http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 9 (5): 949-955, 2010
ISSN 1812-5638

Software Watermarking Based on Condensed Co-Change Graph Cluster

[1,2]Guang Sun and [1]Xingming Sun
[1]School of Computer and Communication, Hunan University, No. 252, Lushan South Road, Changsha, 410082, China
[2]Information Management Department, Hunan College of Finance and Economics, No. 139, Fenglin 2nd Road, Changsha, 410205, China

Abstract: This study presented a method for constructing more accurate condensed co-change graph. By using random walk, all the unchanged artifacts can be merged completely on the condition that the changed artifacts greater than 50%. An extended GN clustering approach is applied to find a good partition of the software condensed co-change graph. The experiment compares the watermark extracting results embedding whole software and embedding the biggest cluster codes by SANDMARK with ten software techniques. The experiment results show that if an attacker copies the biggest cluster codes, only three software watermarking approaches can work. If the watermarks are embedded in the biggest cluster codes, seven watermarking techniques can work.

Key words: Software watermarking, co-changed graph, random walk, cluster, GN approach

INTRODUCTION

Software watermarking, a relatively new research field, is used to discourage intellectual property theft by embedding a secret message W into a program P and reliably extracting W from P when the proof of ownership is needed (Collberg *et al*., 2009).

Recently, there have been many advances in the research underlying software watermarking, but most research focus on the discovery of novel embedding schemes. Few research have considered the embedding objects themselves (Myles and Collberg, 2006; Kamel and Albluwi, 2009). Usually, only one subsystem get plagiarized even if an attacker disassemblies the whole software. If all the watermark codes aren't embedded in the stolen part, the attackers could easily create an unwatermarked (false-negative) version. Thus the watermarks fail to work. Therefore, every subsystem, not the entire program, should be watermarked.

Software shares a graph structure (Christopher, 2003; (Dedrick *et al*., 2010). Well designed software systems are organized into cohesive clusters that are loosely interconnected (Periklis and Vassilios, 2005). A definition of a software cluster (also called community or module) in a weak sense could be that the codes inside a cluster interact more strongly with each others than with the other codes (Periklis Vassilios, 2003). A cluster of software is similar to a subsystem (Beyer and Noack, 2005a, b). In the cases of the following:

- Documentation does not exist
- The documentation hasn't been kept up to date as software grows
- The persons who are using and developing a system are constant changing.
- Models of software are created in the absence of experts

The subsystems are unavailable, but clusters of software will give the researchers a road map of the software's structure and help them understand legacy systems (Wen and Tzerpos, 2004). The attackers use clustering approach to locate the subsystems as well. This gives us an idea of that we can use cluster approach to identify each subsystem, then watermark it.

Constructing graph with unknown their function or function relation is the first step and a key issue to find software cluster structures (Christopher, 2003; Whelan, 2006). Changes of software systems are less expensive and less error-prone if they affect only one subsystem (Beyer and Noack, 2005a, b). Frequently change together artifacts are good cluster candidates. Beyer and Noack (2005a, b), Gall *et al*. (2003), Zimmermann *et al*. (2003, 2004) mentioned the method to build software condensed co-change (commonly change) graph by relating different artifacts, if they all were modified in the same commit. Condensed co-change graph is inexpensively extractable and not limited to program source code, accurately represents commonly changed relationships of the

Corresponding Author: Xingming Sun, School of Computer and Communication, Hunan University, No. 252, Lushan South Road, Changsha, 410082, China Tel: 86-731-88821341 Fax: 86-731-88822417

changed artifacts as well. However, when constructing the condensed co-change graph, the commonly change relationships of never change artifacts are unknown. There is still lack of available techniques to accurately evaluate the relationship of never changed artifacts (Beyer and Noack, 2005a). If the never changed artifacts are abandoned, part commonly change information may lead to bad clustering.

By merging never changed codes with random walk, this study establishes a more accurate condensed co-change graph of the software. Then, an extended GN clustering approach is used to find a good partition of the software condensed co-change graph. At last, the biggest cluster is watermarked with SANDMARK to evaluate the robust.

