A Graph-Based Requirement Traceability Maintenance Model

Facilitating Chronological Evolution

Vikas Shukla[1, 2], Guillaume Auriol[1, 2], Claude Baron[1, 2]
[1] LAAS-CNRS ,
7 avenue du Colonel Roche, F-31077 Toulouse, France
[2] Université de Toulouse; UPS, INSA, INP, ISAE; UT1, UTM, LAAS ;
F-31077 Toulouse, France
{vshukla, gauriol, cbaron}@laas.fr

***Abstract*—Requirement traceability remains a challenging task for the software developers. It helps stakeholders to understand the various relationships between the artifacts produced during the development process. During this requirement evolution process, information is produced and is stocked as trace. Some part of this information is lost owing to traceability maintenance process as links are deleted and removed from the system. This lost information is very useful while making decisions during the development process. In this paper we discuss a graph-based traceability model, which allows easy maintenance without any significant information loss. We show that both nonfunctional and functional requirements can be traced forward and backward using our proposed graph-based traceability model.**

Keywords-Requirement Traceability; Graph; Maintenance; Decision making.

I. INTRODUCTION

Requirement traceability is the ability to describe and follow a requirement in both forward and backward direction in a software development life cycle [1]. Requirement traceability is seen as an index of software quality, it is one of the recommended activities for the system requirement specifications [2], CMMI and ISO 15504 consider it as 'best practice' and strongly suggest its usage. Requirement traceability allows various stakeholders to understand the various existing relationships among the produced artifacts during the product development process.

A requirement is traceable if you can discover who suggested the requirement, why the requirement exists, which requirements are related to it and how that requirement relates to other information such as systems design, implementation and user documentation. Traceability information helps you discover which other requirements might be affected by requirement changes.

Requirement traceability is always associated with artifacts, we define artifact as any product which may have originated during the course of development process or is utilized during the development process or later and is important for the success of project.

Every organization implements its own suitable guiding principles for requirement traceability which are known as 'traceability policies'. Traceability policies define which information dependencies between requirements should be maintained and how this information should be used and managed.

Traceability means different things for different types of users depending on the types of users high-end or low-end [3, 4]. Usually, quality requirement of a system, which are mostly nonfunctional requirements, are high-end users requirements associated with management people. Low-end users are usually developers, programmers or people involved with testing, verification or validation.

For high-end users it implies how the client needs have been fulfilled but usually the low-end users find it unnecessary work overload [3], Tracing of nonfunctional requirements satisfies their needs. Similarly the traceability need of low-end users is satisfied with functional tracing.

We have contributed to the existing state of art by proposing a valid solution to the maintenance problems, i.e., the information loss, and dangling traces. Our paper addresses solution for the existing requirement traceability maintenance problems using graph-based methodologies, based on event-based traceability [5]. We show how we can increase the value of trace for the low-end users and hence involve them rigorously in traceability process. Our approach shows the interesting solution for the dangling-trace and information-loss problem and shows how our technique can be suitably used for minimizing cost of maintenance.

The paper is organized as follows. Section 2 of this paper highlights the current traceability maintenance problems. Section 3 presents the existing related works. Section 4 presents our graph-based traceability maintenance model. Section 5 discusses various aspects of our maintenance scheme and discusses feasibility and scalability issues linked, and equally the various combinations possible with recovery schemes. Section 6 concludes the paper and brings the possible problems and solutions linked to our approach. Finally, Section 7 presents the future perspective works envisaged.

II. TRACEABILITY MAINTENANCE PROBLEM

The requirement traceability is a continuous activity, involving peoples of various levels to participate continuously and maintaining a perfect communication channel among them for avoiding any information lapse. A good communication channel can help to figure out inconsistencies in the interpretation of requirements among various stakeholders which is very necessary for requirement engineering activities. Besides the communication there are various issues in traceability maintenance. Maintenance is the activity of updating and modifying already existing traceability relationships [6]. We discuss a few of the

 ISBN: 978-1-61208-165-6

existing important maintenance problems, which we address in this paper.

A. Cost of Maintenance

As the requirements are continuously evolving through the life of a project, requirements are added, removed or modified. The links between these evolving requirements need to be maintained. In a sufficiently complex system, the number of requirements can vary up-to few thousand requirements depending upon the granularity. Maintaining these requirements can be tedious task involving lot of computational and human resources.

B. Dangling Trace

A dangling trace is one which points nowhere or it lacks either a source or a target [7]. Such situation may arise due to human or system error during the course of a continuous evolution of a fairly complex system. They may also arise due to changes in the system model rendering some part of old system out of the boundaries of new system and hence it becomes difficult to trace them with respect to new requirements.

