Software Design Versioning using Propagation Probability Matrix

A. Ananda Rao and D. Janakiram
Distributed & Object Systems Lab
Dept of Computer Science & Engineering
Indian Institute of Technology, Madras, India
{anand, djram}@dos.cs.iitm.ernet.in

Abstract

In existing models, different versions of an artifact are created based on two types of artifact's attributes called key and non-key attributes. The attributes that govern the design of an artifact are called key attributes (e.g. interface). The attributes that do not govern the design of an artifact are called non-key attributes (descriptive attributes). Changes in key attributes lead to a new design version of an artifact and changes in non-key attributes lead to design equivalents. In either of the above cases, a change in one artifact (in most cases) will result in changes to other related artifacts. But, design versions are a major concern in redesign process because, whenever design of an artifact changes its related artifacts design also will be changed. Therefore, the accurate prediction of design change propagation provides a significant challenge in management of design versions. Hence, this paper focuses on construction of design change propagation probability matrix for software artifacts. Each entry in the matrix represents the probability that a design change in one artifact requires similar changes in other related artifacts. In the process of matrix construction, an earlier proposed model namely, Unified Representation of Artifacts (URA) is used for representing the various artifacts of software systems.

Keywords: Artifact, Design change, Evolution, Change propagation probability matrix, Version management.

1 INTRODUCTION

The software systems are evolving constantly because of the continuous change in requirements specification. That is why in software development, large projects are generally implemented based on iterative paradigm. Iterative process provides successive refinements over previous iterations. These refinements lead to redesign of existing artifacts. During redesign process a change in one artifact (in most cases) will result in changes to other related artifacts. Therefore, the accurate prediction of design change propagation provides a significant challenge in management of design versions. This paper considers the evolutions of design artifact in the context of corrective and perfective maintenance.

In existing design version management models, evolutions of design artifact are managed in different ways based on the type of attributes that participate in the change. Presently the attributes[1] of an artifact are categorized into two types as key attributes (design attributes) and non-key attributes (non-design attributes) [1, 2]. The design attributes[2] govern the design of an artifact and non-design attributes do not govern the design. If any change occurs in the design attributes, it leads to creation of new design version of an artifact. On the other hand changes in non-design attributes lead to design equivalents. For example, consider the design of an elastically stressed element, the attributes considered include Young's modulus, thermal conductivity, ductility, Poisson's ratio, colour etc. Of all these attributes, only Young's modulus govern the design and can be considered as a key attribute. Any change in this attribute can be considered to be a new design of an elastically stressed element. However, changes in other attributes such as colour, ductility etc. will not create new design of an elastically stressed element. These attributes are called non-key attributes.

In either of the above cases, a change is notified to the related artifacts. Based on the amount of change in an artifact and the strength of coupling between the artifacts, the related artifacts will be evolved into their respective versions. But, design versions (changes in key attributes) are a major concern in redesign process because, whenever design of an artifact changes its related artifacts design also will be changed. That is, semantics of related artifacts will be changed. Therefore, it is always an advantage to have a design change (design version change) propagation prediction that conveys the number of related artifacts to be redesigned

[1] In this paper attribute means feature of an artifact

[2] In the paper the terms design attributes and key attributes and non-design attributes and non-key attributes are used interchangeably

if particular artifact changes its design version in the design. This issue is not addressed in the literature. Therefore, this paper is aimed at tackling the above issue. In the case of design equivalents (changes in non-key attributes), one design equivalent of an artifact can be replaced with another without affecting the design of the related artifacts. Therefore, design equivalents propagation is not considered in this paper.

This paper proposes a construction of Design Change Propagation Probabilities (DCPP) matrix for software design artifacts with respect to (changes in key attributes) design versions. Given a software design that consists of N artifacts, N by N matrix is constructed. The entry at row A, column B represents the probability that a design change in artifact A requires similar (design) change in B so as to preserve the overall function of the system. This matrix captures the possible impact of corrective and perfective maintenance in artifacts on the other artifacts in the system. The Unified modeling Language (UML) is used to represent the design of a software system. Using UML diagrams, the relationships and the interactions between artifacts will be obtained easily. This information is used to estimate change propagation probabilities of adjacent artifacts. The constructed matrix will be mapped on to Unified Representation of Artifacts (URA) graph that produces the graph theoretical properties. Using URA graph the DCPP can be computed between artifacts that are related via intermediate artifacts or that are related through simple paths. The constructed matrix provides a wealth of useful information from the design version management perspective. For example, based on these design change propagation probabilities it is possible to know the complexity of a design. Providing such type of information in advance of software development helps in a great way.