DEFINITION

Here, a series of definitions are presented which used in the following:

Software Artifact: A software artifact is a piece of software, e.g., source and executable code, database query, test data and script, documentation, etc.

As the codes are only object and the intermedia language code or byte code is general distributing format. In this study, a software artifact is defined as a basic block consisting of .NET Intermedia language codes or JAVA byte codes with a single entry and a single exit.

Version: A version is a copy of the modified software artifact.

CVS: CVS stands for control version system. Users check in the modified software artifacts(versions) to the CVS server in time. The CVS server stores these software artifacts.

Change transaction: A change transaction is a record of the versions which commonly changed.

Co-change graph: Co-change graph is an undirected graph, denoted as CoG (V, E), derived from a given CVS. The set of vertices V contains all versions and all change transactions. The set of edges E contains only the undirected edge $\{a_i, t_j\}$ if artifact a_i was changed by transaction t_j. In Fig. 1a, a_1, a_2, a_3 and a_4 are versions and a_1, a_2, a_3 changed together in the transaction t_1. The artifacts a_2, a_3 and a_4 are changed together in the transaction t_2.

Condensed co-change graph: Condensed co-change graph is a weighted, undirected graph, denoted as CCoG (V, E, W), derived from the co-change graph.

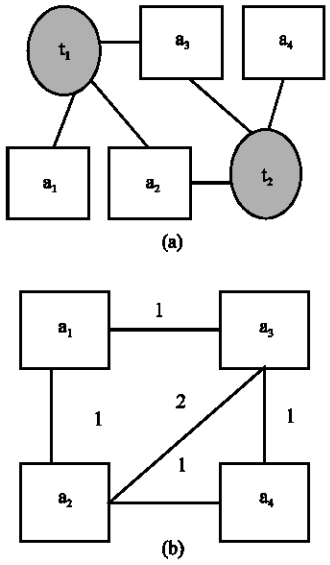

Fig. 1: Example of (a) co-change graph and (b) condensed co-change graph

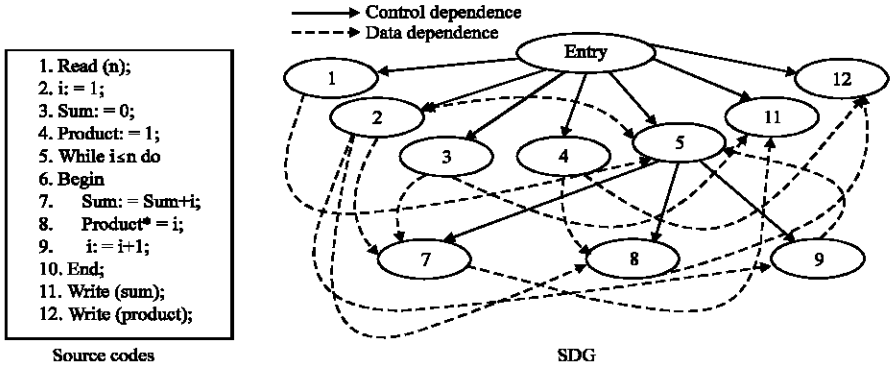

Fig. 2: The sample program and its system dependence graph

condensed co-change graph removes all the transaction vertices from the co-change graph and retains only the artifact vertices. E contains the edge $\{a_i, a_j\}$ if the artifacts a_i and a_j were commonly changed. The weight function w: E→I assigns to each edge an integer value, which is the times that two artifacts were commonly changed.

Figure 1b shows a condensed co-change graph of Fig. 1a. Only the artifact vertices a_1, a_2, a_3 and a_4 in Fig. 1a are remained. The edges connect two commonly change artifacts. In particular, the edge $\{a_2, a_3\}$ represents a_2, a_3 change together, the weight 2 of $\{a_2, a_3\}$ means a_2, a_3 commonly change twice.

