IEICE TRANS. INF. & SYST., VOL.E99–D, NO.3 MARCH 2016

LETTER

# Adaptive Weighting of Structural Dependency and Textual Similarity in Software Architecture Recovery

**Jae-Chul UM**[†], **Ki-Seong LEE**[†], ***Nonmembers*, *and* Chan-Gun LEE**[†a)], ***Member***

**SUMMARY** Software architecture recovery techniques are often adopted to derive a module view of software from its source code in case software architecture documents are unavailable or outdated. The module view is one of the most important perspectives of software architecture. In this paper, we propose a novel approach to derive a module view by adaptively integrating structural dependency and textual similarity. Our approach utilizes Newman modularity and Shannon information entropy to determine the appropriate weights of the dependencies during the integration. We apply our approach to various open-source projects and show the experimental results validating the effectiveness of the approach.

***key words:*** *software architecture recovery, module view, dependency, Newman modularity, information entropy*

## 1. Introduction

Software consists of a set of modules which are the functional and structural units. Because a software module reflects the original developers' design purposes the analysis of software modules is considered an essential step to understand and maintain the software.

One of the most popular techniques used for software architecture's module view recovery (module view recovery hereafter) is software clustering. In this technique, various dependencies between modules are extracted and the most similar files/classes are grouped together. Nowadays various clustering methods and software dependencies for module view recovery are studied in the field of software reengineering. Due to its usefulness, the research on this topic has been actively pursued [12].

Recent studies show that the structural dependency and textual (also called semantic) similarity are useful in deriving module views [8], [10]. Unfortunately, most of the previous approaches are limited to considering either the structural dependency or the textual similarity exclusively or rely on ad-hoc schemes to integrate them. Note that the structural dependency alone may not be enough to figure out developers' full intentions regarding the modules and their relationships. It turns out that the textual similarity is very effective to supplement the structural information and captures different aspects of relationships between the software modules, thus it helps to derive a more accurate module view [10]. However, it should be noted that the quality of textual information varies significantly from one software project to another.

In this paper, we propose a novel approach to automatically derive the module views by adaptively utilizing both structural dependency and textual similarity. Our approach does not require ad-hoc settings for the weights of the structural and textual information. The proposed algorithm can determine the appropriate weights of the structural and the textual information by inspecting their quality.

The rest of our paper is composed as follows. Section 2 explains the sources of the dependencies and summarizes recent related works in this area. We propose our approach in Sect. 3 and present its rationale. Section 4 presents experimental results and analyzes the performance of the approach. Section 5 summaries the paper and presents the future work.

## 2. Related Work

The structural dependency [15] has been the most frequently utilized information for module view recovery in the literature. The structural dependency captures various code relationships such as references, inheritances, shared variables and parameters, and method calls.

The textual similarity [1] is extracted from the textual information represented in the source code such as comments, method names, class names, and variable names. The raw textual information is typically refined by the document analysis methods like LSI (Latent Semantic Indexing) and LDA (Latent Dirichlet Allocation).

Recently a few approaches trying to integrate multidimensional information for module view recovery are proposed. Patel's study [6] uses static analysis first to extract structural dependency and refines the dependency using the dynamic analysis. Garcia et al. [10] combine structural and textual properties to construct feature vectors of software entities. They demonstrated that such combinational dependencies can improve the performance of architectural recovery. Bavota et al. [8] also proposed combined use of structural and semantic dependencies with weights to derive module views. Their scheme is based on both structural and semantic coupling metrics, which are based on the number of method invocations and the cosine similarity between classes, respectively. Lutellier et al. [16] emphasized that quality of dependencies may affect the accuracy of the architecture recovery result.

We argue that none of the aforementioned approaches

Manuscript received July 22, 2015.
Manuscript revised November 9, 2015.
Manuscript publicized December 15, 2015.

[†]The authors are with School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea.
a) E-mail: cglee@cau.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2015EDL8160

provides an automatic scheme to determine the weights of the dependencies, which is essential in practice. We propose an adaptive method to combine the information by considering the quality of the dependencies.