The organization of the paper is as follows. Section 2 presents the related work on design version management and change propagation. It also presents the brief introduction about URA graph. Section 3 addresses construction of design change propagation probability matrix. Also, advantages of the construction of matrix are explained with an example. The conclusions and future directions of the work have been placed in section 4.

2 RELATED WORK

This section presents related work first and discuss the need for development of design change propagation probability matrix for managing different versions of software design artifacts. It also presents brief introduction about URA model.

2.1 Design Version Management Related Work

Different versions of composite objects are maintained in [1]. The attributes of an artifact are divided into two types as intrinsic and non-intrinsic. Various versions of an artifact are created based on the type of attribute that participate in the change. Management of different design versions of an artifact in design databases is discussed in [2]. In this model, attributes of an artifact are divided based on semantics into two different types as design(key) attributes and non-design(non-key) attributes. But these models do not provide change propagation rules.

A generic model for semantics based versioning in projects is proposed in [3]. It addresses the changes as well as the change propagations. But, the change propagations are performed on conceptual values like High or Low. The model is developed based on the URA mechanism. The following subsection gives the brief introduction about the generic model. The authors also proposed version management in Unified Modeling Language [4]. This paper discusses the semantic based version management in the projects that are represented by UML. The UML class diagrams are used to represent the semantic entities in the project. The version propagation is captured through class diagrams and their relationships.

[5] uses design structure metrics to estimate how change would propagate in a system. It uses different techniques to identify relations between the components of the system (or the design tasks). High connectivity between these components suggest that high levels of dependency exists between the components of resulting system. No indication such as the probability of redesign is given by these metrics. [6] describes prediction and management of changes to an existing product resulting from faults or new requirements. In this model risk of change is identified as product of likelihood and impact of a change. But, it assumes that the values for likelihood and impact of a change are provided by the experienced designer. Also, the above two models deal with the software artifacts where their attributes are of the same kind. [7] deals about change propagation probabilities of components of software architecture. But, it does not calculate combined effect of changes that come from various paths and also do not calculate the change propagation probabilities of software components that are related through intermediate components. It only estimates upper bounds of change propagation probabilities of these components. It also do not consider components where their attributes are categorized into different types. However, the matrix construction idea has been taken from this and reapplied in this paper for software design version management.

In all the above said models, change is propagated to other artifacts based on the amount of change and strength of dependency that exists between related artifacts. The

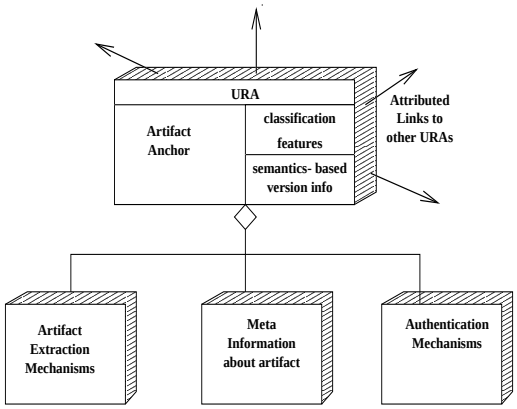


Figure 1. Structure of a URA

amount of change and strength of dependency are represented conceptually which depends on the experience of the designer. Also, they do not provide a way to calculate probability values based on which change may be propagated. Therefore this paper proposes a quantitative approach to construct design change propagation probability matrix whose values are used to propagate changes among artifacts.

2.2 Introduction to URA

A meta level entity called URA which represents an artifact of any type or granularity is presented in [3] [8]. Any logical entity of interest is an artifact. Artifacts map to physical entities in different ways like classes, sets of classes, subsystems, documents etc. The structure of URA is shown in figure 1. A URA mainly consists of three components - first one for extracting the artifact from the information system, second one contains the information about the artifact, third and the last one enforces authentication mechanisms. A set of features is associated with the URA, which allows it to be classified and queried. These features can be either attributes or functionalities of the artifact. Semantics based version information set keeps track of the evolution of the artifact. In addition to these, there are labeled links pointing to other URAs, which reflect the relationship between the artifacts that the URAs represent.

