(12) UK Patent Application (19) GB (11) 2529172 (13) A

(43) Date of A Publication 17.02.2016

(21) Application No: 1414239.2

(22) Date of Filing: 12.08.2014

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

(72) Inventor(s):
James Kenneth Hook
Hamish Cunha Hunt
Nicholas Karl Lincoln
Simon Andrew Satoshi Briggs

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

(51) INT CL:
G06F 9/40 (2006.01) ***G06F 17/22*** (2006.01)
G06F 17/30 (2006.01)

(56) Documents Cited:
WO 2009/076010 A1
Collins-Sussman et al, Vendor Branches, downloaded 27/01/14 from the internet at http://svnbook.red-bean.com/en/1.7/svn.advanced.vendorbr.html

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

(54) Title of the Invention: **Functional component history tracking**
Abstract Title: **Functional component history tracking by creating a bifurcated version history hierarchy**

(57) Tracking functional components in a version history tracker, the functional components having content. It is determined (1504) whether the source of a functional component is known to the version history tracker; responsive to determination that the source of the functional component is not known to the version history tracker, performing an automated analysis (1506) of the content of the functional component; responsive to determination that the source of the functional component is known to the version history tracker, creating a bifurcated version history hierarchy (1508) of the functional component using partial or full data transfer from one functional component to another functional component irrespective of any metadata associated with either of the functional components; and responsive to said automated analysis and said bifurcated version history hierarchy, producing a branched network (1510) of the version history for the functional component.

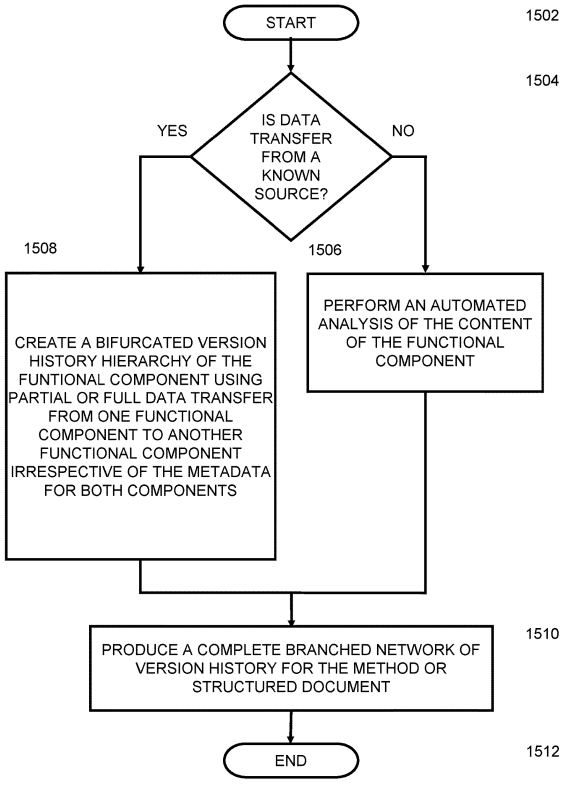
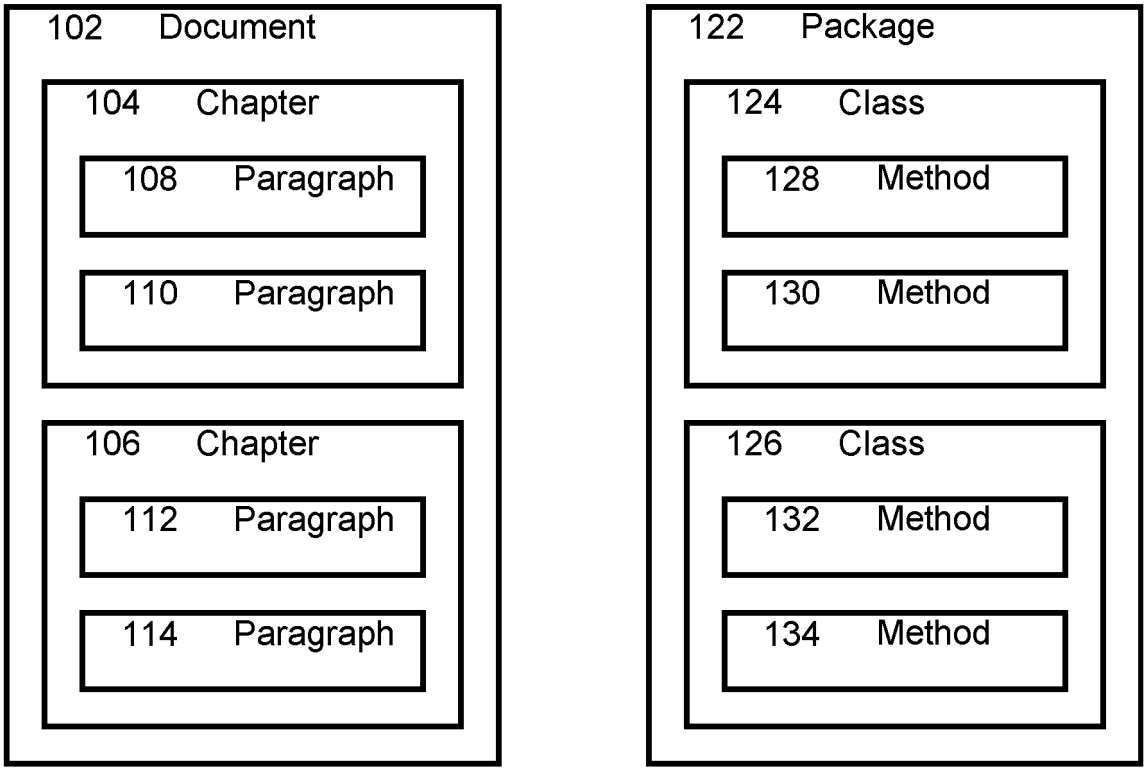


Figure 15

GB 2529172 A



PRIOR ART

Figure 1

302	Class	Classname
304	UCID	1000
306	Component	Class
	Content	
308	Timeline	
310	t0 diff	< 1st version >
312	t1 diff	< 2nd version >