## 3. Extraction and Integration of Dependency Information

Figure 1 illustrates our approach proposed for the adaptive integration of the structural dependency and the textual similarity. By utilizing the Newman modularity and the information entropy, our approach integrates both structural and textual information and calculates the complex dependency. A clustering algorithm accepts this complex dependency and generates the module views for the software. The details are explained in the following.

Software dependencies and similarities are typically represented as DSM (Design Structure Matrix). Depending on the clustering granularity, the entities of software can be classes, files, and packages, etc. In this paper, we consider each file as a software entity for clustering.

$$\mathrm{M_{str}} = \begin{array}{c|cccc} & \mathrm{F1} & \mathrm{F2} & \mathrm{F3} & \mathrm{F4} \\ \hline \mathrm{F1} & 0 & 1 & 1 & 0 \\ \mathrm{F2} & 0 & 0 & 1 & 0 \\ \mathrm{F3} & 0 & 0 & 0 & 1 \\ \mathrm{F4} & 0 & 1 & 0 & 0 \end{array} \qquad (1)$$

• **Structural Dependency**: For representing structural dependency in Fig. 1, we define a DSM called $\mathrm{M_{str}}$. $\mathrm{M_{str}}$ in (1) shows an example of DSM consisting of four entities F1, F2, F3, and F4. In case there exists a dependency from an entity $\mathrm{E_i}$ to $\mathrm{E_j}$, then the corresponding row i and column j will have value 1 or 0 otherwise. Typical factors selected in the literature contributing to the dependency include method calls, class inheritances, implementations of the same interface class, and object references.

• **Textual Similarity**: Textual similarity in Fig. 1 is extracted by following the process Fig. 2. It is a common process to calculate similarity between documents in the field of information retrieval, and a similar process is applied to software source code. Method names, variable names, class names, package names, comments, and etc. can contribute to software textual similarity. Stemming reduces words into their basic roots or stems so that the variations on a word can be handled effectively. Then, a term by document matrix is calculated through the tf-idf (term frequency-inverse document frequency) treatment. The matrix is further processed by LSA (Latent Semantic Analysis). LSA is adopted to reveal inherent relationships between entities and it uses SVD (Singular Value Decomposition) technique. Such latent semantic processing is essential in clustering textual similarity. For example, the modules corresponding to database system may contain seemingly different text information such as transaction, concurrency, SQL, and lock. The latent semantic processing can reveal their relationships and help to group them in a same cluster. In the end, by applying cosine similarity to source file vectors, textual similarity

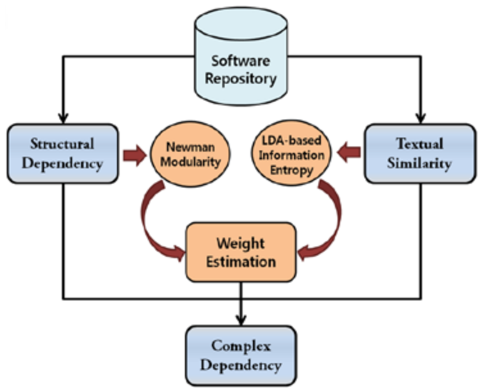

**Fig. 1** The proposed integration process

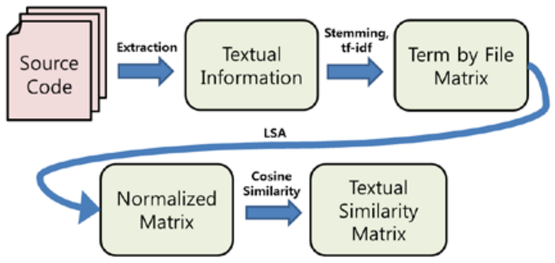

**Fig. 2** The extraction process of textual similarity

between all source files is extracted from the term-by-file matrix.