System dependence graph: The System Dependence Graph (SDG) of a program is shown in Fig. 2. An SDG is a collection of Procedure Dependence Graphs (PDGs), one for each procedure. The vertices of a PDG represent the individual statements and predicates of the procedure. The edges of a PDG represent the control and data dependences among the procedure's statements and predicates. A vertex is control dependent on a predicate if the predicate controls whether or not the vertex will be executed. A vertex is the data dependent on an assignment if the value assigned can be referenced in the vertex.

APPROACH TO BUILD CONDENSED-CHANGE GRAPH

In this study, building a condensed co-change graph takes three steps.

- Constructing co-change graph from CVS
- Deriving the condensed co-change graph from co-change graph
- Merging the never changed artifacts into condensed co-change graph with a random walk

Co-change graph is directly extracted from CVS version repository file. Condensed co-change graph derived from the corresponding co-change graph. However, when establishing a condensed co-change graph, the relationship of never changed artifacts is unknown. If abandoning these never changed artifacts, incomplete commonly changed information may lead to bad clustering. Furthermore, available techniques are not sufficient to accurately evaluate the relationships of never changed artifacts. An eligible approach is therefore merging the never changed artifacts with a random walk. The merging process begins with a random artifact a_n, then randomly chooses another artifact, denoted as a_1, if the edge$<a_n, a_1>$exists, adds 1 to the weight. If the edge$<a_n, a_1>$ doesn't exists, then creates an edge between a_n and a_1 provided that a_n and a_1 are directly connected in SDG and assigns 1 to the weight. Repeat this process with a_1 as the start node. If a_n and a_1 have no edge in CVS or SDG, randomly selects a new start artifact and continues this process.

If C is the set of changed artifacts recorded in the CVS. G_c is the condensed co-change graph with the set of vertices C. U is the set of the never changed artifacts. e is the sum degree of G_c, G_n is the number of artifacts in C and u_n is the number of artifacts in U. The probability that the current artifact is from the C and the next artifact belongs to C as well is:

$$q_c = \frac{c_n - 1}{c_n + u_n - 1} \tag{1}$$

The probability that the current artifact from U and the next artifact from U either is:

$$q_u = \frac{u_n - 1}{c_n + u_n - 1} \tag{2}$$

Starting from a C artifact, after n-1 steps, the probability of the first U artifact coming is:

$$q_c^{n-1}\frac{u_n}{c_n + u_n - 1} = q_c^{n-1}(1 - q_c) \tag{3}$$

Starting from a C artifact, the expected number of steps required to reach a U artifact is:

$$E_c = \sum_{n=1}^{\infty} n q_c^{n-1}(1 - q_c) = \frac{1}{1 - q_c} \tag{4}$$

Analogous, the expected number of steps to reach a C artifact starting from U is:

$$E_u = \sum_{n=1}^{\infty} n q_u^{n-1}(1 - q_u) = \frac{1}{1 - q_u} \tag{5}$$

When two edges between C and U are added, $1+E_c$ edges to be added inside the C and $1+E_u$ edges to be added inside U. When m edges are added between C and U, the expected sum degree of the condensed co-change graph derived from U is:

$$\frac{m(1 + E_u)}{2} \tag{6}$$

The expected sum degree of C is:

$$\frac{2e + m(1 + E_c)}{2} \quad (7)$$

Because the merging process should achieve the goals that: (1) merge the never artifacts into condensed co-change graph as much as possible, (2) after the merging process ended, sum degree of G_c is greater than the sum degree of artifacts from U. If $U_n > C_n \Rightarrow E_u > E_c$, the added edges in U increase faster than the edges in G_c set;

$$\frac{m(1 + E_u)}{2} = \frac{2e + m(1 + E_c)}{2}$$

we obtain

$$m = \frac{2e}{E_u - E_c}$$

The merging process is ended when;

$$m = \frac{2e}{E_u - E_c}$$

or every artifact from U is connected. If $U_n \leq C_n \Rightarrow E_u \leq E_c$, the edges in U can't increase faster than the edges in G_c, when every artifact from U is connected, the merging process is ended. More formally, the merging process algorithm is described as follow:

Algorithm 1: Merging the never changed artifacts with a random walk

Input: Artifact set C and U
Output: The condensed co-change graph
Body:

```
1:    Randomly selects a artifact;
2:    Randomly chooses another artifact;
3:    If U_n > C_n
4:    Then
5:        Merging the never changed artifacts into condensed co-change
          graph with a
6:        Random walk until m edges have been added between C and U,
          or every artifact
7:        From U is connected;
8:    Else
9:        Merging the never changed artifacts into condensed co-change
          graph with a
10:       Random walk until every artifact from U is counected
```

EXTENDED GN CLUSTER DETECTION ALGORITHM

Based on the concept of edge-betweenness centrality, Newman and Girvan (2004) have proposed an algorithm (GN method) to determine undirected graph clusters, without any restriction on the number of clusters and cluster size. Guanqun *et al.* (2008) draws the conclusion that the clustering method based on the GN is better than Bunch algorithm, especially for large scale software. Bunch algorithm belongs to the graph clustering techniques, with a significant relation to GN method (Meunier and Paugam-Moisy, 2006).

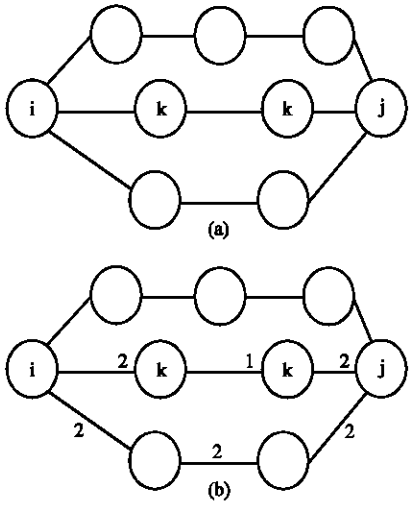

Fig. 3: Example of edge-betweenness centrality

If two nodes are connected, there might be several paths between them. Shortest paths are the smallest number of edges along the path. There might several shortest paths exist between two nodes. For a given edge {k, k'}, the number of shortest paths between two other nodes i and j that go through {k, k'}, denoted as $\sigma_{ij}\{k, k'\}$, divided by the total number of shortest paths that go from i to j, denoted as σ_{ij}. The edge-betweenness centrality $E_b(\{k, k'\})$ of {k, k'}, is the sum of all different pairs of nodes i and j in the graph:

$$E_b(\{k,k'\}) = \sum_{i \neq j} \frac{\sigma_{ij}(\{k,k'\})}{\sigma_{ij}} \quad \text{with } i \neq k \quad \text{and} \quad j \neq k' \quad (8)$$

As show in Fig. 3a, there are two shortest paths exist between i and j and one goes through edge {k, k'}, thus $\sigma_{ij}\{k, k'\} = 1$, $\sigma_{ij} = 2$.

Most of the shortest paths go through the few edges which connect the clusters. These edges comparably have big edge-betweenness centrality. Based on this property, GN method iteratively removing the edges with highest edge betweenness centrality, the clusters will be separated one by one.

Like merging unchanged artifacts into condensed co-change graph, m is an ending measure. Newman and Girvan introduce a measure called modularity to determine when the algorithm stops the cluster building. Modularity is defined as:

$$M(Q) = \sum_{q=0}^{Nq} \left[\frac{d_q}{L} - \left(\frac{l_q}{L} \right)^2 \right] \quad (9)$$

where Nq is the total number of clusters in a given set Q, d_q is the number of edges between nodes of a given cluster q, l_q is the total degree (i.e., the total number of edges) of nodes in cluster q and L the total number of edges in the whole network. The modularity is based on the edges in the initial network, without edge removal. This measure equals 1 if all the edges are inside the clusters of Q and 0 if there is an equal proportion between edges inside and outside the clusters of Q. The optimal set of clusters is the one with the highest modularity during the cluster building. MQ of a particular partition is high when the number of edges within the clusters is much larger than expected for a random partition.