C. Information Loss

Whenever a new requirement is added to the system it needs to be linked to other requirements and available artifacts. The corresponding owners of the linked artifacts should be informed and advised to bring up the necessary changes. Similarly whenever an artifact is removed or altered or its dependency changes all the information should be communicated to the various stakeholders. This task usually involves maintaining these fine grained relationships and continuous update of such information usually leads to loss of data and hence information. We claim this information to be important as they are result of earlier high level discussions and decisions which involved certain cost.

If any such information is deleted permanently then in case of a future discussion there is chance that development team may reach a similar decision which was earlier found to be inutile. This may happen due to a probable change in the team or may be just of a simple absence of a member, which is quite possible as project development may take sufficiently long time.

D. Increasing Value of Trace for low end users

As mentioned earlier, for the low end users traceability seems to be a monotonous task and they are reluctant to involve themselves in traceability process. They do not find it very useful for their objectives and hence traceability does not offer them sufficient valorization for their work.

Whereas with every change brought to an artifact during the course of development there is an inherent risk attached to every dependent artifact involved which may jeopardize the success of project. We show in the following section that this risk evaluation factor can be used as a tool to valorize the work of low-end users and hence to continuously involve them in traceability mechanism. This associated risk can then be utilized in change impact management.

III. OTHER REALATED WORKS

Current literature on traceability contains ample work on need, and generation of traceability [1]; however, fewer work has been produced regarding the maintenance of traceability [5, 7, 9, 10, 14] the existing ones do not address properly the information loss problem. Cleland-Huang *et al*. [5] proposes publish–subscribe mechanism, a relationship between artifacts is registered to a central server. The evolution is represented by the series of change event. When a requirement is changed, the subscribers are notified about the change and they may bring the potential changes to their artifacts. It allows complete removal of requirements.

Another event-based scheme [14] uses a tool called Ttracemaintainer but it uses only UML structural models. Another similar tool to Ttracemaintainer is ArchTrace [13], it addresses the consistency and evolution of trace links between software architecture models and their associated code. Another approach for evolving traceability for heterogeneous artifacts [11] gives interesting insights about which information should be traced for corresponding artifacts so that fine-grained differencing can be used to identify evolution. The graph-based traceability schemes exist in literature like [6, 15, 16]. Schwarz *et al.* [6] recommends the complete deletion of traceability links hence in this respect it is like our maintenance model, but it insists the trace maintenance using the technique based on [5], but essentially they are based on transformation models, while this paper is based on classical techniques. Some earlier works have recommend versioning schemes for traceability maintenance of artifacts [9], but with the versioning schemes it becomes hard to see the evolution at an instant. The other approaches are state-based [7], and scenario-based traceability. The state-based techniques employ syntactic differences between different versions of model. Some use text differencing to identify change. The other techniques for managing traceability, based on evolution, use policy-based support [10].

An important aspect of various traceability models is of the traceability recovery scheme. To reduce the cost of traceability, use of semi-automatic and automatic mechanism for traceability recovery is advocated. This is an important aspect, as for a fairly large sized project creating traces manually can be tardy.

ADAMS [16, 17] uses a latent semantic indexing scheme for traceability recovery from the checked in artifacts. There are many schemes based on IR (information retrieval) and vector space model techniques. The majority of traceability tools equipped with semi-automatic or automatic recovery techniques are plagued with ‘false positive’ problem [16]. The tool ADAMS uses an event notification scheme and claims automatic traceability recovery scheme and other modules for project management. It also uses a versioning scheme for traces, but still some information loss is still possible owing to complete removal of artifacts before the version release.

 ISBN: 978-1-61208-165-6

There are many traceability models, but most of the systems are overly complex and do not address the chronological evolution and information loss problem in particular. Valorizing traceability can be used as a tool in software configuration management [8].

IV. GRAPH-BASED TRACEABILITY MAINTENANCE MODEL

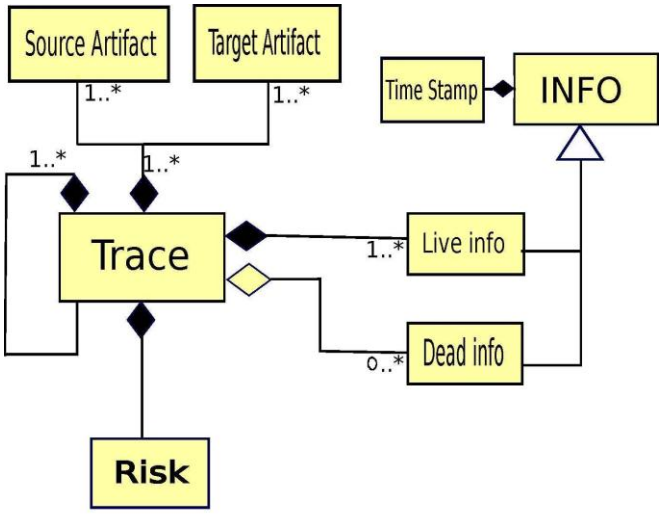

Figure 1 .Trace meta-model

Our graph-based traceability maintenance model is comprised of two entities: trace meta-model and traceability mechanism.