Figure 3

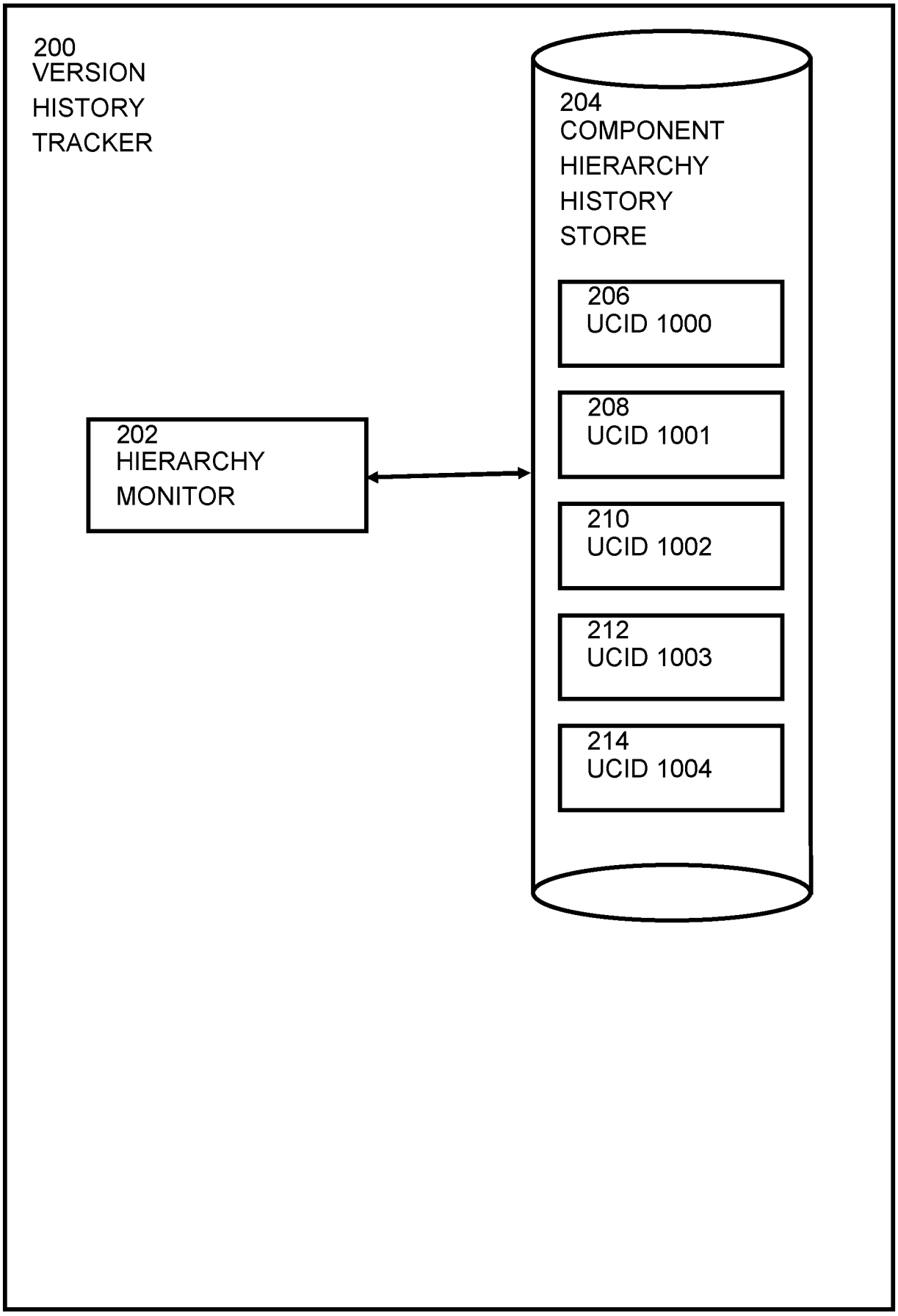

200
VERSION
HISTORY
TRACKER
204
COMPONENT
HIERARCHY
HISTORY
STORE
206
UCID 1000
208
UCID 1001
210
UCID 1002
212
UCID 1003
214
UCID 1004
202
HIERARCHY
MONITOR

Figure 2

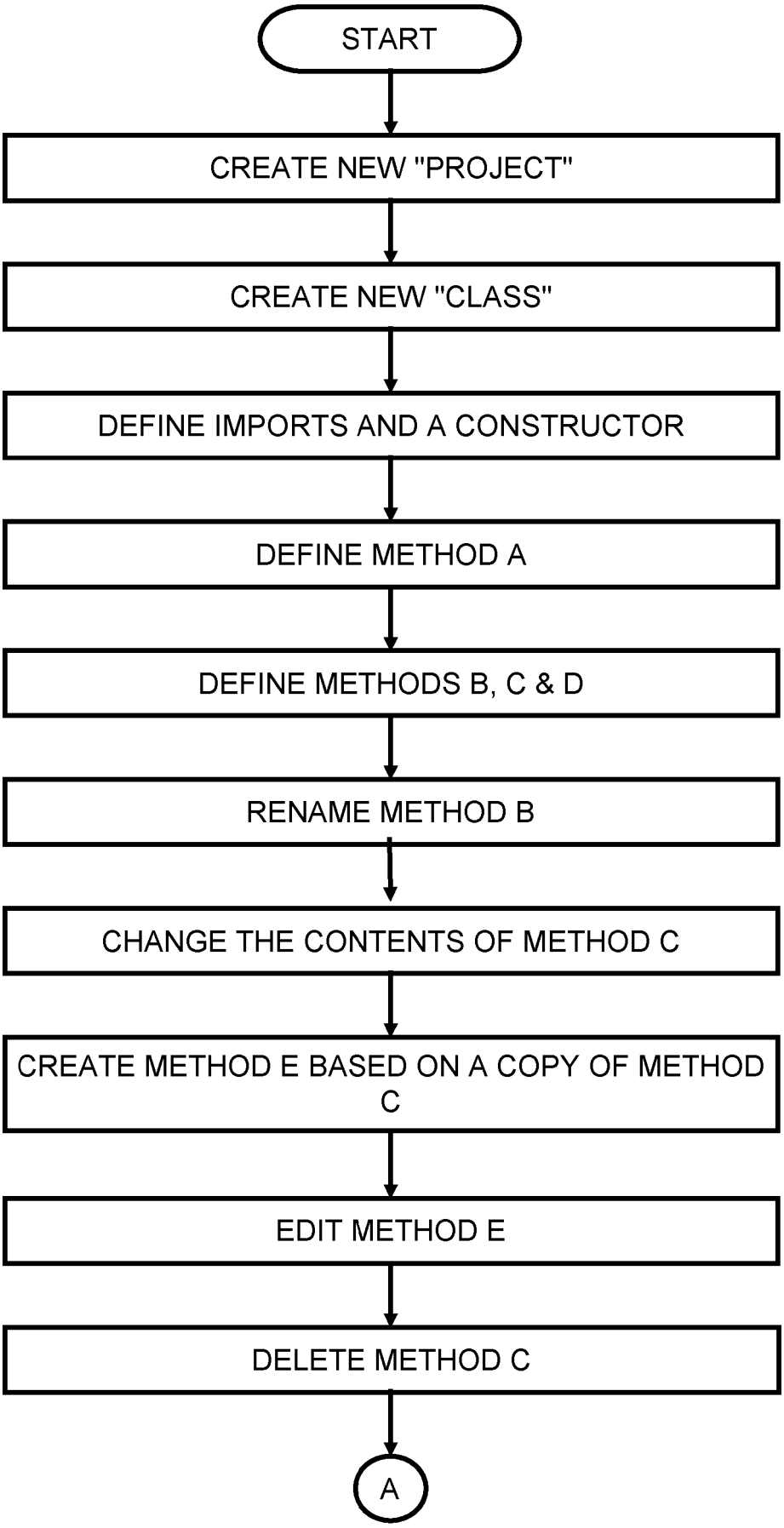

START
402
CREATE NEW "PROJECT"
404
CREATE NEW "CLASS"
406
DEFINE IMPORTS AND A CONSTRUCTOR
408
DEFINE METHOD A
410
DEFINE METHODS B, C & D
412
RENAME METHOD B
414
CHANGE THE CONTENTS OF METHOD C
416
CREATE METHOD E BASED ON A COPY OF METHOD C
418
EDIT METHOD E
420
DELETE METHOD C
422
A

