International Journal of Computer Science and Software Engineering (IJCSSE), Volume 6, Issue 4, April 2017
ISSN (Online): 2409-4285 www.IJCSSE.org Page: 94-98

Algorithms for Automating Artifact Attribute Classification of Software Systems

V. Sethupathi[1] and E. George Dharma Prakash Raj[2]

[1] Department of Computer Science & Engineering, JNTUA College of Engineering, Anantapur, India.

[2] Professor in Department of Computer Science & Engineering, JNTUA College of Engineering, Anantapur, India.

[1]koti4252@gmail.com, [2]akepogu@gmail.com

ABSTRACT

Software systems generally involve a number of phases and tend to evolve over a period of time. Several revisions of individual artifacts which make up the system take place during the evolution process. The revisions and refinements are captured and maintained as different versions using configuration/version management tools. A key issue in the version management of object oriented software system is classification of attributes of an artifact. Evolution needs to be captured, for capturing the evolution we need to maintain the various artifacts of software systems. For this purpose the behavior of the artifacts to be understood properly. This is possible through various attributes of artifacts. Presently attributes of an artifact are categorized into two different types called versioning and non-versioning. Versioning attributes govern the behavior of an artifact and non versioning attribute do not. This paper proposes an algorithms for automating the process of determining the versioning and non-versioning functionalities of the above classification.

Keywords: *Change Propagation, Equivalent Change, Version Change, and Version Management*.

1. INTRODUCTION

Software systems are developed generally based on an iterative paradigm, where each iteration provides a successive refinement over previous iteration. Refinements in software systems are managed by maintaining different configurations of various artifacts of the systems. User requirements of software systems keep changing. This change leads to evolution of software system. As the requirements of users' changes, software has to support the evolution easily. The changes in the artifact normally require corresponding changes in other dependent artifacts. Therefore there is a need to capturing the evolution of related artifacts to keep the system in consistent manner. Capturing the evolution of software system is major issue in software maintenance phase. The concept of version management is used for managing the evolution of the software system.

A key issue in the version management [1], [2] of object oriented software system is classification of attributes of artifacts into two categories namely versioning and non-versioning attributes. Here an attribute is used to mean an instance variable or method of a class. If a change of an artifact leads to change of other related artifacts, then it is versioning attribute, otherwise it is a non-versioning attribute. Versioning attributes determine major functionality of software system and non-versioning attributes determine minor functionality. Version of an artifact is represented in the form: "<major><minor>". If there is a major change in the functionality of an artifact then it is said to be version changed. This is caused when there is a change in one or more of its versioning attributes. On the other hand if there is a minor change in the functionality of the artifact then it is said to be equivalent change. This is caused when there is a change in one or more of its non versioning attributes.

Towards this end the paper discuses about algorithms for automating artifact attribute classification of software systems**.** The remainder of the paper is structured as follows. Section II reviews literature on prior works on artifact attribute classification. Section III presents change propagation mechanism. Section IV provides algorithms & system design and Section V presents conclusions of the paper besides providing directions for future work.

2. RELATED WORK

Many researchers contributed towards artifact attribute classification of software systems. In [1], [2] classification of attributes of artifacts into two categories namely versioning and non-versioning is performed. There are also several authors presented major and minor

functionalities of artifacts of software systems [4] [6]. However, automation is important factor that is to considered with respect to automation attribute categorization. Mahmoud Masmoudi et all., [11] proposes a new change prediction technique for changes are propagated from artifact to artifact and from function to functions, leading to nonvalue development activities with extra costs and delays.

3. CHANGE PROPAGATION MECHANISM

In configuration management, generally whenever a change occurs in an artifact that has to be propagated to the all related artifacts. This propagation preserves the consistency of software system under consideration.

Varieties of change propagation

In object oriented technology classes are considered as basic building blocks of software system. These classes are related using various types of relationships among them such as inheritance, aggregation and association. The classes are represented as an artifact in Attribute Classification (AC) graph and relationship among the classes are represented as links. These links are labeled as cohesive or non-cohesive in the AC graph. In AC graph change is propagated to the related artifacts based on two values which are called as focus of change and property of a link between the artifacts i.e. cohesive or non-cohesive. The propagations in AC graph are categorized into two categories, one is propagation of equivalent change and the other is propagation of version change. These two categories are tabulated in Tables 2.1 and 2.2 respectively. Whenever a change is propagated in the AC graph. The recommended changes are shown in the following tables.

Table 2.1: Version Propagation Table

	Version Focus	
	LOW	HIGH
Cohesive link	V-Change	E-change
Non-Cohesive link	E-change	E-change/ N-change

Change propagation in case of version change of an artifact is as follows.