A. Meta-model

We propose our solution to the aforesaid problems; we assume that the information that a trace should contain are decided by traceability policies of the enterprise. We define our traceability meta-model, as shown in Figure 1. We have introduced the concept of live and dead information in our meta-model. Live information is one which is coherent till date and is represents the current state of artifact, whereas dead information is one which is obsolete with respect to current state of artifact but still holds information which shows the chronological evolution of system.

The trace meta-model defines trace as composition of other traces; a trace always contains at least one source and at least one target artifact. A trace contains two types of information live information and dead information.

Information is always associated with a time stamp indicating the period during which it was conceived or created. A trace should contain at least single live information and may not contain dead information. A trace always contains a risk associated apart from information. We recommend link model of [11] data to be taken in consideration for representing a trace information.

B. Traceability mechanism

Traceability mechanism is based on the graphical traceability techniques in which artifacts are represented as nodes and traces are links between the two or more artifact. The need of a product or product is considered as the root of the tree, non-functional requirements (NFRs) and functional requirements (FRs) are the immediate nodes to the root. As most of the NFRs are implemented as FRs, the NFRs are later linked to FRs and artifacts in next level at finer granularity.

In our traceability mechanism, we define three actions addition, modification, and rejuvenation; they can be applied both on traces and artifacts; there is no deletion operation but instead another sub-operation of modification called suspension. Suspension is envisaged to provide similar functionality like deletion, which permits to keep the track of trace evolution.

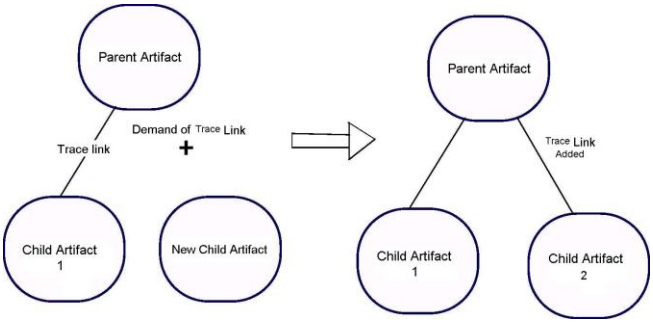

Figure 2. Addition operation

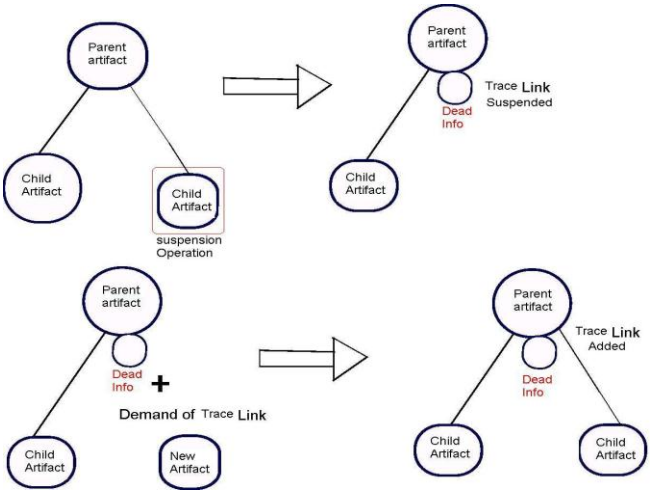

Figure 3. Modification-suspension operation

Each node/artifact maintains two additional lists, one for the dependencies or links, which are pointing to a dead artifact, and one which maintains the names of dead child artifacts.

1) Addition operation

Figure 2 shows the addition operation, when an artifact is created a trace is created pointing from the parent node to the recently created node. All the necessary data are filled and the node is initialized.

2) Modification operation

Modification operations are of two types change and suspension.

a) Change

In case of modification change operation whenever data are updated the earlier existing data are marked dead and the newer ones take their place and are marked alive.

b) Suspension operation

Modification-suspension operation is one when an artifact is no longer coherent with the current system state, and user actually wants to remove it, in this case the artifact is marked dead and is suspended and instead of complete deletion from tree it is moved one level up and is added to the list of dead artifacts of corresponding node. Figure 3 shows the modification–suspension operation. The other consequence to modification–suspension operation is that all the links from the various other artifacts which were pointing to dead artifact are added to the list of dead pointers.