Figure 4A

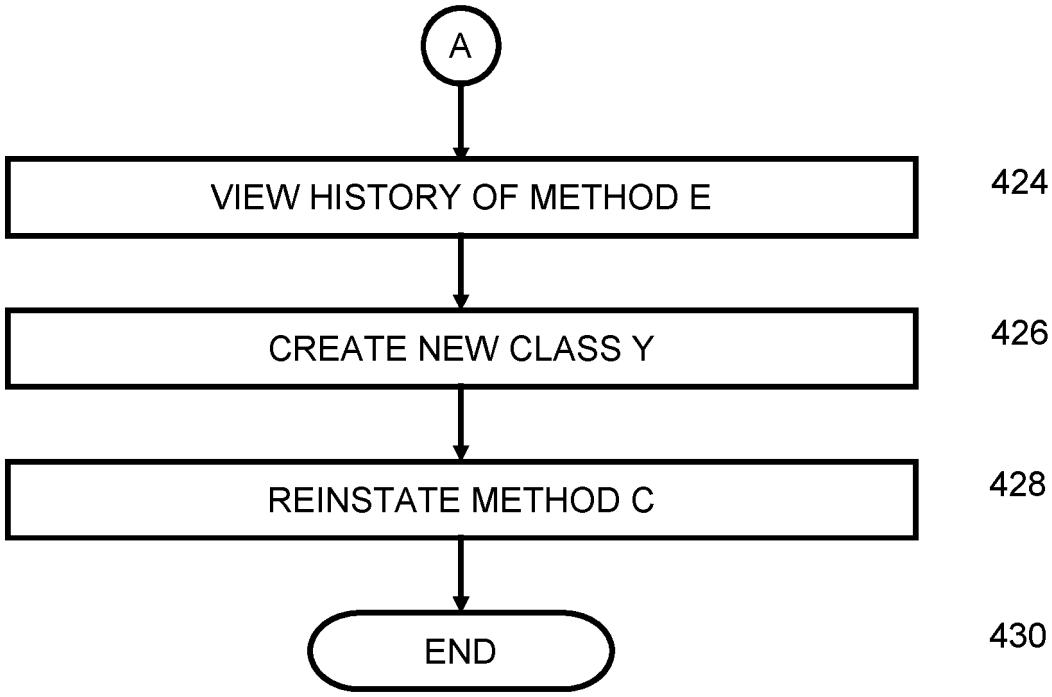


Figure 4B

206	Class	Z
502 504 506 508	UCID Component Content Timeline t0 diff t1 diff	1000 Class < NULL > < import java.awt <constructor > >

Figure 5

208	Method	A
602 604 606	UCID Component Content Timeline t0 diff	1001 Method int i=1; int a=1

210	Method	B
622 624 626	UCID Component Content Timeline t0 diff	1002 Method int i=1; int b=1

212	Method	C
642 644 646	UCID Component Content Timeline t0 diff	1003 Method int i=1; int c=1;

214	Method	D
662 664 666	UCID Component Content Timeline t0 diff	1004 Method int i=1; int d=1;

Figure 6

210	Method	Renamed B
622	UCID	1002
624	Component	Method
	Content	
	Timeline	
626	t0 diff	int i=1; int b=1
702	t1 diff	name=Renamed B

Figure 7

212	Method	C
642	UCID	1003
644	Component	Method
	Content	
	Timeline	
646	t0 diff	int i=1; int c=1;
802	t1 diff	int i=2; int c=2;
804	t2 diff	int i=2; int c=2;

Figure 8

902	Method	E
904	UCID	1005
906	Component	Method
	Content	
	Timeline	
908	t0 diff	1003 → t2

Figure 9

902	Method	E
904	UCID	1005
906	Component	Method
	Content	
	Timeline	
908	t0 diff	1003 → t2
1002	t1 diff	d int c=3; a int e=5;

Figure 10

212	Method	C
642	UCID	1003
644	Component	Method
	Content	
	Timeline	
646	t0 diff	int i=1; int c=1;
802	t1 diff	int i=2; int c=2;
804	t2 diff	int i=2; int c=2;
1102	t3 diff	d int i=2; int c=2;

Figure 11

1200	Method	C (reinstated to new class)
1202	UCID	1007
1204	Component	Method
	Content	
	Timeline	
1206	t0 diff	Int i=2; int c=2;

