset = 46

Average region end offset = 52

Region Importance Weight = 1141.7 + 240.8 = 1382.5

Region record 2

Region record ID = 777

Code file ID = trunk/Test/src/com/test/Types.java

Average region start offset = 8

Average region end offset = 9

Region Importance Weight = 902.7 + 190.4 = 1113.1

Region record 4

Region record ID = 666

Code file ID = trunk/Docs/src/Installation.xml

Average region start offset = 29

Average region end offset = 44

Region Importance Weight = 345.2+72.8+20= 438

Region record 1 (705) is shown in figure 7; Region record 2 (805) is shown in figure 8; and Region record 4 (910) is shown in figure 9.

Preferably, the resolver (515) sorts (step 630) the region records in order of region importance weight.

The results of the sorting are depicted below:

1. Region record 1 = 1382.5

2. Region record 2 = 1113.1

3. Region record 4 = 438

The sorting also allows some order to be imposed on the region records obtained as several such records could be obtained, potentially causing the user difficulty in interpretation.

The resolver (515) passes the sorted list to the dependency display generator (510) which displays (step 635) the list.

In the example herein, the dependency display generator (510) displays a list of results starting with a pointer to the file Installation.xml and the associated lines 29-44. Next, the dependency display generator (510) displays a pointer to the file Test.java and the associated lines 46-52. Finally, the dependency display generator (510) displays a pointer to the file Types.java and the associated lines 8-9. Preferably, display of the results can be executed in a number of ways, e.g., if a user selects a result, the associated lines are highlighted within the file. More preferably, the dependency display generator (510) is operable to truncate the list for clarity.

It should be understood that although the resolver (515) sorts the list above based on size of region, the resolver (515) can sort the list on any of the further data e.g., recency of changes; similarity of the changes; complexity of the changes etc.

Advantageously, the preferred embodiment allows a user, in response to making a change to a file, to be able to review sections of the file and/or sections of other files which may require updating also. For example, with reference to figures 7-9 and the example above, when a user makes a change (e.g., proposed change 1 (700) and proposed change 2 (800)), in response to the process of the preferred embodiment, a user is able to view a sorted list of further sections of files which may require updating (e.g., Region record 1 (705); Region record 2 and Region record 4 (910)).

CLAIMS

1. An apparatus for managing changes to one or more files, wherein a link is associated with changes that result in an updated version of the one or more files, the apparatus comprising:

a changed section determiner operable to determine which sections associated with the one or more files have changed and operable to determine metadata regarding the nature of the changes to the sections;

a metrics generator operable to use the metadata to generate a first weight associated with each of the changed sections; and

a mapping record generator operable to generate a mapping record for each of the changed sections, wherein a first mapping record comprises data associated with the first weight; is associated with a particular version of the one or more files; and is linked to mapping records associated with other changed sections using the link.

2. An apparatus as claimed in claim 1, wherein the metadata is associated with at least one of: size of a change; complexity of a change; and recency of a change.

3. An apparatus as claimed in claim 2, wherein the metrics generator is operable to assign a greater first weight to at least one of: a larger size of a change; a greater complexity of a change; and a more recent change.

4. An apparatus as claimed in any preceding claim, further comprising:

a resolver, responsive to receipt of a proposed change, for retrieving each proximate change mapping record associated with previous changes that are proximate to the proposed change.

5. An apparatus as claimed in claim 4, wherein the metrics generator is operable to assign a second weight to the proximate change mapping record.

6. An apparatus as claimed in claim 4 or claim 5, wherein the resolver is operable to obtain further mapping records of further changes having the same link as the proximate change mapping record.

7. An apparatus as claimed in claim 6, wherein the metrics generator is operable to assign a third weight to a further mapping record.

8. An apparatus as claimed in claim 6 or claim 7, wherein the resolver is operable to generate a region record that is an aggregation of each proximate changes and further changes having overlap, wherein the proximate changes and the further changes need not have the same link.

9. An apparatus as claimed in claim 8, wherein the metrics generator is operable to assign a fourth weight to the region record.

10. An apparatus as claimed in claim 8 or claim 9, wherein the resolver is operable to sort one or more region records.

11. A method for managing changes to one or more files, wherein a link is associated with changes that result in an updated version of the one or more files, the method comprising the steps of:

determining which sections associated with the one or more files have changed;

determining metadata regarding the nature of the changes to the sections;

using the metadata to generate a first weight associated with each of the changed sections; and

generating a mapping record for each of the changed sections, wherein a first mapping record comprises data associated with the first weight; is associated with a particular version of the one or more files; and is linked to mapping records associated with other changed sections using the link.

12. An apparatus as claimed in claim 11, wherein the metadata is associated with at least one of: size of a change; complexity of a change; and recency of a change.

13. A method as claimed in claim 12, further comprising the step of:

assigning a greater first weight to at least one of: a larger size of a change; a greater complexity of a change; and a more recent change.

14. A method as claimed in any of claims 11 to 13, further comprising the step of:

retrieving, in response to receipt of a proposed change, each proximate change mapping record associated with previous changes that are proximate to the proposed change.

15. A method as claimed in claim 14, further comprising the step of:
assigning a second weight to the proximate change mapping record.

16. A method as claimed in claim 14 or claim 15, further comprising the step of:
obtaining further mapping records of further changes having the same link as the proximate change mapping record.

17. A method as claimed in claim 16, further comprising the step of:
assigning a third weight to a further mapping record.

18. A method as claimed in claim 16 or claim 17, further comprising the step of:
generating a region record that is an aggregation of each proximate changes and further changes having overlap, wherein the proximate changes and the further changes need not have the same link.

19. A method as claimed in claim 18, further comprising the step of:
assigning a fourth weight to the region record.

20. A method as claimed in claim 18 or claim 19, further comprising the step of:
sorting one or more region records.

21. A computer program comprising program code means adapted to perform all the steps of any of claims 11 to 20 when said program is run on a computer.

Application No:	GB1211485.6	**Examiner:**	Jake Collins
Claims searched:	1-21	**Date of search:**	9 October 2012

Patents Act 1977: Search Report under Section 17

Documents considered to be relevant:

Category	Relevant to claims	Identity of document and passage or figure of particular relevance
A	-	US2011/0239192 A1 (FEIGEN)
A	-	WO2008/063797 A2 (AUTODESK)
A	-	US 2008/0250394 A1 (JONES ET AL)
A	-	US 5047918 A (SCHWARTZ ET AL)

Categories:

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

Field of Search:

Search of GB, EP, WO & US patent documents classified in the following areas of the UKC^X :

Worldwide search of patent documents classified in the following areas of the IPC

G06F

The following online and other databases have been used in the preparation of this search report

WPI, EPODOC

International Classification:

Subclass	Subgroup	Valid From
G06F	0017/22	01/01/2006