Global Journal of Computer Science and Technology
Software & Data Engineering
Volume 13 Issue 4 Version 1.0 Year 2013
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

An Empirical Investigation for Understanding & Extraction of Services from Monolithic Legacy Software

By Asfa Praveen & Shamimul Qamar
Shri Venkateshwara University

Abstract - While working on modernization of large monolithic application; speed , synchronization and interaction with other components are the major concern for practical implementation of target system; as Service-Oriented Computing extends and covering many sections of monolithic legacy to web oriented development, these aspects becoming a new challenges to existing software engineering practices, the paper presents work which is undertaken for service orientation of monolithic legacy application including initial steps of service understanding, comprehension and extraction so that it can take a part in further migration activities to service oriented architecture platform. The work also shows that how several useful techniques can be applied to accomplish the result.

Keywords : web services, ADT, SOA, clusters, comprehension.

GJCST-C Classification : D.2.11

AN EMPIRICAL INVESTIGATION FOR UNDERSTANDINGEXTRACTION OF SERVICES FROM MONOLITHICLEGACY SOFTWARE

Strictly as per the compliance and regulations of:

An Empirical Investigation for Understanding & Extraction of Services from Monolithic Legacy Software

Asfa Praveen [α] & Shamimul Qamar [σ]

Abstract - While working on modernization of large monolithic application; speed , synchronization and interaction with other components are the major concern for practical implementation of target system; as Service-Oriented Computing extends and covering many sections of monolithic legacy to web oriented development, these aspects becoming a new challenges to existing software engineering practices, the paper presents work which is undertaken for service orientation of monolithic legacy application including initial steps of service understanding, comprehension and extraction so that it can take a part in further migration activities to service oriented architecture platform. The work also shows that how several useful techniques can be applied to accomplish the result.

Keywords : web-services, ADT, SOA, clusters, comprehension.

I. Introduction

A difficult and complex procedure for any maintenance project is the initial investigation which includes understanding of programs of software with its source code. This research is undertaken for service orientation of monolithic legacy software, till now many formal understanding and comprehension methods have been presented but conceptually and practically differ from one investigator to the other. Easy and quick monolithic legacy program understanding with fast comprehension is major concern of the work which plays a very important and crucial role in the planning, designing, feasibility study and cost estimation for services orientation projects of monolithic legacy [1]. The empirical examples/case studies have been presented to explain how the processes can be used to support better and improved comprehension in the program and incorporated services. The role of Ha-Slicer tool and web-mining techniques have been presented with application that appear to be reasonable for manual and automatically grouping extraction of services semantically similar in monolithic software and components [2]. The clusters of services understood, extracted by these processes represent an abstraction of the program source code based on a semantic similarities which should be incorporated further to high-level design of target system.

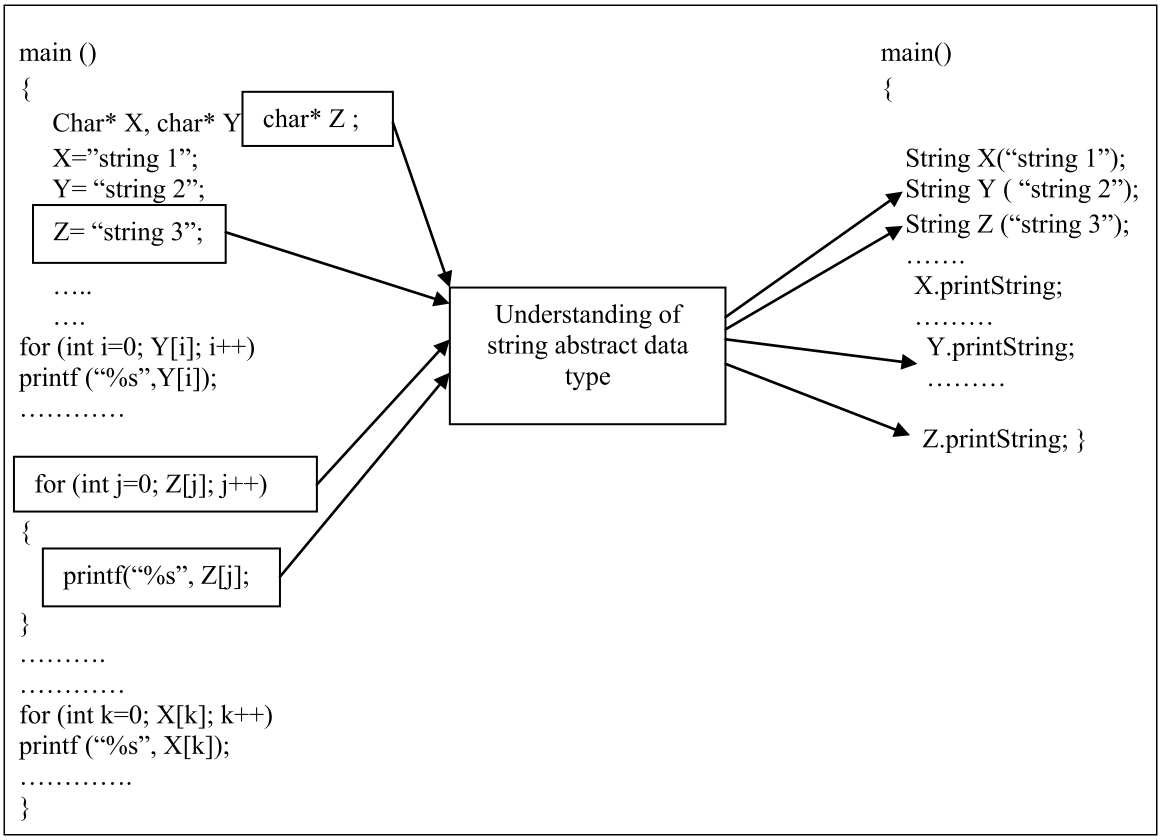
II. Problems for Understanding of Program