• **Newman Modularity**: Newman modularity [14] in Fig. 1 is calculated as shown in (2). In this paper, we rely on this metric to measure the modularity of the given structural dependency information.

$$Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(c_i, c_j) \qquad (2)$$

where $m$ is the number of all edges, $A_{ij}$ is the weight from node $i$ to $j$, $k_i$ is the number of edges linked to node $i$. $\delta(c_i, c_j)$ is set to 1 in case the two nodes $c_i$ and $c_j$ are clustered to the same group and 0 otherwise. In our case, $A$, node, and $k_i$ correspond to dependency matrix, file, and degree of $i$th file, respectively.

• **LDA-based Information Entropy**: For the LDA-based information entropy in Fig. 1, we adopt Shannon's information entropy to quantify its quality. As shown in (3) and (4), we utilize the topic model to compute the information entropy of each file. The topic model is based on a theory stating that some words in a document probabilistically represent the document itself and all words in the document are defined as random variables of the topics. Specifically, LDA was chosen to derive random variables based on the topic model. LDA selects the words following Poisson and Dirichlet distributions and calculates the conditional probability of the topics [7]. For the experiments, we use Mallet [13] implementing LDA.

$$q_{t_i}^{d_j} = \frac{p_{t_i}^{d_j}}{\sum_{k=1}^{n} p_{t_i}^{d_k}} \tag{3}$$

$$D(t_i) = \frac{\sum_{k=1}^{n} -q_{t_i}^{d_k} \times \log\left(q_{t_i}^{d_k}\right)}{\log n} \tag{4}$$

In the above, (3) and (4) are the information entropy of the topics defined in [18]. $p_{t_i}^{d_j}$ represents the document-topic probability distribution for the topic $i$ and document $j$. Similarly $q_{t_i}^{d_j}$ represents the topic-document probability distribution. $n$ is the number of documents. Note that the information entropy of the $i$-th topic $D(t_i)$ is normalized. We calculate the quality of the textual information as shown in (5) by considering all $D(t_i)$ where $N$ is the number of topics.

$$D = \frac{\sum_{i=1}^{N} D(t_i)}{N} \tag{5}$$

• **Weight Estimation and Complex Dependency**: Using both Newman modularity and LDA-based information entropy, we perform weight estimation presented in Fig. 1. In this step, we integrate the structural and textual information to determine a complex dependency. An Eq. (6) shows how to compute the complex dependency. A complex dependency matrix $M$ is consist of two part, a weighted $M_{str}$ and a weighted $M_{sem}$ where $M_{str}$ is a structural dependency matrix and $M_{sem}$ is a textual similarity matrix.

$$M = Q \times M_{str} + \min\left(1, \frac{D}{Q}\right) \times M_{sem} \tag{6}$$

In (6), $Q$ is Newman modularity and $D$ is quality of the textual information. Note that, in weighted $M_{sem}$, we take the minimum of one and the ratio of $D$ to $Q$ to limit the maximum weight for the textual information to one.

## 4. Experiments and Evaluation

Our implementation uses Blondel's graph clustering [17], Bunch [4], and BorderFlow graph clustering [2] to derive module views of the software automatically. We have chosen the Blondel's graph clustering due to its popularity and effectiveness in community detection for large scale graph. The Bunch clustering algorithm derives software decompositions with the maximum cohesion and the minimum coupling by utilizing the MQ [4]. In numerous past studies on software architecture recovery, the Bunch was one of the most frequently cited tools. The BorderFlow graph clustering has been successfully applied in the area of concept location in recent studies. None of these three algorithms needs to specify the number of clusters to be found. In addition, the fact that their implementations are all available to public would facilitate the reproduction of our experimental results [3], [5], [9].

In order to evaluate the module views from those clustering algorithms, we utilize recently published ground-truth data and a metric designed to compare two module views. Garcia et al. [12] proposed a systematic process for recovering the module view of software manually and published the module views of Apache Hadoop, Apache OODT, and ArchStudio. We shall use them as ground-truth results in our experiments. MoJoFM [11] is used to measure the similarity degree between the ground-truth and a given module view.