Figure 12

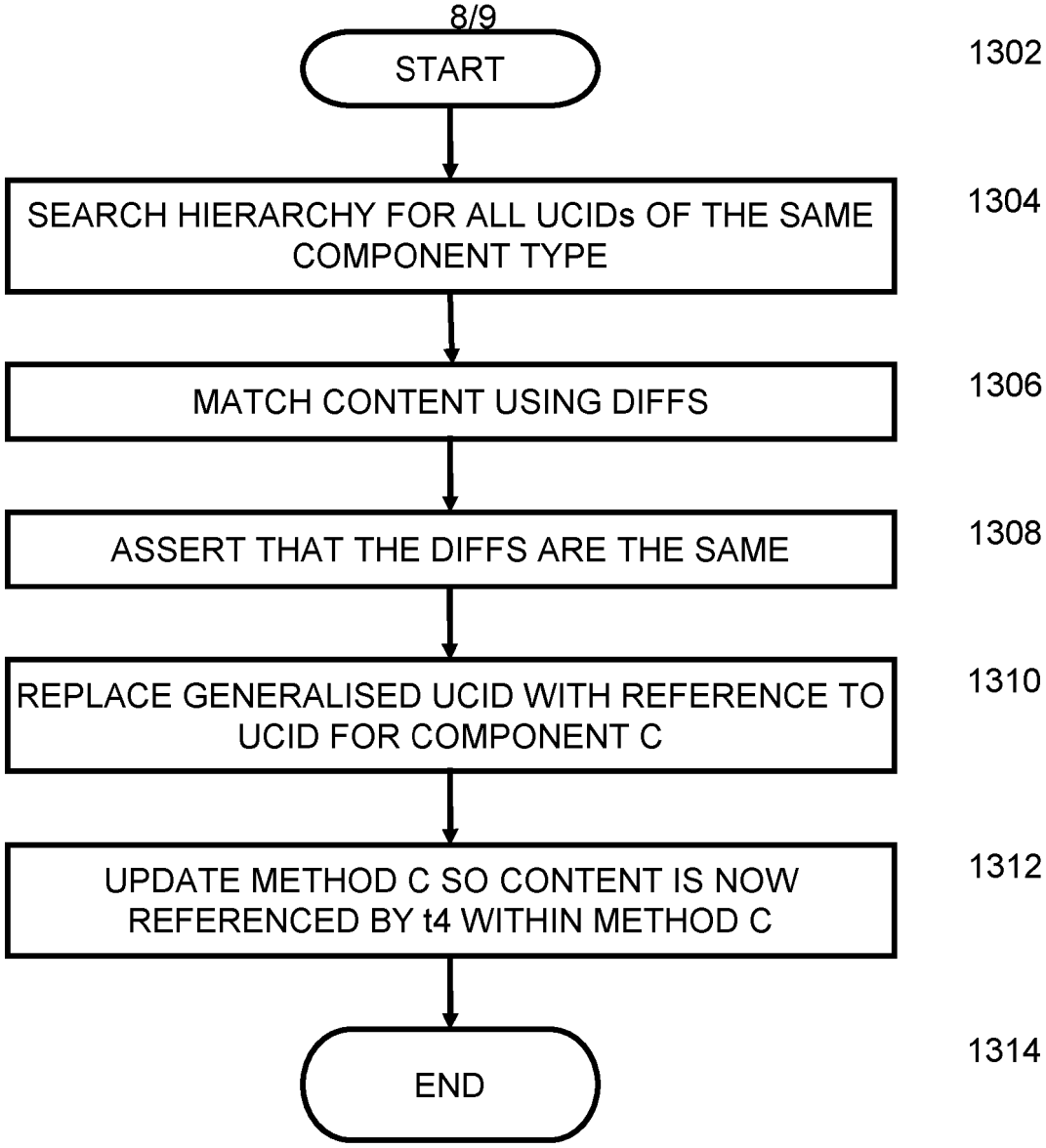


Figure 13

1200	Method	C
1202	UCID	1003
1204	Component	Method
	Content	
	Timeline	
646	t0 diff	int i=1; int c=1;
802	t1 diff	int i=2; int c=2;
804	t2 diff	int i=2; int c=2;
1102	t3 diff	d int i=2; int c=2;
1402	t4 diff	int i=2; int c=2;

Figure 14

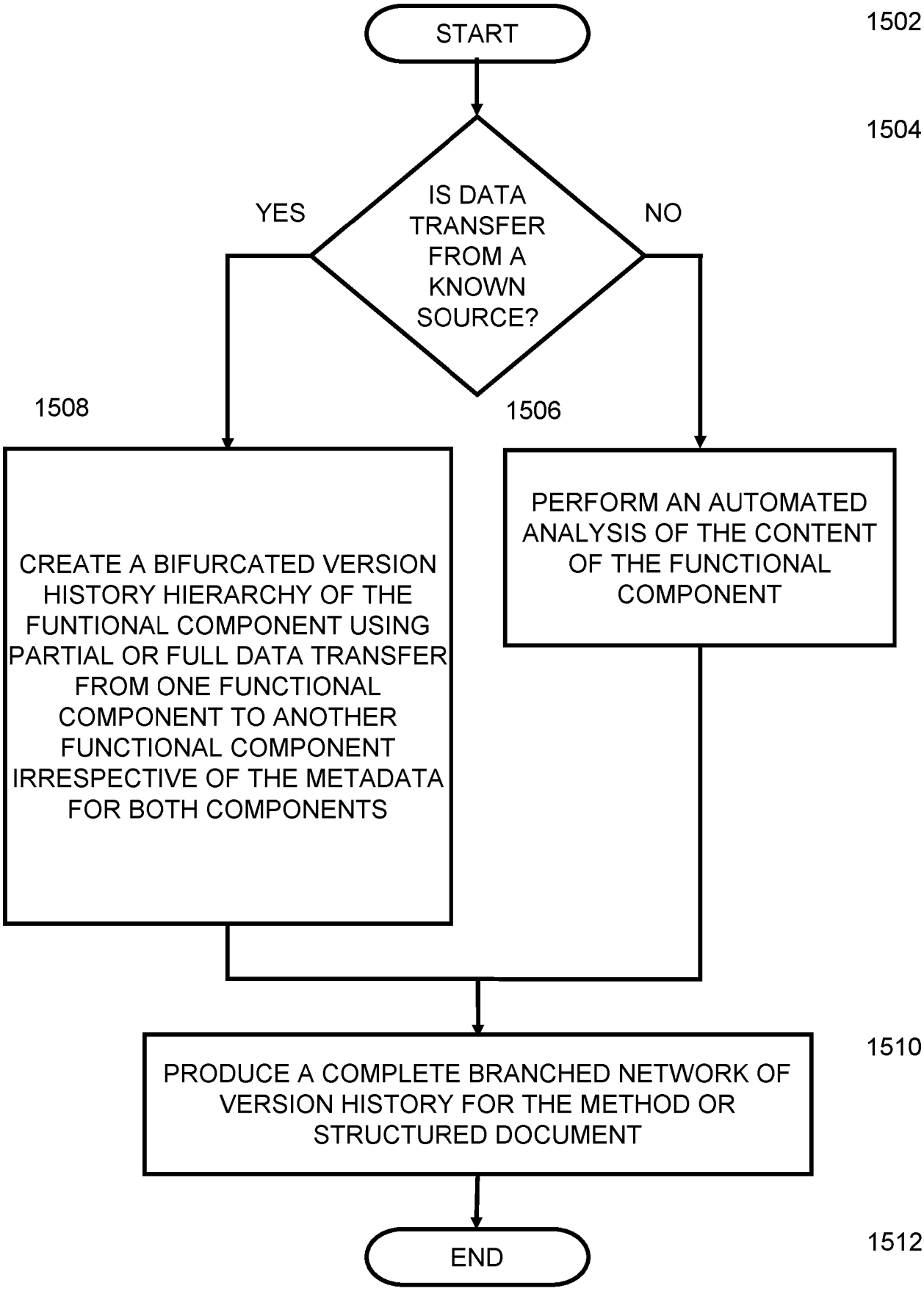

START
1502
1504
IS DATA TRANSFER FROM A KNOWN SOURCE?
YES
NO
1508
1506
CREATE A BIFURCATED VERSION HISTORY HIERARCHY OF THE FUNTIONAL COMPONENT USING PARTIAL OR FULL DATA TRANSFER FROM ONE FUNCTIONAL COMPONENT TO ANOTHER FUNCTIONAL COMPONENT IRRESPECTIVE OF THE METADATA FOR BOTH COMPONENTS
PERFORM AN AUTOMATED ANALYSIS OF THE CONTENT OF THE FUNCTIONAL COMPONENT
PRODUCE A COMPLETE BRANCHED NETWORK OF VERSION HISTORY FOR THE METHOD OR STRUCTURED DOCUMENT
1510
END
1512

Figure 15

FUNCTIONAL COMPONENT HISTORY TRACKING

FIELD OF THE INVENTION

[0001] The present invention relates to the tracking of the history of functional components, and more particularly to the production of a branched network of the version history for a functional component, such as a file, a method or a structured document.