This section shows a sample program as presented in fig. 1, an analyzed program is depicted in the left hand side of the fig. 2, contains declarations, initializations and embedded print loop for each of three strings. The strings are considered as primitive data type for this illustration with no shared functionality for printing. To understand this program the library of program plan has to be considered, which has previously compiled knowledge for composition of program in this domain as shown in fig. 3; fig. 3 shows the library plan which contains the program plan [3] that contains the class string or abstract data type. The understanding of services and translation for source code with including singly abstract data type can be performed if the mapping can take place between original source and complied knowledge in the domain of services which is shown in fig. 2.

Author α : Ph.D. (Computer Sc.) Research Scholar, Faculty of Science & Technology, Shri Venkateshwara University, Gajraula, (U.P.), India. E-mail : asfa_praveen@yahoo.com
Author σ : Professor of Electronics & Computer Engineering, Noida Institute of Engineering & Technology, Greater Noida, (U.P.), India. E-mail : jsqamar@gmail.com

```
Class String {
                    char locStr [SIZE]
                    String( char* intStr )
                            for (int a=0; intStr [a]; a+ +)
                                    locStr[a] = intStr[a]; }
                    printString () {
                                    for ( int a=0; locStr[a]; a++);
                                    printf("%s", locStr[a];}}
```

Figure 1 : Sample Program



```
main ()
{
   Char* X, char* Y   char* Z ;
   X="string 1";
   Y= "string 2";
   Z= "string 3";
   …..
   ….
for (int i=0; Y[i]; i++)
printf ("%s",Y[i]);
…………
 for (int j=0; Z[j]; j++)
{
   printf("%s", Z[j];
}
……….
…………
for (int k=0; X[k]; k++)
printf ("%s", X[k]);
………….
}
```

```
main()
{
  String X("string 1");
  String Y ( "string 2");
  String Z ("string 3");
  …….
   X.printString;
   ………
    Y.printString;
    ………
    Z.printString; }
```

Figure 2 : Presentation of understating of mappings of C code abstract data type in view of object code [3]