If the software is well designed, the density of intra-cluster edges is much higher than that of inter-cluster edges (Li *et al*., 2004). With this reasoning, the software clustering problem is converted to find a particular partition for which MQ is the global maximum of modularity (Chen *et al*., 2004; Merali and McKelvey, 2006).

This study extends the cluster formation process by redefining shortest path. Shortest paths are the smallest number of edges and the biggest sum weights along the path. As shown in Fig. 3b, there are two paths with smallest number of edges exist between i and j and only one path has biggest sum weights, so the number of shortest paths is one. This shortest path goes through edge {k, k'}, thus $\sigma_{ij}\{k, k'\} = 1$, $\sigma_{ij} = 1$. More formally, GN method is described as follow:

Step 1: Calculate betweenness for all edges in the network
Step 2: Find the edge with the highest score and remove it from the network. If two or more edges tie for the highest score, choose one of them at random and remove it
Step 3: Calculate MQ of the current partition
Step 4: Recalculate edge betweenness for all remaining edges
Step 5: Repeat step 2~4 until no edge remains
Step 6: Choose the partition with maximum MQ

RESULTS AND DISCUSSION

In this study, the experiment is designed as follow:

- Constructing condensed co-change graph
- Clustering condensed co-change graph using GN algorithm provided by NetDraw, with maximum MQ as our clustering result
- Watermarking the biggest cluster by SANDMARK and comparing the robust with watermarking which aims at the whole program

Table 1: Experiment applications and the number of classes

No. of the application	Application name	SD
A1	jdrill2_3_1	18
A2	XMLTree	27
A3	Toy_1.4	124
A4	TreeView-1.1.3	303

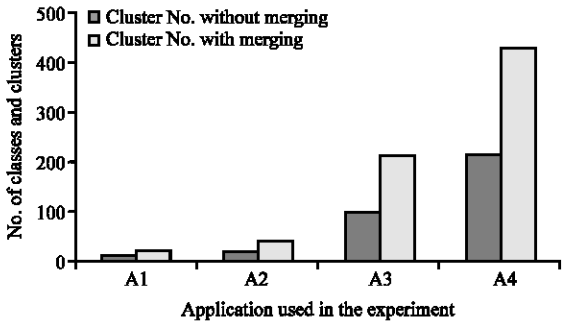

Fig. 4: The result of clustering co-change graph

Four Java applications which are frequently used in SANDMARK are adopted. The application name and the number of classes are shown in Table 1.

As show earlier, random walk can merge the unchanged artifacts completely on the condition that $U_n \leq C_n$. For the artifacts with bigger degree would have more commonly change possibilities., the experiment selects 50% artifacts with bigger degree to construct the condensed co-changed graph G_c. The weight of an edge$<a_i,a_j>$ is the number of edges connecting artifact a_i with artifact a_j in SDG. Each edge whatever represents the control dependence or data dependence has same commonly change possibilities. The rest of artifacts belong to unchanged artifacts set U.

The GN algorithm is provided by NetDraw and clustering condensed co-change graph with maximum MQ as our clustering result. Clustering result is shown in Fig. 4. From Fig. 4, we can see that, for each application, applying merging process can produce more clusters than without it.

Two factors affect the clustering number of condensed co-change graph. First, random walk merged unchanged artifacts into the condensed co-change graph, results in the condensed co-change graph has more vertices and edges. Second, the extended GN algorithm redefining the shortest paths which are the smallest number of edges and the biggest sum weights along the path. When two or more shortest paths exist between two nodes, the edge-betweenness centrality of each edge is cut down. More clusters with low edge-betweenness centrality are produced.