A project is represented as a directed graph of URAs. The graph will evolve as changes occur in the project. An artifact in the project is represented as a URA, that is a node in the URA graph. Directed edges in the graph are labeled. These labels are the relationships between the artifacts. The labeled links indicate the dependencies between the nodes of the graph and the need to propagate the changes. A pivot node in the graph represents the whole project. URA nodes are linked to pivot node by dependency links. Changes are propagated to this node too. The version of this node is the version of the project.

3 CONSTRUCTION OF DCCP MATRIX

In this section, the construction process of Design Change Propagation Probability (DCPP) matrix is explained initially. Various advantages and applications of DCCP matrix is explained later.

In semantics based versioning, various versions of an artifact are managed as design versions and design equivalents. But, design versions (changes in key attributes) are a major concern in redesign process because, whenever design of an artifact changes, its related artifacts design also will be changed. Design equivalents can be replaced with other design equivalent without affecting the over all functionality of the system. Therefore, this paper considers the design change propagation matrix with respect to design changes of artifacts.

3.1 Design Version Change Propagation

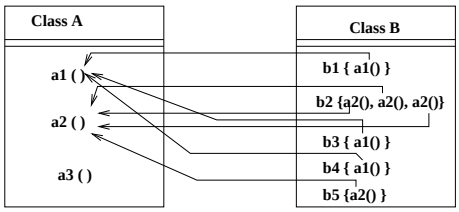
Whenever a change occurs in an artifact it has to be propagated to the related artifacts. This is propagated based on the amount of dependency that exists between the related artifacts. The amount of dependency is represented by the term called *cdegree* in this paper. The *cdegree* defined as follows.

Definition of cdegree: The degree of cohesion (*cdegree*) of a link is the indicator of the amount of dependency that exists between two related artifacts represented by the URAs. The value of a *cdegree* has the range [0,1]. Therefore, a change is propagated to related artifacts based on the *cdegree* value. For example, a design change is propagated to related artifacts whose *cdegree* value is more than the threshold (say 0.3) value. Since the artifacts are related one another directly (adjacently) or indirectly (through intermediate artifacts) the change propagation probability (*cdegree*) should be calculated for the above two cases. The following subsection explains the estimation of *cdegree* value.

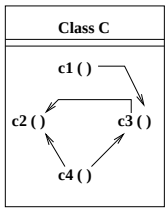
3.1.1 Cdegree Estimation

To derive formulas to estimate *cdegree*, the design of software is represented using UML. From UML diagram it is easy to obtain design level information that is needed to calculate *cdegree*. Calculation of *cdegree* depends on the artifacts involved. First a simple formula for calculation of *cdegree* between adjacent artifacts is presented and later a formula is proposed to calculate *cdegree* between artifacts that are related through intermediate artifacts.

Let A and B are two artifacts that are related adjacently and attributes of an artifact B access attributes of an artifact A. Since various number of links are possible between



a) Two related artifacts



b) Interactions of attributes of an artifact

Figure 2. Example class diagrams and their interactions

two artifacts, each link can be given a weightage. Based on these weightages the total strength between these two artifacts can be calculated. For example, consider a figure 2 (a) where class A has three attributes (a1, a2 and a3) and B has five (b1, b2, b3, b4 and b5) attributes and attribute a2 is called four times by attributes of an artifact B. Therefore, the weightage of attribute a2 is 4/7 where total number of calls exist between A and B are 7. Similarly, weightage for other attributes can also be calculated. After calculating the weightages for all the attributes, *cdegree* between A and B is defined as follows.

cdegree = sum of weightages of each link with respect to an attributes of A from B / Total number of possible links from B to A.

The denominator can be taken as the total number of attributes exist in A, since this many maximum links can be made. In the above equation, a link is defined as a call made by class B with respect to a method. It is irrespective of number of calls made to each method . That is, all calls of a method constitute a single link even though if it is called more than once by a method of class B. The number of call references are taken into consideration while calculating weightage of each attribute.