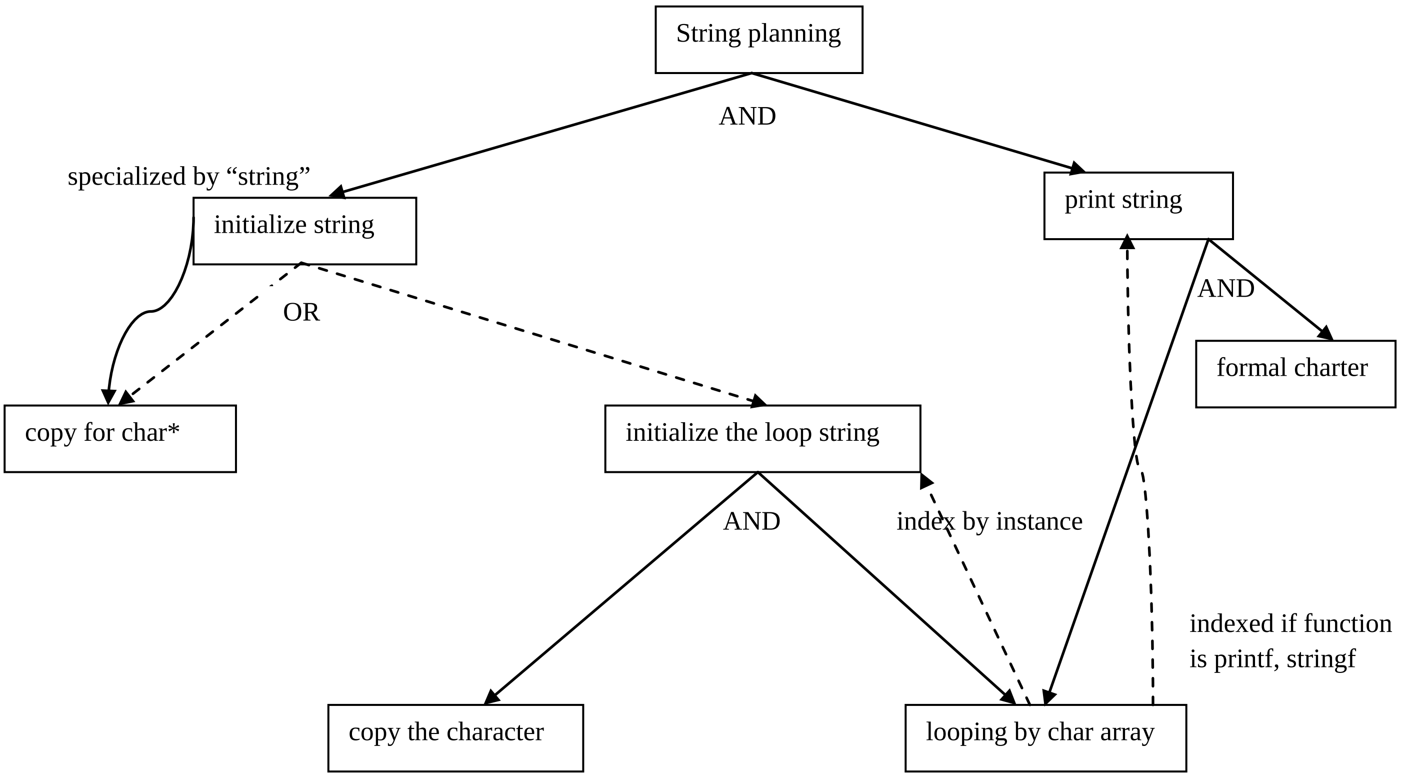


Figure 3 : String ADT within hierarchical plan program library [3]

Code in the left hand side of fig. 2 is given to explain the problem of understanding for program plan with the known context of as in the sting abstract data types. Identification of services and duplicate code may result in one time instance and incorporation of abstract data types. Abstract data types functionalities has been implemented same types in the right hand side of the fig. 2 in the object code the same concept has also been implemented in any other service oriented frameworks.

Suggested solution of the problems of understanding process is proposed in two phases, (1) investigate all instances of abstract data type in the source code with intention to convert them in services by abstract program features, (2) identify services plan blocks with program slices will relate to assure the hierarchical structure specified in the program plan based on knowledge. This can resolve the problem of knowledge management. Some useful advantages of identification are applied in the mapping of source code to the target service comprehension plan as when planning for replacing the source code by service oriented code resulting code will contain less code with same functionality and abstract data type and size for running, saving will be reduced, that will helpful for further implementation tasks. Mapping of the source code to the services is the main elementary blocks for the service oriented plan [4] and establishing plan library for either translation of code or identification of knowledge; it will reduce the bigger task of understanding.

III. Applications of Slicing, Haslicer Tools for Service Identification

Slicing based on Functional Dependence Graphs (FDG) contains five phases as illustrated in fig. 4 [5] the study conducted by Nuno et.al. The first phase parses the source code and originates the Abstract Syntax Tree instance t, which is followed by an abstraction procedure that extracts the useful information from *t* for constructing a FDG instance g with estimating the different types of nodes. Actual slicing is performed in the third phase, a slicing standard is composed here by a node of *t* and a specific slicing algorithm, the original FDG g is sliced, generating a sub-graph of g that is g'. The slicing takes place over the FDG to make the result which is always a sub-graph of the original graph.]

The fourth phase performs cutting AST t, based on the sliced graph g. At this point, each program entity that is not present in graph g*'*, is used to clip the correspondent syntactic entity in *t*, giving origin to a sub-tree t' of t. Finally, code reconstruction takes place, where the clipped tree *t'* is consumed to generate the sliced program by a reverse process of phase one.

