TRACEABILITY MANAGEMENT ARCHITECTURES SUPPORTING TOTAL TRACEABILITY IN THE CONTEXT OF SOFTWARE ENGINEERING

Héctor García, Eugenio Santos, Bruno Windels

Abstract: In the area of Software Engineering, traceability is defined as the capability to track requirements, their evolution and transformation in different components related to engineering process, as well as the management of the relationships between those components. However the current state of the art in traceability does not keep in mind many of the elements that compose a product, specially those created before requirements arise, nor the appropriated use of traceability to manage the knowledge underlying in order to be handled by other organizational or engineering processes. In this work we describe the architecture of a reference model that establishes a set of definitions, processes and models which allow a proper management of traceability and further uses of it, in a wider context than the one related to software development.

Keywords: Traceability, Software Architectures, Configuration Management, Software Maintenance.

ACM Classification Keywords: D.2.7: Distribution, Maintenance and Enhancement – Documentation; D.2.9: Management - Software configuration management

Conference: The paper is selected from Sixth International Conference on Information Research and Applications – i.Tech 2008, Varna, Bulgaria, June-July 2008

Introduction

The main goal in software traceability is to trace all the elements that can be considered relevant enough for the organization within a particular project or software product. Some classical examples of these elements are requirements, designs, source code or tests. However there is a number of information that is not considered carefully in current literature. Emails sent by stakeholders, minutes of meetings, project proposals or cost benefit analyses are documents that are also an essential part of a software product, retaining a great amount of knowledge that is required by organizations (e.g. in order to manage process improvement and capability determination [16]).

The capability to establish and maintain relationships between elements contained in these and other documents is essential, no matter the typology or stage during the product life cycle in which they arise. Managing all sort of relationships between whatever elements is what we define as total traceability. Some authors have previously discussed this term [14], although they have covered only aspects related to the engineering process.

In figure 1 below we illustrate the concept of total traceability that we try in our work. A software project does not start in the requirements engineering stage, as pointed by many authors [24],[29],[8],[20],[14]. Moreover, we should not consider requirements as the core in traceability, even when a wider perspective is considered, including information not directly related to engineering processes as described in [25].

We claim that software traceability should be focused in establishing relationships between the elements that compose the product, regardless of their type or stage in which they first appear. None of those elements should be considered the core or the start point in traceability. Only the specific purpose of the traceability itself, in a particular scenario, is to determine it.

Traceability should conform to an open and decentralized network in which we could include any element of particular interest in a given scenario.

Figure 1 shows a simplified hypothetic case, that we can easily find in the real world. The product life cycle starts when a user sends an email asking for some kind of software meeting some high level user requirements in order

to carry out with some tasks. After this email a project proposal or tender is prepared, covering those requirements, after a first approach to the problem. Later, a cost benefit analysis is developed, determining if it is viable to tackle the software. The solution to the problem of the user consists of developing specific software, carrying out a typical life cycle generating requirements specification, use cases definition, class diagrams and state charts, sequence diagrams and source code. In order to simplify the concept we avoided including a number of models and documents that can be considered, but in essence the main idea remains, all possible information, documents or models are candidates for tracing purpose.

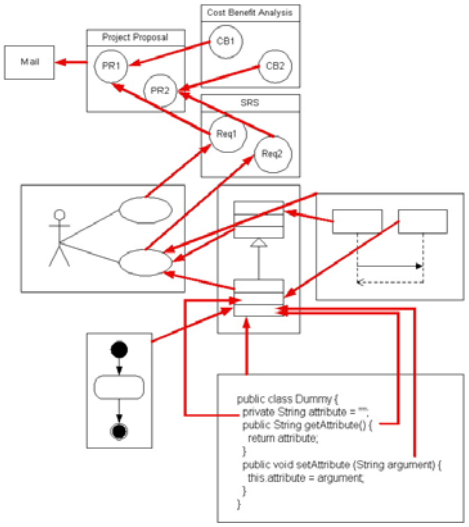

Figure 1 – Basic example on total traceability

Let us now imagine that, after carrying out the cost benefit analysis, the project is not developed because of the lack of resources. Then, considering a historical approach to traceability, all those emails and documents are not traced, so it could be potentially impossible to retrieve all knowledge underlying together with their relationships. What if we tackle the problem, again, after some time? Or if, also later, we decide to take it up again, using COTS? As long as the organization did not get to requirements engineering stage, no information on the project is available. How could we evaluate a number of commercial products or components? We should start everything from a scratch.