3) Rejuvenation operation

A rejuvenation operation permits to change the status of a trace from dead to alive. This operation can only be applied when all the pre-artifacts to current artifact are alive or controlled, i.e., all the earlier artifacts which were the existential reason for the current artifacts should have been taken in account suitably.

V. DISCUSSION

In principle, the majority of graph-based traceability tools are more or less similar, plagued with similar deficiencies. We would recommend a semi-automatic traceability recovery technique. As, in a fairly large system a fully automatic mechanism can lead to false-positive notifications, which can be errant for requirement engineers. The current traceability mechanisms based on information retrieval (vector space models, latent semantic indexing, and probabilistic model etc.), structural rule-based, linguistically rule-based, transformation rule-based or other hybrid techniques are still error prone and needs to be improved.

Our traceability maintenance technique can be coupled with any traceability recovery technique, and used efficiently. Our paper addresses vital issue of information loss; for example, in a fairly large project which has duration of several years, it is possible that one artifact which was previously decided not to be included in the product owing to a certain constraint, is reintroduced. If the analyst had removed this artifact from system, the information regarding its exclusion was lost which was valuable to the project, and hence it costs again time and money, only to be discovered later regarding its deficiency. We claim that this 'artifact evolution information' is useful and should not be lost whether the decision regarding the artifact is finally affirmative or negative.

The major limitation of event-based traceability approach is of scalability; as the number of messages generated passes a certain limit, it becomes difficult to handle so many notifications manually [17]; even reduced subscription cannot answer this problem. This maintenance problem is addressed by our technique. The cost of maintenance using our technique is fairly less, as compared to other techniques. For every artifact updated, the information which is obsolete becomes part of the parent node in the form of dead information, and the pointing trace is also removed and stocked as dead information with parent node, this eases the work of requirement engineer. In a large project with an event-based notification procedure, using our proposed technique, the deletion operation on any artifact could be executed without the overhead of notifications, and overhead of follow-up trace deletion requests from lower level artifact owners to higher level artifact owners.

Our traceability model includes risk evaluation of every trace created, this helps to valorize the traceability task of requirement engineer. The risk involved can be the information vital information regarding the dependencies or the rationale behind the existence of the artifact. We claim that, this can help requirement engineer to valorize his work and renders the tracing activity interesting by coupling analysis together, which can be used later, for calculating ripple effect.

VI. CONCLUSION

This paper has presented a new approach for traceability maintenance scheme, trying to address chief problems of current trace processes. The proposed traceability model emphasizes on maintenance with efficient maintenance schemes, we are developing a tool which comprises our technique, and we are yet to obtain results and observations which support our claims. Our technique provides interesting solution to the dangling trace problem, which can immensely help to reduce the tediousness of tracing process. Our solution offers a plausible solution to the information-loss problem as the information ever generated in the development process remains in system to provide the exact trace of evolution of the system.

With the ease in trace maintenance process the cost of maintenance can be reduced noticeably as the dangling pointer problem is solved the effort in maintenance is reduced and hence less time and less human resources are engaged to do the same task.

We claim that our technique can bind tightly the low-end users to the traceability process and can help them to valorize their work by involving them in risk assessment process of every artifact they own. Usually in the system development process there are numbers of iterations before an artifact is finally accepted as the part of system, our technique allows retaining the information regarding iterations and chronological evolution and hence helps in better decision making.

We can still not trace 100% of information as it is always difficult to trace the informal aspects of many artifacts. We advocate the usage of semi- automatic trace mechanism with

 ISBN: 978-1-61208-165-6

event specific human intervention for the optimal benefits of traceability.

VII. FUTURE WORK AND PERSPECTIVES

We are currently working to fully implement our technique, which addresses maintenance issues which we discussed in this paper. In spite of these facts there are other issues which need to be addressed like heterogeneous traceability schemes for capturing informal aspects.

Usually graph becomes large and hard to understand [12], our technique can be constrained to map intra-level traceability, reducing size and increasing the understandability of graph. Our technique can be evolved further to enable global distributed traceability.

There are still issues like increasing the value of trace and methods to augment the usability of trace in organization and how to holistically link the various aspects of system development with the traces. Can we utilize traces for rapid development process? Can traceability patterns be used for product development? How to evolve traceability techniques as a tool for change impact analysis? These are the numerous issues which need to be addressed by research communities.

ACKNOWLEGEMENTS

The research leading to above results has received funding from the European Community' Seventh Framework Program (FP7/2007-2013) under grant agreement n° 234344.