Global Journal of Computer Science and Technology (C) Volume XIII Issue IV Version I Year 2013

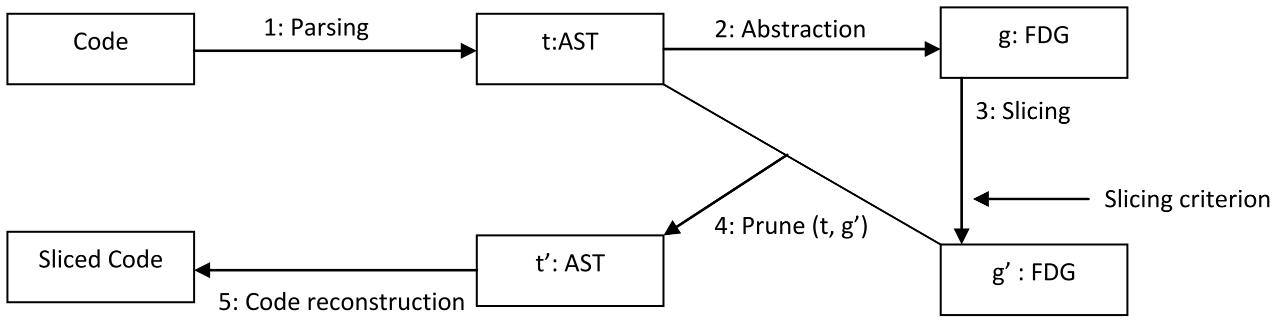


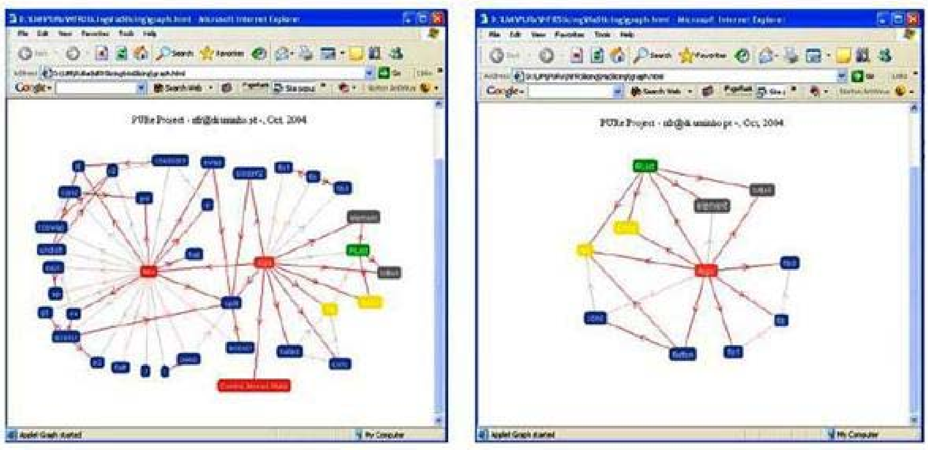
Figure 4 : Slicing Process [5]

Haslicer [6] is a sample tool for slicing for monolithic programs as sample here entirely written in Haskell language that will cover forward, backward and forward dependency slicing. The samples are sliced by implementing the slicing [7] and mention some other problems which are fundamentals to service component identification as (1) the definition of the extraction process from source code and (2) the incorporation of a visual interface by the generated FDG to support programmer interaction. It is accepting now only Haskell code [8] but can be plug-in for other monolithic code written in functional languages including purely functional language.

Fig.5 shows two snapshots of the sample working over a Haskell program [9]. Screenshot 5(a) shows the visualization of the entire FDG loaded in the tool. Notice that the differently colored nodes indicate different program entity types according to Table 1. Fig. 5(a) reproduces the sub-graph resulted from performing slice over one of the nodes of the graph from fig. 5(b). Once a slice has been computed, the user may retrieve the corresponding code. The whole process can be canceled or started again with different criteria.

Node Color	Node Type
🟥	N_m
🟦	N_f
🟩	N_{dt}
🟨	N_c
⬛	N_d

Table 1 : FDG Edge Code



(a) (b)

Figure 5 : Slicing Process with HaSlicer [6]

There are basically two ways in which slicing techniques and the HaSlicer tool can be used in the process of service identification; either as a support procedure for manual identification or as a discovery procedure in which the whole system is searched for possible location of services. In this section both approaches are applied and briefly discussed, the first approach applied for manual service identification guided by analyzing and slicing some representation of the legacy code. In this context, the FDG seems to provide applicable representation model. Through its analysis, the program designer can identify all the dependencies between the code entities and look for certain architectural patterns or undesired dependencies in the graph. The process starts in a top-down way, looking for the top level functions that characterize the desired service, once these functions are found, forward dependency slicing is applied, starting from the corresponding FDG nodes. It produces a series of sliced code files, that have to be merged together in order to build the desired services. Forward dependency slice collects all the program entities in which each top level function requires to operate correctly. Thus, by merging all the forward dependency slices corresponding to a particular service one gets the least derived program that implements it.