BACKGROUND

[0002] There are currently numerous tools to enable a user to inspect a document history or to perform a comparison between different versions of a document. Whilst it is possible to use software such as Beyond Compare from Scooter Software Inc and SlickEdit® from SlickEdit Inc. to view single instance changes to data files, it would be desirable to be able to search data file changes from their current state back to their inception. Document histories can often be viewed inside the native tool within which they are developed. A user may look though previous auto-saves or previous commit versions as is the case with version control software.

[0003] These version history trackers show all changes to a document as it evolves, enabling the user to move back through the change history in a chronological order. Version history trackers highlight any changes within the contents of an entire document by doing a string comparison between the document and a previous version of that document. However, the user is restricted to seeing isolated document changes only. If two seemingly disparate sections of two different documents are the same or originate from a common origin, then this will not be visible in a version history tracker.

[0004] It would be desirable to be able to inspect changes that are restricted to a component of the document and to ignore all unrelated document changes. This is especially pertinent when the document is computer code, where a user may only be interested in the changes made to a particular method within a class or a subset of the function. It is equally

applicable to other traditional components such as chapters or paragraphs within a text document.

[0005] It would further be desirable to carry out a more targeted search for document histories, to inspect changes with much more precision, timeliness and rigor. A typical problem is when changes across multiple files occur as a result of a code refactor. Typically, a user is only interested in changes in a single method of the file, which may or may not have been moved from another file.

[0006] United States Patent Application 2009/0293043 A1 discloses that in instruction sets which may be developed through many versions, groups of instructions may be changed to achieve improvements in prior versions of the instruction sets (e.g., to correct errors, to add features, or to improve compatibility with other components.) These new versions may be tracked, e.g., in a versioning tool that may record changes among versions of an instruction and may display a version of the instruction set at a particular stage of development. A development environment may store and utilize associations of a version of an instruction with observations recorded in a bug database that indicate an undesirable behavior caused by the instruction, and with notes by the developer of the version concerning the improvements to be achieved by the version as compared with other versions of the instruction. Accordingly, when a developer wishes to explore the history of an instruction (comprising one or more versions of the instruction, including the current version), the development environment may query the various sources of information about the instruction and aggregate the results into a summary of the version history of the instruction.

[0007] United States Patent Application 2012/0272151 A1 discloses a system that provides a flexible and intuitive approach for displaying and navigating the revision history of a document. A large revision history that includes hundreds of user operations may be reduced to a simple graphical representation that may be navigated by a user to visualize the revision history at finer and finer detail. A user may use tools within the system to filter or search the revision history for particular types of user operations. The hierarchical, high-level clustering algorithm also presents each of the user operations within the context of the

complete revision history, allowing a user to visualize and learn various techniques for creating or modifying the content of a document. In addition, captured video content associated with the revision history may be played back to provide the user context within the application of how a document was revised. Although a more intuitive GUI for viewing version history is disclosed, it provides no knowledge of where any content has come from and is only a method of presenting data within a GUI environment.

[0008] "CoEd - A Tool for Cooperative Development of Hierarchical Documents" (1997) by Lars Bendix, Per Nygaard Larsen, Anders Ingemann Nielsen, Jesper La , Søndergaard Petersen, Fredrik Bajers Vej E, discloses the maintaining of an overview of how a document is evolving, and at the same time the maintaining of a complete version history. Traditional version control is extended by providing version control on both the entire structure of the document and its constituent parts. Although a hierarchical system is disclosed, there is no knowledge by the system of where any internal/external sources of data have come from.

[0009] "Using Version Control History to Follow the Changes of Source Code Elements", Toth, Z., Novak, G., Ferenc, R., Siket, I., 17th European Conference on Software Maintenance and Reengineering (CSMR), 2013, pp. 319 - 322, discloses an algorithm which is able to follow the changes of the source code elements by using the changes of files. The algorithm alerts the user to changes made within tracked components from a specific change set. It is left to the user to infer any possible relationships there may or may not be between all methods and classes.

BRIEF SUMMARY OF THE INVENTION

[0010] Embodiments of the invention provide a method for use in a version history tracker, of tracking functional components, the functional components having content, the method comprising: determining whether the source of a functional component is known to the version history tracker; responsive to determination that the source of the functional component is not known to the version history tracker, performing an automated analysis of the content of the functional component; responsive to determination that the source of the

functional component is known to the version history tracker, creating a bifurcated version history hierarchy of the functional component using partial or full data transfer from one functional component to another functional component irrespective of any metadata associated with either of the functional components; and responsive to said automated analysis and said bifurcated version history hierarchy, producing a branched network of the version history for the functional component. Embodiments of the invention have the advantage that similar functional components can be associated to create branched hierarchies from a singular unified identifier for a functional component. This has the benefit that two or more seemingly unrelated functional components can be tracked back to a single source that may or may not still exist. If the single source does not exist, embodiments of the present invention retain the identifier of the single source.

[0011] In an embodiment of the method of the present invention, said version history tracker further comprises a component hierarchy history store having unique identifiers, component types and differences between versions of functional components stored therein; said step of performing an automated analysis of the content of the functional component comprises the steps of: searching the component hierarchy history store for all unique identifiers of the same component type as the functional component to be analyzed; and using the differences between versions of the functional component to identify if the functional component corresponds to any of the functional components in the component hierarchy history store. This has the advantage that versions of deleted functional components that are no longer in the source of the functional component, such as deleted methods no longer in the source code, are available for tracking as the unique identifier for them is retained, even after they are deleted. This means that there is the ability to track functional components that have been reinstated from an external source.

[0012] In a further embodiment of a method of the present invention, said step of performing an automated analysis of the content of the functional component further comprises the steps of: responsive the functional component corresponding to a one of the functional components in the component hierarchy history store, replacing the unique identifier of the functional component with a reference to the unique identifier of the one of the functional components in the component hierarchy history store; and updating the

functional component by incorporating the version history of the corresponding one of the functional components in the component hierarchy history store. This has the advantage that, content analysis of two seemingly disparate functional components can be used to automatically suggest and/or implement version merges.

[0013] In an embodiment, said functional component is a computer program method usable within a computer program package.

[0014] In another embodiment, said functional component is a paragraph usable within a structured document.

[0015] Embodiments of the invention also provide a system for use in a version history tracker, of tracking functional components, the functional components having content, the system comprising: a hierarchy monitor, the hierarchy monitor: determining whether the source of a functional component is known to the version history tracker; responsive to determination that the source of the functional component is not known to the version history tracker, performing an automated analysis of the content of the functional component; responsive to determination that the source of the functional component is known to the version history tracker, creating a bifurcated version history hierarchy of the functional component using partial or full data transfer from one functional component to another functional component irrespective of any metadata associated with either of the functional components; and responsive to said automated analysis and said bifurcated version history hierarchy, producing a branched network of the version history for the functional component.