Ten different watermarking techniques are used to evaluate the robust of watermarking which embedding in

Table 2: Watermarking techniques used in the experiment

Watermarking symbol	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Watermarking technique	String constant	Stern	Register type	Qu/ Potkonjak	Mondern	Graph theoretic watermarking	Add meth field	Add switch	Add Initialization	Add expression

Table 3: Results of watermarking the whole applications and extracting the watermarks from the biggest cluster codes

	T									
A	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
A1	1/0	1/0	1/0	1/1	1/0	0/0	1/0	1/1	1/0	1/1
A2	1/0	1/0	1/0	1/1	1/1	1/0	1/0	1/0	1/0	1/1
A3	1/0	1/0	1/0	1/1	1/1	0/0	1/0	1/1	1/0	1/0
A4	1/0	1/0	1/0	1/1	1/1	1/0	1/0	1/0	1/0	1/1

T: Watermarking technique, A: Watermarking application

Table 4: Results of watermarking and extracting the watermarks from the biggest cluster codes

	T									
A	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
A1	1/0	0/0	1/1	1/1	1/1	0/0	1/1	1/1	1/0	1/1
A2	1/0	0/0	1/1	1/1	1/1	1/1	1/1	1/1	1/0	1/1
A3	1/0	0/0	1/1	1/1	1/1	0/0	1/1	1/1	1/0	1/1
A4	1/0	0/0	1/1	1/1	1/1	1/1	1/1	1/1	1/0	1/1

T: Watermarking technique, A: Watermarking application

the biggest cluster not the whole software. These watermarking techniques are shown in Table 2 and were implemented in SANDMARK.

Table 3 shows the results that embedding the watermarks in the whole applications and extracting the watermarks from the biggest cluster codes. Results are showed in binary measures, the values in the left of symbol "/" represent watermark embedding results. 0 means fail to embed the watermarks, 1 means success to embed the watermarks. Values in the right of symbol "/" represent watermark extracting results. 0 means fail, 1 means success. Almost all the watermarking techniques can embed the watermarks in the selected applications, except A1 and A3 with Graph Theoretic Watermarking, but only the watermarks embedded with W4, W5, W8, W10 work in the biggest cluster codes. That is to say if an attacker copies the biggest cluster codes, only W4, W5, W8, W10 techniques can protect it.

Table 4 shows the results that embedding and extracting the watermarks in the biggest cluster codes. We can see that if the watermarks are embedded in the biggest cluster codes, almost all the watermarking techniques can work in it except W1, W2 and W9. W1 embeds the watermarks in the String constant, can't be extracted from the code. W2 constructs the watermarks on the instruction frequency, needs sufficient instructions. Appling extended GN algorithm in condensed co-change graph, more clusters with little codes are produced. Leading to few cluster fit for W2. W9 adds initializations to form the watermarks, like W1, can't extract the watermarks from the cluster codes. Watermarking cluster is practical for protecting subsystems.

CONCLUSION

In most case, an attacker only plagiarizes a portion of software which is a specific subsystem. In this situation, the watermarks embedded in the whole software meet a cut attack. Most watermarking techniques fail to withstand it. In this study, we find that hiding the watermark inside cluster codes which are similar to the subsystem makes the watermarking more practical. The research presents a technique used to merge the unchanged artifacts into condensed co-change graph. Complete information makes clustering more accurate. An extended GN clustering approach is used to cluster the software condensed co-change graph. The clustering result shows that, applying extended GN approach to condensed co-change graph can get good partition. At last, the biggest cluster codes are watermarked in SANDMARK. By comparing the extracting result we can conclude that watermarking each cluster codes can withstand part codes stolen, it's more robust than watermarking the whole software.

ACKNOWLEDGMENTS

This study was supported by National Natural Science Foundation of China (60736016, 60873198, 60973128 and 60973113) and National Basic Research Program of China (2006CB303000, 2009CB326202, 2010CB334706).

REFERENCES

Beyer, D. and A. Noack, 2005a. Clustering software artifacts based on frequent common changes. Proceedings of the 13th International Workshop on Program Comprehension, (IWPC '05), IEEE Computer Society, Washington, DC, USA., pp: 259-268.