The second approach uses slicing mentioned in the beginning of this section under the name of component discovery relies on slicing techniques for the automatic isolation of possible components. As per experience this was found particularly useful at early stages of service identification. Such procedures, however, must be used carefully, since they may lead to the identification of both false positives and false negatives. This means that there might be good candidates for services which are not found as well as situations in which several possible services are identified which turn out to lack any practical or operational concern. To use an automatic service discovery procedure, one must first understand what to look for, since there is no universal way of stating which characteristics correspond to a possible software code component. Thus, process has to look for services by indirect means, which certainly include the identification of certain characteristics that components usually bear, but also some filtering criteria. Therefore a possible criterion for service discovery is based on the data types defined on the original code. The idea is to take each data type and isolate both data types and every program entity in the system that depends on it. Such an operation can be accomplished by performing a backward slicing starting from each data type node in the FDG.

Another identified feature for service-orientation task relates to the fact that interesting services typically present a high level of coupling and a high level of cohesion [10]. Coupling is a measure to estimate how mutually dependable two components services are, so it tries to measure how much a change in one service component affects other service components in a system whereas cohesion estimates how the functions as shown below [5] of a specific component which are internally related.

$$Coupling(G, f) \triangleq \sharp\{(x, y) \mid \exists x, y.\ yGx \wedge x \in f \wedge y \notin f\} \quad (1)$$

$$Cohesion(G, f) \triangleq \sharp\{(x, y) \mid \exists x, y.\ yGx \wedge x \in f \wedge y \in f\} \quad (2)$$

$$CCAnalysis(G) \triangleq \{(Coupling(G, f), Cohesion(G, f)) \mid \forall f \in \mathcal{P}F\} \quad (3)$$

In a service component with a low cohesion degree errors and undesirable behavior are difficult to detect. In practice if its functions are weakly related, errors may hide themselves in hardly ever used areas and remain unseen to testing for some time. The conjunction of these two measurement leads to discovery criteria, which uses the FDG to look for specific clusters of functions that is sets of strongly related functions, with reduced dependencies on any other program entity outside this set. Such function clusters cannot be identified by program slicing techniques, but the FDG is still very useful in determining this clusters. The HaSlicer tools compute the combined value where *G* is a FDG and *F* a set of functions under inspection. This is presented in study conducted by Nuno et.al. [5], which is depending on how easily or hardly service component discovery criteria are; then different acceptance limits for coupling and cohesion can be used. This will explain what clusters will be considered as location of prospective service components. Once such clusters are identified, the process continues by applying forward dependency slicing on every function in the cluster and merging the resultant code.

IV. Clustering Source Code Component's Services and Documents

Clustering for services of source code is based on semantic and structural information which is useful in the understanding and comprehension of monolithic software systems, on the other hand clustering can be used to support in re-modularization of systems and the identification of services from abstract data types [11]. If the program is to be reengineered for a service-oriented platform from a monolithic program this type of clustering would be very useful. The purpose is to decrease the quantity of source code when an engineer wants to observe at once and guess about possible relationships with the system not obvious from the current organization's documentation.

The method used here focuses on using reports generated by conceptual approaches, for this case a vector represented by latent semantic indexing [12] is generated to compare services and classify them into clusters of semantically similar concepts and for huge program it can be partitioned into a group of only source code documents by which the features for each document are prepared. Program documents are divided semantically based on similarities for connecting other documents to cluster the source code. For this purpose there are many applied clustering algorithms: construction, optimization, hierarchical and graph theoretical algorithms. There are also several other algorithms that use notion of hybrid concept applied for different classification for specific problems. The framework here proposes graph theoretic approach although numbers of other types of clustering algorithms have been used to cluster software program.