- If the link is cohesive and version focus is LOW then a version change (V-Change) is recommended to related artifacts or AC graph node.
- If the link is cohesive and version focus is HIGH then an equivalent change (E-Change) is recommended to related artifacts.
- If the link is Non-cohesive and version focus is LOW then an equivalent change (E-Change) is recommended to related artifacts.
- If the link is Non-cohesive and version focus is HIGH then an equivalent change (E-Change) is recommended to related artifacts. If the version focus is too HIGH and cohesion of link is too low then no change is recommended.

Table 2.2: Non-Version Propagation Table

	Equivalent Focus	
	LOW	HIGH
Cohesive link	E-Change	E-Change
Non-Cohesive link	E-Change	N-Change

Change propagation in case of equivalent change of an artifact is as follows.

- It the link is cohesive and focus is LOW, then an equivalent change is recommended to related artifacts.
- If the link is cohesive and focus is HIGH, then an equivalent change is recommended to related artifacts.
- If the link is non-cohesive and focus is LOW then an equivalent change is recommended to related artifacts.
- If the link is non-cohesive and the focus is high then no change (N-change) is recommended to related artifacts.

Reasons for Change Propagation

Change propagation can occur because of two reasons:

- If an attributes of an artifact is changed, then change is propagated to related artifacts. Various cases of this reason have been depicted in the table 2.3
- New dependency links will be created when a new artifact is added to the system. These link directions can be to or from the new artifacts. The recommended changes of an artifact based on the direction as well as cohesiveness of the link are shown in table 2.3.

Change Management

A class is considered as a basic entity in object oriented systems. Hence, each class is treated as an artifact and denoted as a node in AC graph. Links between the AC graph nodes shows the relations between the classes such as inheritance, aggregation and association. It is easy to manage the versions through a AC graph. There are two issues of change management. These are version change and version propagation. They are addressed below. When ever a version change occur to an artifact there is a need to propagate the change to other dependent artifacts.

Version change

Version change of software systems are of two types. One is change in version and the other one is equivalent change. If the changes in software are significant and affect the software system functionality then it is a version change. Otherwise if the changes in software are due to minor improvements and system functionality is not affect much, then it is said to be an equivalent change. Changes can also be categorized as follows. One is internal change of artifacts and other is change propagated from related artifacts. Internal change of an artifact can occur through version or non-versioning attributes. The type of change of an artifact is decided by versioning attributes or non-versioning attributes. Change can occur in two ways. One is change in attributes and the other is addition of new attributes to the artifact. If the attribute is versioning attribute then the type of the change occurring in the class is called as version change (V-change). If the attribute is non-versioning attribute then the type of the change occurring in the class is called as equivalent change (E-change).

Version Propagation

In every software system the changes of the artifact will cause changes of other related artifacts. Thus change propagation mechanism is a major issue in version management. Version change of an artifact will occur if the related artifacts having accessibility to the artifact attributes and functionality. In a class there are three types of access specifiers for an attribute namely public, private and protected. The main aspects of version propagation are focus and cdegree (degree of cohesion). The focus is with respect to change in AC graph node. Each node in AC graph represents an artifact of the software system. A change in AC graph node has a value called focus [1]. The focus of change is a probability that the change does not impose similar change in other related AC graph nodes which are nothing but artifacts. Related AC graph nodes means there exist some dependency links among corresponding artifacts of AC graph.

The change pertaining to an attribute depending on the accessibility is tabulated in table 3. 1 If an attribute of an artifact is private then change of that attribute may not impose changes in related artifacts. Thus the focus of private attribute is HIGH.

Table 3.1: Focus evaluation table

Attribute	FOCUS
Private	HIGH
Public	LOW

Protected	If (link = Inheritance)
	LOW
	Else
	HIGH

Similarly public attribute focus is LOW. If the attribute is protected, the change may affect the related artifacts depending on link between AC graph nodes. If the link is inheritance link then focus is LOW, otherwise focus is HIGH. Cdegree of a link is the indicator of the amount of dependency that exists between the two related artifacts [1]. The value of the cdegree has a range [0,1]. The link is said to be "strong", if the cdegree value is more than the threshold (say 0.5) and this link is called cohesive link. The link is said to be "weak" if the cdegree value is less than threshold value and link is called non-cohesive link.

4. SYSTEM DESIGN & PROPOSED ALGORITHMS

The following section explains AC graph generator, attribute classifier and change propagator.

AC Graph Generator

The Fig. 4.1 shows AC graph generator which takes two versions of software systems as input and generates their corresponding AC graphs.