[0016] Embodiments of the invention also provide a computer program product for use in a version history tracker, for tracking functional components, the functional components having content, the computer program product comprising: a computer readable storage medium having computer readable program code embodied therewith, the computer readable program code adapted to perform the method described above when said program is run on a computer.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] Preferred embodiments of the present invention will now be described in more detail, by way of example only, with reference to the accompanying drawings, in which:

Figure 1 shows a prior art document and a prior art computer program structure in which the present invention may be implemented;
Figure 2 shows a block diagram of a version history tracker according to an embodiment of the present invention;
Figure 3 shows a component according to an embodiment of the present invention and its constituent elements;
Figures 4A and 4B show a flow chart of the evolution of a plurality of components to demonstrate usage of embodiments of the present invention;
Figure 5 shows a component of type class including elements of history tracking according to embodiments of the present invention;
Figure 6 shows a plurality of components of type method including elements of history tracking according to embodiments of the present invention;
Figure 7 shows a version of the method B component of figure 6 which has been renamed;
Figure 8 shows a version of the method E component which has been generated as a copy of the method C component of figure 6;
Figure 9 shows the method E component of figure 8 after it has been edited;
Figure 10 shows a version of the method C component of figure 6 which has been deleted;
Figure 11 shows a version of the method C component of figure 6 which has been reinstated to a new class Y;
Figure 12 shows a flow chart of an embodiment of a method of history tracking of functional components according to the present invention;
Figure 13 shows a flow chart of an embodiment of a method of performing an analysis of the content of a functional component within a version history tracker according to the present invention;
Figure 14 shows a version of the method C component of figure 6 after processing by the embodiment of figure 13; and

Figure 15 shows an embodiment of a method of transferring data into a version history tracker according to the present invention.

DETAILED DESCRIPTION OF THE EMBODIMENTS

[0018] Figure 1 shows a prior art document 102 and a prior art computer program package 122 in which the present invention may be implemented. Document 102 comprises a plurality of chapters 104, 106. Each chapter 104, 106 comprises a plurality of paragraphs 108, 110, 112, 114. The paragraphs 108, 110, 112, 114 and chapters 104, 106 form a hierarchical structure under the document 102. Similarly, computer program package 122 comprises a plurality of classes 124, 126. Each class 124, 126 comprises a plurality of methods 128, 130, 132, 134. The methods 128, 130, 132, 134 and classes 124, 126 form a hierarchical structure under the package 122.

[0019] Figure 2 shows a block diagram of a version history tracker 200 according to an embodiment of the present invention. A hierarchy monitor 202 monitors the hierarchical structure of a component hierarchy history store 204. The component hierarchy history store 204 comprises entries for each UCID 206, 208, 210, 212, 214. Further entries for other UCIDs may also be stored, but are not shown in figure 2 for clarity. A pointer from a UCID 206, 208, 210, 212, 214 to a related UCID 206, 208, 210, 212, 214 may be stored. A UCID 206, 208, 210, 212, 214 may be related to another UCID 206, 208, 210, 212, 214 if, for example, it is a method within the same class. Associated with each UCID 206, 208, 210, 212, 214 is a respective component 302.

[0020] Figure 3 shows a component 302 according to an embodiment of the present invention and its constituent elements. Component 302 comprises a Universal Component IDentifier (UCID) 304, which in the example of figure 3 is "1000". A component hierarchy history store database 204 of UCIDs 304 is maintained and was described with reference to figure 2. Component 302 further comprises an indication 306 of the type of component 302, which in the example of figure 3 is "Class". Component 302 further comprises content, which is a timeline 308 of the different versions of the component 302. In the example of

figure 3, there are two different versions shown, a first version t0 diff 310 and a second version t1 diff 312.

[0021] Figures 4A and 4B show a flow chart of an exemplary embodiment showing the evolution of a plurality of components and the history tracking of those components. The method of the exemplary embodiment starts at step 402. At step 404, a new "project" is started. In the exemplary embodiment, the project is a Java® project. However, embodiments of the present invention are not limited to Java projects and may be implemented in other programming languages having hierarchical constructs or in, for example, documents, such as the document of figure 1. Metadata for the project may be defined. The project is a component at the top of the hierarchical structure. The project contains pointers to components at the next level down in the hierarchy which are files. Each of the files contains pointers to the components at the next level down which is are classes (124, 126 in figure 1). Each of the classes contains pointers to the next level down which are methods (128, 130, 132, 134 in figure 1). The methods are at the bottom of the hierarchical structure. Each of the components (project, file, class and method) is identified by its UCID (304 in figure 3).

[0022] At step 406, a new class Z (206 in figure 5) is created. Referring briefly to figure 5 which shows the components for the class Z 206. The class Z 206 is assigned a new UCID 502. The indication 504 of the type of component 206 is "Class". The t0 diff 506 is set to blank as the class 124, 126 currently has no content. The project detects and points to the newly created UCID 502. This may be by a refresh or it may occur automatically without a refresh. At step 408, some imports and a constructor are defined for a class Z 206. The import may be, for example, "import java.awt". The constructor may be, for example, a standard java empty constructor, that is "public <method name>(){}". The imports and the constructor are contained within the component at t1 diff 508.

[0023] At step 410, a method A 208 within the class Z 206 is defined. For the purposes of this description only, the contents of method A is:

```
int i=1;
int a=1;
```

[0024] The invention is not limited to this particular content, but rather the simple content above has been chosen to allow explanation of the invention with a maximum of clarity.

[0025] Referring briefly to figure 6 which shows the components for the method A 208. The method A 208 is assigned a new UCID 602 of "1001". The component contains an identifier 604 indicating that it is of type method. The t0 diff 606 is "int i=1; int a=1;". There is no t1 diff as there have been no changes to the method at this stage.

[0026] At step 412, further methods B, C & D within the class Z 206 are defined in the same manner to that described above for method A 208. The components for each of these methods are shown in figure 6 as components 210, 212 and 214 respectively. The contents of the methods B, C & D are:

Method B	Method C	Method D
int i=1;	int i=1;	int i=1;
int b=1;	int c=1;	int d=1;

[0027] The invention is not limited to this particular content for these methods, but rather the simple content above has been chosen to allow explanation of the invention with a maximum of clarity.

[0028] Referring briefly to figure 6 which shows the components for the methods B 210, C 212 and D 214. The methods B 210, C 212 and D 214 are assigned new UCIDs 622, 642, 662 of 1002, 1003 and 1004 respectively. The component for the methods B 210, C 212 and D 214 contains identifiers 624, 644 and 664 respectively indicating that they are of type method. The t0 diff 626 for method B 210 is "int i=1; int b=1;", for method C 212 the t0 diff 646 is "int i=1; int c=1;" and for method D 214 the t0 diff 666 is "int i=1; int c=1;". There are no t1 diffs as there have been no changes to methods B 210, C 212 or D 214 at this stage.

[0029] At step 414, method B is renamed. The new method is shown in figure 7 as "Renamed B" 210. A new entry is created at t1 diff 702 which contains an indicator such as

"name=Renamed B" within the change set. This indicator can be a field or other indicator contained within the diffs. The UCID 622 and the component type 624 are unchanged.