To cluster the documents minimal spanning tree algorithm [13] can be is used if the document is similar to at least some documents in the cluster then it is added to cluster, this can give an opportunity to group together the documents of similar type. The measurement of similarity is calculated by the cosine of the two vector representations of the source code documents. The similarity values has [-1, 1] for a domain, with the value 1 being closely similar. Non required symbols can be removed by using simple parsing of the source code that can break the source into the proper way. Comment delimiters and syntactical reserved words are removed; they had to add little or no semantic knowledge for problem domain [14]. On the other hand information retrieval process will analyses such confusing words such as semi-colons with a completely non-selective characteristic between source code and service components. So the variation with this characteristic is very little like zero thus; if two components have a semi-colon then not sure about their similarity. For the uses of latent semantic indexing on natural language perspectives a paragraph or code section is used for document because sentences are to be small and chapters too large. Source code that has similar concepts are: function, structure, module, file, class, etc. Observably the statement granularity is very low and file containing many functions can be too big.

V. Applying Web-Mining Techniques to Understand Services

So many techniques have been developed in Web-mining for successfully analyze the structure of web-services [16]. These techniques undertake the internet based web as a large graph which is based on hyperlink structure to identify the intended web pages. This section presents the study shows the application of web mining techniques, how to apply them to trace and understand classes and web services. HITS web-mining algorithm [15] is suggested to identify hubs and authorities for the web services. The HITS algorithm can be combined with the compressed call graph. The classes which are related with excellent "hubs" in the compressed call graph are good candidates for introduction of aspects.

a) Identifying Hubs and Authority in Big Web-Graphs

The concepts of "hub" and "authority" are introduced by HITS web-mining algorithm [15], hubs are pages that refer to pages containing information rather than being enlightening themselves, for examples web directories, lists of personal pages etc. and a page is called an authority if it has useful information. Thus, a web-page is a good hub if it is providing useful information. A page can be called as good authority if it is used by many good hubs. The HITS algorithm is based on this relation between hubs and authorities. This example considers the web-graph shown in fig. 6.

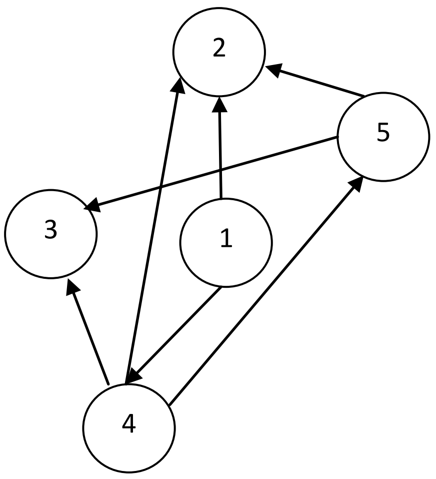


Figure 6 : Example Web-Graph

In this graph, 2 and 3 are good authorities, and 4 and 5 are good hubs, and 1 is a less good hub. The authority of 2 is larger than the authority of 3, because the only in-links that do not have in common are 1 2 and 2 3, and 1 is a better hub than 2. 4 and 5 are better hubs than 1, as they point to better authorities. The HITS algorithm works as follows: Every page i get assigned to it two numbers; ai denotes the authority of the page, while hi denotes the hubiness. Let i j denote that there is a hyperlink from page i to page j. The recursive relation between authority and hubiness is confined by the following formula.

$$h_i = \sum_{i \to j} a_j \qquad (1)$$

$$a_j = \sum_{i \to j} h_i \qquad (2)$$

The HITS algorithm starts with initializing all h's and a's to 1, and repeatedly updates the values for all pages, using the formula (1) and (2). If after each update the values are normalized, this process converges to stable sets of authority and hub weights [15]. It is also possible to add weights to the edges in the graph. Adding weights to the graph can be interesting to capture the fact that some edges are more important than others. This extension only requires a small modification to the update rules. Let w [i, j] be the weight of the edge from page i to page j. The update rules become

$$h_i = \Sigma_{i \to j}\, \omega[i, j].a_j$$

and

$$a_j = \Sigma_{i \to}\, \omega[i, j].h_i$$

Example: For given graph, the hub and authority weights to the following values:

$h_1 = 64$	$a_1 = 0$
$h_2 = 48$	$a_2 = 100$
$h_3 = 0$	$a_3 = 94$
$h_4 = 100$	$a_4 = 24$
$h_5 = 100$	$a_5 = 0$

In the context of web-mining, the identification of hubs and authorities by the HITS algorithm has turned out to be very useful. Because HITS only uses the links between web-pages then can be used in services [15].