REFERENCES

[1] Gotel, O.C.Z., and Finkelstein, C.W., "An analysis of the requirements traceability problem," Proceedings of the First International Conference on Requirement Engineering (ICRE 1994), pp. 94-101, 18-22 Apr-1994, doi: 10.1109/ICRE.1994.292398.

[2] "IEEE Recommended Practice for Software Requirements Specifications," IEEE Std 830-1998, 1998 doi:10.1109/IEEESTD.1998.88286.

[3] Ramesh,B., "Factors influencing requirements traceability practice," Commun. ACM 41, 12 (December 1998), pp. 37-44. doi=10.1145/290133.290147.

[4] Ramesh, B., and Jarke, M., "Toward reference models for requirements traceability," IEEE Transactions on Software Engineering, vol.27, no.1, pp. 58-93, Jan 2001 doi: 10.1109/32.895989.

[5] Cleland-Huang, J., Chang, C.K., Christensen, M., "Event-based traceability for managing evolutionary change," IEEE Transactions on software engineering, vol.29, no.9, pp. 796- 810, Sept. 2003, doi: 10.1109/TSE.2003.1232285.

[6] H. Schwarz, J. Ebert, and A.Winter., "Graph-based traceability: a comprehensive approach," Softw. Syst. Model. 9, 4 (September 2010), pp. 473-492. doi=10.1007/s10270-009-0141-4.

[7] N ,Drivalos-Matragkas; D.S. Kolovos. R. F. Paige; and K.J. Fernandes., "A state-based approach to traceability maintenance," Proceedings of the 6th ECMFA Traceability Workshop (ECMFA-TW '2010). ACM, New York, NY, USA, pp. 23-30. doi=10.1145/1814392.1814396.

[8] K.Mohan, P.Xu, LCao, B.Ramesh., "Improving change management in software development: Integrating traceability and software configuration management," Decision Support Systems, Volume 45, Issue 4, Information Technology and Systems in the Internet-Era, November 2008, pp. 922-936, ISSN 0167-9236, doi: 10.1016/j.dss.2008.03.003.

[9] T. N. Nguyen, C. Thao, and E. V. Munson., "On product versioning for hypertexts," Proceedings of the 12th international workshop on Software configuration management (SCM '2005), ACM, New York, NY, USA, pp. 113-132. doi=10.1145/1109128.1109137.

[10] A.Seibel, S. Neumann, and H.geise., "Dynamic hierarchical mega models:comprehensive traceability and its efficient maintenance," Softw. Syst. Model. 9, 4 (September 2010), pp. 493-528. doi=10.1007/s10270-009-0146-Z.

[11] Hong, Y; Kim, M; Lee, S-W., "Requirements Management Tool with Evolving Traceability for Heterogeneous Artifacts in the Entire Life Cycle," Proceedings of the Eighth ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2010), pp. 248-255, 24-26 May 2010 doi: 10.1109/SERA.2010.39.

[12] Winkler, S., and Pilgrim, J.V., "A survey of traceability in requirements engineering and model-driven development," Softw. Syst. Model. 9, 4 (September 2010), pp. 529-565. doi=10.1007/s10270-009-0145-0.

[13] Murta, L.G.P.,van der Hoek, A., Werner, C.M.L., "ArchTrace: Policy-Based Support for Managing Evolving Architecture-to-Implementation Traceability Links," Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE '06), pp. 135-144, 18-22 Sept. 2006 doi: 10.1109/ASE.2006.16.

[14] P.Mäder, O.Gotel, and I.Philippow., "Enabling Automated Traceability Maintenance through the Upkeep of Traceability Relations," Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA '09), LNCS 5562, pp. 174-189, doi: 10.1007/978-3-642-02674-4_13.

[15] Pinheiro, F.A.C., Goguen, J.A., "An object-oriented tool for tracing requirements," Proceedings of the Second International Conference on Requirements Engineering (ICRE 1996), pp. 219, 15-18 Apr 1996, doi: 10.1109/ICRE.1996.491449.

[16] De.Lucia, A., Fausto, F., Rocco, O., and Genoveffa, T., "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM* Trans. Softw. Eng. Methodol. 16, 4, Article 13 (September 2007). doi=10.1145/1276933.1276934.

[17] De.Lucia, A., Fausto, F., Rocco, O., and Genoveffa, T., "Fine-grained management of software artefacts: the ADAMS system.," Softw. Pract. Exper. 40, 11 (October 2010), pp. 1007-1034, doi=10.1002/spe.v40:11.

 ISBN: 978-1-61208-165-6