[0030] At step 416, the contents of method C are changed by the user to "int i=2; int; c=2;". Referring briefly to figure 8, this change is reflected at t1 in t1 diff 850. At step 418, a new method E 942 is created based on a copy of the existing method C 212. The new method E 942 is shown in figure 9 as "E". A new UCID 944 of "1005" for the method E 942 is assigned.. The component type 946 "Method" for method E 942 is the same as that for method C 212. A new entry is created at t0 diff 948 which refers to the UCID 642 of "1003" for method C 212. Optionally, a forced save of method C 212 may be done at time t2 so that it is easier to see that when the copy of method C 212 was created as method E 942, the state of method C 212 at that time can be seen. The state of method C 212 at t2 is shown in figure 8 as t2 diff 804.

[0031] At step 420, the method E 942 created at step 416 is now edited to delete "int c=3;" and to add "int e=5;". The edited method is shown in figure 10 as "E". A new entry is created at t1 diff 1050. The entry of "d int c=3; a int e=5;" shows that "int c=3;" has been deleted and "int e=5" had been added.

[0032] At step 422, the method C 212 created at step 412 is now deleted. The deleted method is shown in figure 11 as "C". A new entry is created at t3 diff 1102. The entry of "d int i=1; int c=3;" shows that "int i=1; int c=3;" has been deleted. The UCID 642 of "1003" remains in a UCID database stored in the component hierarchy history store 204.

[0033] At step 424, the history of method E 942 is viewed. Figure 10 shows the content of the component history for method E 942. At t1 diff 1050 is shown the most current changes to method E 942, that is "d int c=3; a int e=5". At t0 diff 948 is shown a pointer to the UCID on which method E 942 is based, in this case a UCID of "1003", which corresponds to method C 212. The pointer also indicates the specific diff, in this case t2, of method C 212 which method E 942 was copied from. Referring to figure 11 for method C 212 pointed at by t0 diff 948 of method E 942 and continuing to review the component history, t2 diff 804 for method C 212 contains a forced save copy of method C 212 at the

time that method E 942 was created, that is "int i=2; int c=2;". t1 diff 850 for method C 212 contains the contents of method C 212 when it was changed at time t1, that is "int i=2; int c=2;". t0 diff 646 for method C 212 contains the contents of method C 212 when it was first created, that is "int i=1; int c=1;" As there is no reference from method C 212 to any other UCID, then the end of the history has been reached.

[0034] In summary, the history of method E 942 when viewed in the preceding paragraph is:

Method E t1 diff d int c=3; a int e=5
Method E t0 diff Reference to diff t2 of UCID of method C
Method C t2 diff int i=2; int c=2;
Method C t1 diff int i=2; int c=2;
Method C t0 diff int i=1; int c=1;

[0035] Note that t3 diff 1102 of method C 212 is not included in the history of method E 942 because method E 942 was created from method C 212 at a time when method C 212 was at the version documented by t2 diff 804. Subsequent to the creation of method E 942 from method C 212, method C 212 was edited and t3 diff 1102 for method C 212 created.

[0036] At step 426, a new class Y is created in the same manner as described above at step 406 when the class Z 206 was created.

[0037] At step 428, method C 212 is reinstated from an external source of which the history tracker has no knowledge. Figure 12 shows the reinstated method C 1200. The UCID 1202 is initially a new UCID of "1007". The component type 1204 is "Method". The t0 diff 1206 is "int i=2; int c=2;", that is the t2 diff for the deleted method C 212. The method of the exemplary embodiment ends at step 428.

[0038] Figure 13 shows how the version history tracker of figure 2 determines that the reinstated method C 1200 corresponds to previously deleted method C 212. The method of an exemplary embodiment starts at step 1302. At step 1304, all UCIDs 206, 208, 210, 212, 214 in the component hierarchy history store 204 of the same component type as the

reinstated method C 1200 are searched. At step 1306, the content associated with each of the UCIDs 206, 208, 210, 212, 214 is compared to the content associated with the reinstated method C 1200. When t2 diff for method 212 having a UCID of "1003" is reached, it is determined that the content of reinstated method C corresponds to original method C 212. At step 1308, it is asserted that the diffs for method C 212 and the reinstated method C 1200 are the same. At step 1310, the UCID 1202 of "1007" of the reinstated method C 1200 is replaced with a reference to the UCID for method C 212, that is a UCID of "1003". At step 1312, the t0 diff 646, the t1 diff 802, the t2 diff 804 and the t3 diff 1102 of method C 212 are added to the reinstated method C 212. The original t0 diff of the reinstated method C 1200 now becomes the t4 diff 1402 of method C 1200. The method ends at step 1314.

[0039] Referring to figure 14, the UCID 1202 of "1003" is shown, together with the t0 diff 646, the t1 diff 802, the t2 diff 804, the t3 diff 1102 and the t4 diff 1402 of method C 1200.

[0040] Figure 15 shows an embodiment of a method of transferring data into a version history tracker according to the present invention. The embodiment of the method starts at step 1502. At step 1504 it is determined whether a data transfer into a version history tracker 200 of a functional component 108, 128 is from a source known to the version history tracker 200 or from a source unknown to the version history tracker 200.

[0041] If a functional component was transferred from an unknown source, then processing passes to step 1506. An example of a data transfer of a functional component is that of step 428 of figure 4B where method C 1200 was reinstated from an external source of which the version history tracker 200 has no knowledge. At step 1506, an automated analysis of the content of the functional component is performed. Such analysis may include the assignment of a unique UCID. An example of the automated analysis was described with reference to figure 13 above which resulted in the assigned UCID being replaced with the UCID for the already known method C 212.

[0042] If a functional component was transferred from a known source, then processing passes to step 1508. At step 1508, the UCID of the functional component is used to locate

the functional component in the component hierarchy history store. A bifurcated version history hierarchy of the functional component is the created using partial or full data transfer from one functional component to another functional component. Any metadata associated with either of the functional components is ignored.

[0043] At step 1308, a branched network of the version history of the functional component is produced using the information created by steps 1506 or 1508. Such metadata may typically include metadata to associate versions of a static object, including creation date, author and the like. Such metadata is unnecessary in embodiments of the present invention because versions of a static object can be associated by analyzing changes made to the object itself. The embodiment of the method of the present invention ends at step 1510.

[0044] A particular advantage of embodiments of the present invention is that as every functional component created has a unique UCID assigned to it upon creation that is retained by the version history tracker 200 several new features are enabled. Versions of deleted functional components that are no longer in the source of the functional component, such as deleted methods no longer in the source code, are available for tracking as the UCID for them is retained, even after they are deleted. This feature gives the ability to track functional components that have been reinstated from an external source. In the case, for instance, that a functional component is deleted, is stored in an external source such as a text file and is reinstated, the analysis by the version history tracker of the system as a whole for the same functional component has the capability to reinstate this new functional component as a version of a previously deleted functional component and not just as a new functional component from some other source. This functionality is possible due to the structure and analysis stages, described above with reference to figure 13.

[0045] A further advantage of embodiments of the present invention is that, based on a similar argument to the above, content analysis of two seemingly disparate functional components can be used to automatically suggest and/or implement version merges.