The above estimated *cdegree* value represents the probability of the changes made to artifact A that causes similar changes in artifact B. But, the above procedure is used to estimate *cdegree* value between adjacent artifacts. A more general approach would be to estimate the ramifications due to a single change. The motivation for this is based on considering the change as a signal and the project graph as a network of conductors for the signal. When a signal is injected into the network, the strength of the signal at any point in the network is dependent on the strength of original signal and the conductivity of the network. Based on the above analogy, the type of change recommendation received by a node in the URA graph is dependent on the strength of the change and the conductivity of the URA graph (*cdegree* value). The values of *cdegree* for the artifacts that are not connected directly but connected via intermediate artifacts can also be calculated. Artifacts in the design do not have predetermined behavior. Therefore, to know for each artifact how it is affected by change and how its change affects the other artifacts depends on the exact change that is required. Therefore, the following method is used to compute combined *cdegree* value for the artifacts that are connected through intermediate artifacts. The combined *cdegree* value is the probability that the end effect will arise, regardless of the path. This can be calculated using probability lemmas. While calculating the combined *cdegree* value it is to be noted that the events are not mutually exclusive. Therefore, the following formulas are used to estimate the combined *cdegree* values.

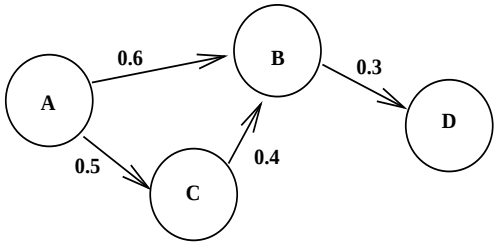


Figure 3. Example URA graph of artifacts

$$cdegree_{p,q} \cap cdegree_{p,r} = cdegree_{p,q} X\, cdegree_{p,r} \qquad (1)$$

$$cdegree_{p,q} \cup cdegree_{p,r} = cdegree_{p,q} + cdegree_{p,r} - (cdegree_{p,q} X\, cdegree_{p,r}) = 1 - [(1 - cdegree_{p,q}) X (1 - cdegree_{p,r})] \qquad (2)$$

For example, in figure 3, values present on the top of links represent *cdegree* values between various artifacts. The combined *cdegree* value at artifact B in the figure is calculated as follows.

$cdegree_{a,b}$ = 1 - [(1-$cdegree_{b,a}$) x (1-($cdegree_{b,c}$ x $cdegree_{c,a}$))] = 0.68.

3.1.2 Example Diagram and its DCPP Matrix

Consider a design digram that is shown in figure 4. The design change propagation probability (*cdegree*) values are calculated for this diagram as explained in the previous subsection. The calculated values are tabulated in table 1. Since each software design artifact is represented using a URA,

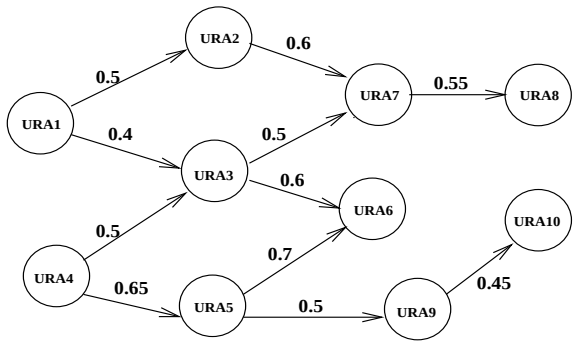


Figure 4. Example Design Diagram

the matrix is constructed using N by N URAs. In the matrix, changes are initiated in the URAs that represent rows. The URAs that represent columns are affected URAs. Note that, in the table probability values along the diagonal of a matrix are one. This represents the probability that any change in an artifact affects its design is maximum (i.e., one). Hence probability values along the diagonal of a matrix are represented by one. This table gives wealth of useful information about the design version management. The uses of the table are discussed as follows

3.2 Advantages of DCPP Matrix

The matrix values not only helps in finding out number of different versions of an artifact that are to be created but also helps in number of ways. The advantages are mentioned below.