We performed experiments on various configurations to compare with our approach as shown in Table 1. M represents the weighted integration method proposed in the previous section. STR is a method utilizing only structural dependency, while SEM utilizes only textual one. We assigned fixed weights to C1, C2, ..., and C13 to observe how the performance is varied on different settings.

Table 2 shows the average MoJoFM values obtained from 100 runs for each combination of clustering algorithms and weighting methods. Since MoJoFM value measures the similarity degree between a module view and its ground-truth, the higher means the better. In the table, MoJoFM values in boldface represent the maximum in each case.

Our proposed method performs the best in 6 out of

**Table 1** Weights of various approaches

| Method | Structural Weight | Textual Weight | Method | Structural Weight | Textual Weight |
|---|---|---|---|---|---|
| M | Q | Min(1, D/Q) | C6 | 1.5 | 1 |
| STR | 1 | 0 | C7 | 1.5 | 2 |
| SEM | 0 | 1 | C8 | 1.5 | 2.5 |
| C1 | 1 | 1 | C9 | 2 | 1 |
| C2 | 1 | 1.5 | C10 | 2 | 1.5 |
| C3 | 1 | 2 | C11 | 2 | 2.5 |
| C4 | 1 | 2.5 | C12 | 3 | 1 |
| C5 | 1 | 3 | C13 | 3 | 2.5 |

**Table 2** Average MoJoFM(%) of clustering results

| Method | Blondel's clustering | | | Bunch | | | BorderFlow | | |
|---|---|---|---|---|---|---|---|---|---|
| | OODT | Hadoop | Arch-Studio | OODT | Hadoop | Arch-Studio | OODT | Hadoop | Arch-Studio |
| M | **72** | **69** | 60 | 55 | **48** | 52 | **69** | **62** | **65** |
| STR | 67 | 54 | 56 | 52 | 46 | 45 | 61 | 56 | 54 |
| SEM | 55 | 37 | 38 | 43 | 42 | 40 | 53 | 50 | 48 |
| C1 | 66 | 55 | 52 | 53 | 43 | 46 | 64 | 53 | 52 |
| C2 | 56 | 48 | 50 | 51 | 44 | 50 | 58 | 52 | 54 |
| C3 | 62 | 45 | 49 | 48 | 47 | 48 | 58 | 46 | 49 |
| C4 | 68 | 46 | 50 | 46 | 46 | 46 | 63 | 42 | 50 |
| C5 | 54 | 39 | 42 | 51 | 40 | 42 | 48 | 38 | 37 |
| C6 | 52 | 40 | 46 | 54 | 39 | 45 | 59 | 48 | 46 |
| C7 | 61 | 35 | 52 | **57** | 45 | 47 | 62 | 50 | 52 |
| C8 | 57 | 52 | 55 | 54 | 46 | 55 | 60 | 47 | 53 |
| C9 | 67 | 65 | 62 | 48 | 45 | 56 | 62 | 54 | 52 |
| C10 | 60 | 56 | **64** | 43 | 43 | 55 | 56 | 52 | 60 |
| C11 | 63 | 54 | 52 | 46 | 42 | **58** | 53 | 56 | 52 |
| C12 | 62 | 64 | **64** | 51 | 36 | 52 | 55 | 57 | 52 |
| C13 | 55 | 58 | 53 | 47 | 41 | 54 | 52 | 51 | 53 |

the total 9 cases. Especially, it derives the module views most similar to the ground-truth for the all projects when the BorderFlow clustering is adopted. It should be noted that our method provides consistently good results even when it is not the best. In the cases of ArchStudio with Blondel's clustering, OODT with Bunch, and ArchStudio with Bunch, the proposed method ranks 4th, 2nd, and 6th, respectively.

Most importantly, the proposed weighting method provides better performance than STR and SEM, where either structural dependency or textual similarity is exclusively utilized, in every case. In a recent study [11], nine different architecture recovery techniques for various open source projects are compared. The best MoJoFM values for OODT, Hadoop, and ArchStudio are reported as 48%, 63%, and 88%, respectively. The experimental results show that our approach based on Blondel's graph clustering outperforms the state-of-the art techniques for OODT and Hadoop.