Beyer, D. and A. Noack, 2005b. Mining co-change clusters from version repositories. Technical Report IC/2005/003, Ecole Polytechnique F'ed'erale de Lausanne (EPFL). http://infoscience.epfl.ch/record/52706/files/IC_TECH_REPORT_2005003.pdf.

Chen, X., B. Francia, M. Li, B. McKinnon and A. Seker, 2004. Shared informationand program plagiarism detection. IEEE Trans. Inform. Theory, 50: 1545-1551.

Christopher, M., 2003. Software systems as complex networks: Structure, function and evolvability of software collaboration graphs. Phys. Rev. E-Stat., 68: 046116-046116.

Collberg, C., A. Huntwork, E. Carter, G. Townsend and M. Stepp, 2009. More on graph theoretic software watermarks: Implementation, analysis and attacks. Inform. Software Technol., 51: 56-67.

Dedrick, J., E. Carmel and K.L. Kraemer, 2010. A dynamic model of offshore software development. J. Inform. Technol., 10.1057/jit.2009.23

Gall, H., M. Jazayeri and J. Krajewski, 2003. CVS release history data for detecting logical couplings. Proceedings of the 6th International Workshop on Principles of Software Evolution, (IWPSE'03), Technical University of Vienna, German, pp: 84-94.

Guanqun, Q., Z. Lin and Z. Li, 2008. Applying complex network method to software clustering. Proceedings of the 2008 International Conference on Computer Science and Software Engineering-Volume 02, Dec. 12-14, IEEE Computer Society Washington, DC, USA., pp: 310-316.

Kamel, I. and Q. Albluwi, 2009. A robust software watermarking for copyright protection. Comput. Secur., 28: 395-409.

Li, M., X. Chen, X. Li, B. Ma and P.M.B. Vitanyi, 2004. The similarity metric. IEEE Trans. Inform. Theory, 50: 3250-3264.

Merali, Y. and B. McKelvey, 2006. Using complexity science to effect a paradigm shift in information systems for the 21st century. J. Inform. Technol., 21: 211-215.

Meunier, D. and H. Paugam-Moisy, 2006. Cluster detection algorithm in neural networks. Proceedings of 14th European Symposium on Articial Neural Networks. April 26-28, Bruges, Belgium, pp: 19-24.

Myles, G. and C. Collberg, 2006. Software Watermarking Through Register Allocation: Implementation, Analysis and Attacks. In: Digital Rights Management: Technologies, Issues Challenges and Systems, Safavi-Naini, R. and M. Yung (Eds.). Springer Verlag, Berlin, Heidelberg, pp: 180-191.

Newman, M.E.J. and M. Girvan, 2004. Finding and evaluating community structure in networks. Phys. Rev. E-Stat., 69: 026113-1-026113-15.

Periklis, A. and T. Vassilios, 2003. Software clustering based on information loss minimization. Proceedings of the Working Conference, Reverse Engineering. Nov. 13-16, Victoria, B.C., Canada, pp: 334-344.

Periklis, A. and T. Vassilios, 2005. Information-theoretic software clustering. IEEE Trans. Software Eng., 31: 150-165.

Wen, Z. and V. Tzerpos, 2004. An effectiveness measure for software clustering. Proceedings of the 12th IEEE International Workshop on Program Comprehension, June 24-26, IEEE Computer Society, Bari, Italy pp: 194-203.

Whelan, E., 2006. Exploring knowledge exchange in electronic networks of practice. J. Inform. Technol., 19: 5-12.

Zimmermann, T., S. Diehl and A. Zeller, 2003. How history justifies system architecture (or not). Proceedings of the 6th International Workshop on Principles of Software Evolution, Sept. 1-2, IEEE Computer Society Washington, DC, USA., pp: 73-83.

Zimmermann, T., P. Weigerber, S. Diehl and A. Zeller, 2004. Mining version histories to guide software changes. Proceedings of the 26th International Conference on Software Engineering, May 23-28, IEEE Computer Society Washington, DC, USA., pp: 563-572.