1. It is possible to know the complexity and cost of maintenance operations on the design of software system. The identity matrix indicates the perfect modular system which may not be possible in reality. In this type of systems, changes are local to the artifacts. The closer a matrix to identity case the better. By observing example figure it can be found that the design of this diagram is not too complex because only few (URA_1 and URA_4) artifacts propagate their design change to more number of artifacts.
2. If a column contains larger values with respect to a particular artifact, then it can be inferred that this artifact is likely to undergo frequent changes in the maintenance phase. Therefore, care must be taken to redesign this artifact to make modifications easy. There are no such type of artifacts exist in example design diagram.
3. If a row contains larger values with respect to a particular artifact, this situation indicates that, any change to this artifact will require changes in more number of artifacts and hence lot of versions will be created which results in more maintenance. Therefore, preventive measures can be taken to optimize the design of this component. From the values of table 1, it can be stated that no artifact comes under this category.
4. DCPP matrix can also be used to compare different designs of software system with respect to maintenance costs. Therefore, by constructing design change propagation probability matrix, one can obtain not only the information at an early stage that is useful in maintenance phase but also the information about the design version management.

4 CONCLUSIONS AND FUTURE DIRECTIONS

A methodology is proposed to construct DCPP matrix. This methodology (matrix) provides a quantitative measure to calculate *cdegree* value that is used to propagate design version change between artifacts. With the help of UML diagram of software design, the change propagation probability (*cdegree*) of adjacent artifacts is calculated. The DCPP matrix is expressed in terms of URA graph to calculate *cdegree* values for the artifacts that are not connected directly. The number of advantages and applications of DCPP is discussed.

The proposed metrics and concepts are to be evaluated empirically using a case study. The artifacts can also be categorized into various types based on the impact of average changes initiated in one artifact on the fraction of the total number of artifacts in the design. For example, an artifact is called as *Ripple* artifact if on an average changes initiated in this artifact produces the limited number of (few artifacts that are to be redesigned) changes in the design which are under control from the maintenance perspective.

References

[1] Rafi Ahmed and S.B. Navathe, *Version Management of Composite Objects in CAD Databases*, Proceedings of ACM SIGMOD international conference on Management of data, pp. 218 - 227, Denver, Colorado, May 29-31, 1991.

[2] Ramakrishnan, R., and D. Janaki Ram., *Modeling Design Versions*, Proceedings of 22^{nd} International Conference on VLDB, Mumbai(Bombay), India, pp. 556-566, September 1996.

[3] Srinath, S., R. Ramakrishna and D. Janaki Ram, *A Generic Model for Semantics Based Versioning in Projects*, IEEE Transactions on Systems, Man and Cybernetics, Part A, Vol. 30, No. 2, pp. 108-123, March 2000.

[4] D. Janaki Ram, M. Sreekanth and A. Ananda Rao, *Version Management in Unified Modeling Language*,

Table 1. DCPP matrix for design diagram

Artifacts	URA_1	URA_2	URA_3	URA_4	URA_5	URA_6	URA_7	URA_8	URA_9	URA_{10}
URA_1	1	0.5	0.4			0.24	0.44	0.14		
URA_2		1					0.6	0.34		
URA_3			1			0.6	0.5	0.3		
URA_4			0.5	1	0.65	0.6	0.25	0.14	0.33	0.15
URA_5					1	0.7			0.5	0.23
URA_6						1				
URA_7							1	0.55		
URA_8								1		
URA_9									1	0.45
URA_{10}										1

Proceedings of 6^{th} International Conference on Object Oriented Information Systems (OOIS 2000), London, pp. 238-252, December 18-20, 2000.

[5] Steward, D. V., *The Design Structure System: A Method for Managing the Design of Complex Systems*, IEEE Transactions on Engineering Management, EM-28(3),1981.

[6] Clarkson, P.J., C. Simons, and C.M. Eckert, *Predicting Change Propagation in Complex Design*, Proceedings 13th International Conference of Design Theory and Methodology (DETC'01), ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Pittsburgh, Pennsylvania, USA, 2001.

[7] http://sergiobogazzi.com/top/computers/software/software%20engineering/software%20metrics/code%20metrics/change%20propagation/cguide/2003/cp.doc

[8] S. Srinath, *URA: A Paradigm for Context Sensitive Reuse*, A Thesis of Master of Science by Research, Department of computer Science & Engineering, Indian Institute of Technology, India, April 1998.