Our experimental results support the idea of integrating structural dependency and textual similarity to achieve better performance of the module view recovery. In addition, the integration weights can be determined adaptively by considering the quality of dependencies.

## 5. Conclusion and Future Work

In this paper, we proposed an adaptive approach to integrate structural dependency and textual similarity information in deriving a software module view. Our approach utilizes both dependencies to achieve better performance and determines the combination weights automatically by analyzing their quality. The experiment results show that our proposed method is effective in deriving the module view for software architecture recovery.

For future work, we are planning to perform experiments with more software projects with ground-truth data and extend our approach to God class detection. In addition, a mechanism and the effect of adding evolutionary dependency into our approach will be investigated.

## Acknowledgments

This research was supported by the National Research Foundation (NRF-2014R1A2A2A01005519) and the MSIP under the ITRC (Information Technology Research Center) support Program (IITP-2015-H8501-15-1012) supervised by the IITP, Korea.

### References

[1] A. Kuhn, S. Ducasse, and T. Girba, "Semantic clustering: Identifying topics in source code," Information and Software Technology, vol.49, no.3, pp.230–243, 2007.

[2] A.N. Ngomo, F. Schumacher, "BorderFlow: a local graph clustering algorithm for natural language processing," Proc. CICLing 10th International Conference, pp.547–558, 2009.

[3] BorderFlow, http://sourceforge.net/projects/borderflow/

[4] B.S. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the Bunch tool," IEEE Trans. Softw. Eng., vol.32, no.3, pp.193–208, 2006.

[5] Bunch, https://www.cs.drexel.edu/~spiros/bunch/

[6] C. Patel, A. Hamou-Lhadj, and J. Rilling, "Software clustering using dynamic analysis and static dependencies," Proc. European Conference on Software Maintenance and Reengineering, pp.27–36, 2009.

[7] D.M. Blei, A.Y. Ng, and M.I. Jordan, "Latent Dirichlet allocation," Journal of Machine Learning Research, vol.3, pp.993–1022, 2003.

[8] G. Bavota, A.D. Lucia, A. Marcus, and R. Oliveto, "Using structural and semantic measures to improve software modularization," Empirical Software Engineering, vol.18, no.5, pp.901–932, 2013.

[9] Gephi (Supporting Blondel's clustering), http://gephi.github.io/

[10] J. Garcia, D. Popescu, C. Mattmann, N. Medvidovic, and Y. Cai, "Enhancing architectural recovery using concerns," Proc. International Conference on Automated Software Engineering, pp.552–555, 2011.

[11] J. Garcia, I. Ivkovic, and N. Medvidovic, "A comparative analysis of software architecture recovery techniques," Proc. IEEE/ACM 28th International Conference on Automated Software Engineering, pp.486–496, 2013.

[12] J. Garcia, I. Krka, C. Mattmann, and N. Medvidovic, "Obtaining ground-truth software architectures," Proc. International Conference on Software Engineering, pp.901–910, 2013.

[13] Mallet, http://mallet.cs.umass.edu/

[14] M.E.J. Newman, "Modularity and community structure in networks," Proc. National Academy of Sciences of the USA, vol.103, no.23, pp.8577–8582, 2006.

[15] N. Sangal and V. Sinha, "Using dependency models to manage complex software architecture," Proc. ACM SIGPLAN conference on Object-oriented programming, pp.167–176, 2005.

[16] T. Lutellier, D. Chollack, J. Garcia, L. Tan, D. Rayside, N. Medvidovic, and R. Kroeger, "Comparing software architecture recovery techniques using accurate dependencies," Proc. International Conference on Software Engineering, 2015.

[17] V.D. Blondel et al., "Fast unfolding of communities in large networks," Journal of Statistical Mechanics: Theory and Experiment, vol.2008, 2008.

[18] Y. Liu, D. Poshyvanyk, R. Ferenc, and T. Gyimothy, "Modeling class cohesion as mixtures of latent topics," Proc. IEEE International Conference on Software Maintenance, pp.233–242, 2009.