Attribute Classifier

The input to attribute classifier is AC graphs of the two versions, which are generated by the AC graph generator. It gives the AC graph of the first version with

its attributes classified as versioning, non-versioning or unknown. Fig. 3.2 shows the attribute classifier

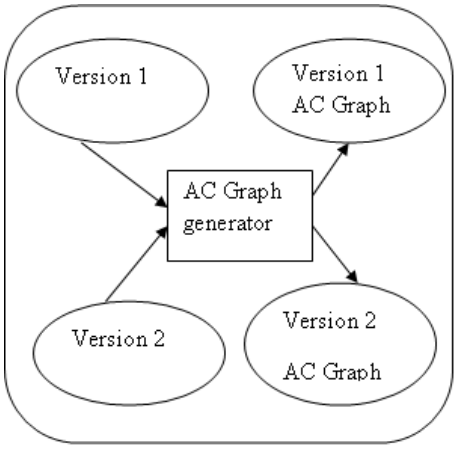
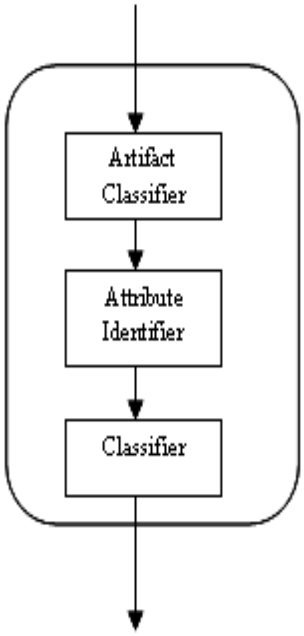


Fig. 4.1 AC Graph Generator

AC Graph of Two Versions



AC Graph of first version with Attribute Classified

Fig. 4.2. Attribute Classifier

Change Propagator

The input to change propagator are class (artifact) in the source version and the modified class. It generates the list of affected artifacts by using the AC Graph of the source version. Fig 4.3 shows change propagator.

Algorithms

The following sections explains the AC Graph generation, Attribute classification

AC graph generation

Input: Two successive versions of the project of two versions.
Output**:** AC graphs
Repeat the steps 1 to 2 for each class (artifacts) present in the source.
Step 1: A table of all existing classes is constructed in the first parse of the source. The attributes and methods are extracted for each class. This forms the URA node of this class.
Step 2: The links between the classes are determined in the second parse of the source.

Attribute classification

Input**:** AC graphs of two versions, generated by the AC graph generator.
Output**:** AC graph of first version, with the attributes of the artifacts classified as versioning,
non-versioning and unknown.
Algorithm**:**
Step 1: Consider a particular class from the two versions of system.
Step 2: Determine whether the change is version change or an equivalent change.
Step 3: Determine the attributes, which caused the above change.
Step 4: Accordingly classify the attributes as versioning and non-versioning attributes.

5. CONCLUSIONS

This process of automation provides algorithms for finding the versioning and non-versioning attributes of an artifact. The process of automation of attribute classification is carried out in this paper. Since, software systems produced thousands of millions of artifacts, using this method really helps categorizing attributes effectively and quickly. As a future directions an improved measures for calculation of cdegree (cohesion degree) and focus may be adopted. In addition, the effect of changes made in an artifact can be determined to a higher degree of precession. This may be achieved by slight improvement in the strategy used for keeping track of the links of an artifact.

REFERENCES

[1] D. Janaki Ram, M. Sreekanth, A. Ananda Rao, Version Management in Unified Modeling Language", Technical Report IITM-CSE-DOS, IIT Madras, India.
[2] D. Janaki Ram, S. Sreenath, R. Rama Krishna," A Generic Model for Semantics- Based Versioning in Projects ", IEEE Transactions on Systems, Man and Cybernetics, vol. 30, No. 2, March 2000.
[3] S. Srinath, k. Venkatesh, D. Janaki Ram,"An Integratd Solution Based Approach to Software Development using Unified Reuse Artifacts", ACM Software Engineering Notes. July 1997.
[4] Lucki, "A Graph Model for Software Evolution ", IEEE Transactions on Software Engineering, Vol. 60, No. 8, Aug 1990.
[5] Chia-Song Ma, Carl K. Chand and Jane Cleland-Huand, "Measuring the Intensity of Object Coupling in C++ Programs" IEEE 2001.
[6] "Versioning in Apache", Http//www.apr. apache.org/versioning.html.
[7] Beech D. and B. Mahbod, "Generalized Version Control in an Object Oriented Database", ICDE, PP. 14-22, 1988.
[8] Zeller A., A Unified Version Model for Configuration Management, SIGSOFT'95: proceedings of 3rd ACM SIGSOFT symposium on foundations of software engineering, New yark, NY, USA .
[9] Babich W. A., software configuration management. Addotion –Wesley, Reading, Massachusetts, 1986.
[10] Conradi R. and B. Westfechtel, Version Models for Software Configuration Management, ACM Compt. Surv., Vol. 30 , no. 2, pp. 232-282, 1998.
[11] Mahmoud Masmoudi, Patrice Leclaire, Marc Zolghadri and Mohamed Haddar, "Change propagation prediction: A formal model for two-dimensional geometrical models of products", Journal of Project Management, 300–314, March 2017.