[0046] A yet further advantage of embodiments of the present invention is that because a set of UCIDs is maintained in the component hierarchy history store, similar functional

components can be associated to create branched hierarchies from a singular unified UCID. This has the advantage that two or more seemingly unrelated functional components can be tracked back to a single source that may or may not still exist. If the single source does not exist, embodiments of the present invention retain the UCID of the single source.

[0047] The computer readable storage medium can be a tangible device that can retain and store instructions for use by an instruction execution device. The computer readable storage medium may be, for example, but is not limited to, an electronic storage device, a magnetic storage device, an optical storage device, an electromagnetic storage device, a semiconductor storage device, or any suitable combination of the foregoing. A non-exhaustive list of more specific examples of the computer readable storage medium includes the following: 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), a static random access memory (SRAM), a portable compact disc read-only memory (CD-ROM), a digital versatile disk (DVD), a memory stick, a floppy disk, a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon, and any suitable combination of the foregoing. A computer readable storage medium, as used herein, is not to be construed as being transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide or other transmission media (e.g., light pulses passing through a fiber-optic cable), or electrical signals transmitted through a wire.

[0048] Computer readable program instructions described herein can be downloaded to respective computing/processing devices from a computer readable storage medium or to an external computer or external storage device via a network, for example, the Internet, a local area network, a wide area network and/or a wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and/or edge servers. A network adapter card or network interface in each computing/processing device receives computer readable program instructions from the network and forwards the computer readable program instructions for

storage in a computer readable storage medium within the respective computing/processing device.

[0049] Computer readable program instructions for carrying out operations of the present invention may be assembler instructions, instruction-set-architecture (ISA) instructions, machine instructions, machine dependent instructions, microcode, firmware instructions, state-setting data, or either source code or object code written in any combination of one or more programming languages, including an object oriented programming language such as Smalltalk, C++ or the like, and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The computer readable program instructions 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). In some embodiments, electronic circuitry including, for example, programmable logic circuitry, field-programmable gate arrays (FPGA), or programmable logic arrays (PLA) may execute the computer readable program instructions by utilizing state information of the computer readable program instructions to personalize the electronic circuitry, in order to perform aspects of the present invention.

[0050] Aspects of the present invention are described herein 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 readable program instructions.

[0051] These computer readable 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 readable program instructions may also be stored in a computer readable storage medium that can direct a computer, a programmable data processing apparatus, and/or other devices to function in a particular manner, such that the computer readable storage medium having instructions stored therein comprises an article of manufacture including instructions which implement aspects of the function/act specified in the flowchart and/or block diagram block or blocks.

[0052] The computer readable program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other device to cause a series of operational steps to be performed on the computer, other programmable apparatus or other device to produce a computer implemented process, such that the instructions which execute on the computer, other programmable apparatus, or other device implement the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0053] 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 instructions, which comprises one or more executable instructions for implementing the specified logical function(s). 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 carry out combinations of special purpose hardware and computer instructions.

CLAIMS

1. A method for use in a version history tracker, of tracking functional components, the functional components having content, the method comprising:

determining whether the source of a functional component is known to the version history tracker;

responsive to determination that the source of the functional component is not known to the version history tracker, performing an automated analysis of the content of the functional component;

responsive to determination that the source of the functional component is known to the version history tracker, creating a bifurcated version history hierarchy of the functional component using partial or full data transfer from one functional component to another functional component irrespective of any metadata associated with either of the functional components; and

responsive to said automated analysis and said bifurcated version history hierarchy, producing a branched network of the version history for the functional component.

2. A method as claimed in claim 1, wherein:

said version history tracker further comprises a component hierarchy history store having unique identifiers, component types and differences between versions of functional components stored therein;

said step of performing an automated analysis of the content of the functional component comprises the steps of:

searching the component hierarchy history store for all unique identifiers of the same component type as the functional component to be analyzed; and

using the differences between versions of the functional component to identify if the functional component corresponds to any of the functional components in the component hierarchy history store.

3. A method as claimed in claim 2, wherein:

said step of performing an automated analysis of the content of the functional component further comprises the steps of:

responsive the functional component corresponding to a one of the functional components in the component hierarchy history store, replacing the unique identifier of the functional component with a reference to the unique identifier of the one of the functional components in the component hierarchy history store; and

updating the functional component by incorporating the version history of the corresponding one of the functional components in the component hierarchy history store.

4. A method as claimed in claim 1 wherein said functional component is a computer program method usable within a computer program package.

5. A method as claimed in claim 1 wherein said functional component is a paragraph usable within a structured document.

6. A system for use in a version history tracker, of tracking functional components, the functional components having content, the system comprising:

a hierarchy monitor, the hierarchy monitor:

determining whether the source of a functional component is known to the version history tracker;

responsive to determination that the source of the functional component is not known to the version history tracker, performing an automated analysis of the content of the functional component;

responsive to determination that the source of the functional component is known to the version history tracker, creating a bifurcated version history hierarchy of the functional component using partial or full data transfer from one functional component to another functional component irrespective of any metadata associated with either of the functional components; and

responsive to said automated analysis and said bifurcated version history hierarchy, producing a branched network of the version history for the functional component.

7. A system as claimed in claim 6, wherein said system tracker further comprises:

a component hierarchy history store having unique identifiers, component types and differences between versions of functional components stored therein;

said hierarchy monitor performs said automated analysis of the content of the functional component by:

searching the component hierarchy history store for all unique identifiers of the same component type as the functional component to be analyzed; and

using the differences between versions of the functional component to identify if the functional component corresponds to any of the functional components in the component hierarchy history store.

8. A system as claimed in claim 7, wherein:

said hierarchy monitor, responsive the functional component corresponding to a one of the functional components in the component hierarchy history store:

replaces the unique identifier of the functional component with a reference to the unique identifier of the one of the functional components in the component hierarchy history store; and

updates the functional component by incorporating the version history of the corresponding one of the functional components in the component hierarchy history store.

9. A system as claimed in claim 6 wherein said functional component is a computer program method usable within a computer program package.

10. A system as claimed in claim 6 wherein said functional component is a paragraph usable within a structured document.

11. A computer program product for use in a version history tracker, for tracking functional components, the functional components having content, the computer program product comprising:

a computer readable storage medium having computer readable program code embodied therewith, the computer readable program code adapted to perform the method of any one of claim 1 to claim 5 when said program is run on a computer.

12. A method substantially as hereinbefore described, with reference to figures 2, 13 and 15 of the accompanying drawings.

Intellectual Property Office

Application No: GB1414239.2 **Examiner:** Mr Jim Calvert

Claims searched: 1-12 **Date of search:** 27 January 2015

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	1-12	Collins-Sussman et al, Vendor Branches, downloaded 27/01/14 from the internet at http://svnbook.red-bean.com/en/1.7/svn.advanced.vendorbr.html see page 5, "When you run svn_load_dirs.pl...."
A	1-12	WO2009/076010 A1 (MICROSOFT) See e.g. fig.7 and associated text.

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, TXTE, Internet

International Classification:

Subclass	Subgroup	Valid From
G06F	0009/40	01/01/2006
G06F	0017/22	01/01/2006
G06F	0017/30	01/01/2006