VI. Service Extraction Process

The initial three steps of the service extraction process [18] as shown in fig. 7 represent the candidate service identification phase; candidate service identification is a challenging job, so a step-wise identification approach is designed. (a) Initially, the research finds how to utilize architectural reconstruction and source code visualization techniques. This step facilitates the proper understanding of code and to obtain structural properties of the source code [17]. The source code visualization technique presented by Geet et.al.[19] appears to be a good starting point. (b) The next step is to identify the design patterns [Gamma, 1995], one of the largely studied and applied techniques in context of reverse engineering. This is the extension for design pattern detection and its applicability in legacy to service oriented migration. (c) In the last step, linguistic analysis techniques are used [20] and concept analysis is used to find appropriate concepts that have been applied in the source code. The service extraction is performed after the application of concept slicing technique which can be further applied to extract the complete code generating the identified functionalities; it is fairly used to extract from source code various useful features for program comprehension [20]. It can independently extract from source code with the help of code query method. [21], that helps to extract abstract data type and common concern features, this extraction maps source code to service composition, the language features then supports the fixing composition related issues, in order to build new services. The main advantage of extraction is to generate services by component effective approaches then compose them to achieve target system implementation. With all these steps, process has achieved a simplified identification of candidate services in the monolithic code.

a) Service Understanding and Extraction Guidelines

There are following guidelines [22] for extraction and understanding of services;

i. *Realistic Representation*

Any service understood from the code must present real functional state in the program. This is the most important criteria if this is having any conflict with any other then this should be given priority because initial investigation goes through the program which is more trustworthy asset than a documentation.

ii. *Multiple representations for different abstraction of hierarchy*

Different development teams require different representations of services, for example, a programmer would like to have web services represented as code segments, while developer may require different types of forms as decision table, tree or chart to get the logical structure; so it must be represented in a hierarchy oriented way. Service understanding is more complex if they are in various constraints based perspectives, such as legal, marketing and technology. It is hardly tuff task to trace services without some form of abstractions or decomposition of the program.

iii. *Domain Oriented Policies*

Services expressed in the domain specific environment are far better because it connects with domain concept and propagates path for easy application. Then many tools can be applied for identification of logic of algorithm, data structures and other program entities.

iv. *Human-Assisted Automation*

As monolithic programs are huge having a lot of difficulties so if not impossible it is devised to use semi-automatic tool for service understanding. The software maintainers prefer to have an interactive tool that allow them to extract services, simplify their representations, and provide linkage to the code, rather than providing a black-box tool that generate services code automatically.

v. *Maintenance Tool*

Understood services will be useful in other software reengineering activities. Understanding should be managed together with the monolithic software using the same tool as the mapping from any service to its corresponding code segments. This capability will permit the software engineers to focus on only those segments and functions of the software that is relevant to a particular service type.

VII. Conclusion

This paper focused on program understanding and extraction for service orientation of monolithic code and presented various applied methods, techniques as clustering, web-mining, slicing, reverse engineering, and some more used tools that facilitates the automated process as Ha-Slicer with their previous applications, also presented the results and procedures, which was started with the understanding of code and finally concluded on the extraction of services.

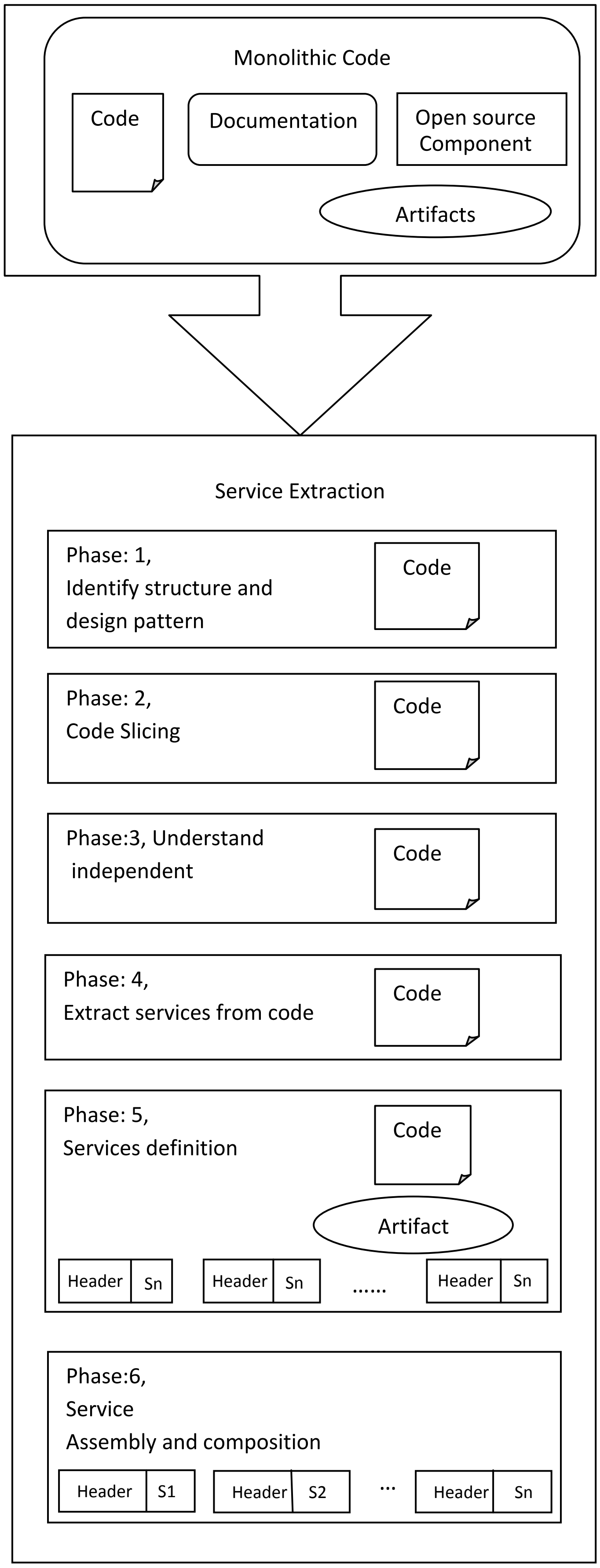


Figure 7 : Service Extraction Process

References Références Referencias

1. http://www.podcast.com/Technology/I-9445.htm
2. Shahanawaj Ahamad, "Web Centric Evolution of Legacy System", International Journal of Electrical and Computer Science, Vol: 10, No:1, pp: 19-24, 2010.
3. Steven Woods and Qiang Yang, "Program Understanding as Constraint Satisfaction", IEEE, Department of Computer Science, University of Waterloo, Canada, 1995.
4. http://technet.microsoft.com/en-us/magazine/ee677579.aspx
5. Nuno F. Rodrigues, Lu´ıs S. Barbosa, "Component Identification Through Program Slicing", in Electronic Notes in Theoretical Computer Science, Science Direct, Elsevier, pp: 291-304, 2006.
6. Nuno Miguel et.al, "Slicing Techniques Applied to Architectural Analysis of Legacy Software", report of Departamento de Informatica Escola de Engenharia, Universidade do Minho, 2008.
7. N. Rodrigues, "A basis for slicing functional programs". Technical report, PURe Project Report, DICCTC, U. Minho, 2005.
8. http://www.haskell.org/haskellwiki/Haskell
9. http://en.wikipedia.org/wiki/Haskell_(programming_language)
10. http://www.shu.ac.uk/softeng/extrabits/modularity/modularity%20-%20new%20version.doc
11. Adrian Kuhn et. al., "Semantic clustering: Identifying topics in source code", Information and Software Technology, Elsevier, pp: 230–243, 2007.
12. T. K. Landauer and S.T. Dumais, "A Solution to Plato's Problem: The Latent Semantic Analysis Theory of the Acquisition, Induction, and Representation of Knowledge", Psychological Review, vol. 104, no. 2, pp. 211-240, 1997.
13. J. B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", Proc. Amer. Math. Soc., vol.7, no. 1, pp. 48-50, 1956.
14. Jonathan I. Maletic et.al, Supporting program comprehension using semantic and structural information, Proceeding ICSE '01 Proceedings of the 23rd International Conference on Software Engineering Pp: 103-112, 2001.
15. J. M. Kleinberg, "Authoritative sources in a hyperlinked environment", Journal of the ACM, 46(5): 604–632, 1999.
16. D. Gibson, J. M. Kleinberg, and P. Raghavan. "Inferring web communities from link topology", In UK Conference on Hypertext, pages 225–234, 1998.
17. http://www.erikvanveenendaal.nl/NL/files/e-book%20PRISMA.pdf
18. http://www.docstoc.com/docs/94425343/Journal-of-Computer-Science-and-Research-Vol-9-No-8-August-2011

19. J. Van Geet and S. Demeyer, "Lightweight visualisations of cobol code for supporting migration to SOA," in 3rd International ERCIM Symposium on Software Evolution, October, 2007.
20. N. och Dag, B. Regnell, V. Gervasi, and S. Brinkkemper, "A linguistic engineering approach to large-scale requirements management," Software, IEEE, vol. 22, no. 1, pp. 32–39, 2005.
21. S. R. Tilley, D. B. Smith, and S. Paul, "Towards a framework for program understanding," in 4th International Workshop on Program Comprehension (WPC'96), 1996, pp. 19–28.
22. http://www.infosys.com/infosys-labs/publications/Documents/knowledge-engineering-management.pdf