On the other hand, if we consider all these information, previous to requirements, we could trace our project proposal to the specifications given by vendors and take the project up again from the same point in which it was stopped.

Some of the documentation we are talking about is considered in [15] as essential to software life cycle, however nor this organization has treated these aspects with such a purpose, even in the new [18] or previously deployed [17], in which traceability is not considered as the essential part it is to software development.

May the reader appreciate that in some sense our work is not only related to software traceability, but also to knowledge management in the context of software development and maintenance, focusing traceability in the knowledge that documents, models or their relationships may provide in terms of the knowledge life cycle described by Birkinshaw and Sheenan [4].

To avoid misunderstandings, in this paper we will describe element as any item, under configuration management within an organization, and which may be related, or even not, to a software product in any sense. Also we will describe trace as any kind of traceability link in the widest semantics of the term.

In our work we considered the following hypotheses, which determine the context in which TRAMA is to be understood and that we discussed in [11]:

H1: The lowest level of granularity in traceability shall agree to the granularity established in Configuration Management.

H2: It shall be established a common framework regarding Configuration Management and Traceability from an organizational point of view, considering needs and goals of the organization.

H3: The products generated during the software life cycle can be modeled as structured and processable documents.

H4: Those tools used during all product life cycle and related tasks allow to make information persistent, in processable and structured formats, such as XML technology. In other case, such formats can be obtained through exports or digital processing.

Traceability Management Architecture

The basic components of the proposed architecture for traceability management systems are to support a set of operations that we shall enumerate below. We distinguish four main components: documentation and traceability data model support, traceability management, information retrieval and interfaces to external tools.

The data models define structures that allow making persistent all that information required to trace elements and managing traceability.

The traceability management component supports the basic operations allowed within traceability, using the data in the data models.

Information retrieval provides the capability to search and locate the information, as well as to infer new relationships and generate candidate lists.

Interfaces to external tools allow integrating the traceability management system to other applications used within the organization in order to automate the different tasks related, closely or not, to software projects.

In figure 2 below the different components of which TRAMA is composed, and some relationships to software life cycle processes [15], are shown.

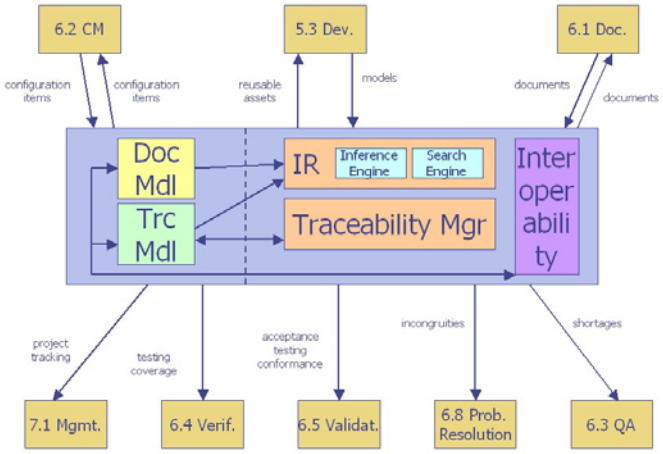

Figure 2 – Basic architecture for a traceability management system

Regarding data models we suggest to implement two different components, establishing proper relationships between them.

Documentation model is in charge of storing all information related to configuration items (e.g. projects, documents, sections of documents). Traceability model supports the storage of the traces established between configuration items.

Separating each model provides the capability of integrating configuration management systems from third parties. The purpose is to manage separately the information managed by external tools from the information managed by the traceability management system, which is the system our work is focused on.

In our implementation documentation model takes into account, amongst others, the following configuration items:

- Projects and project stages, as well as staff assigned to the different tasks. These data retain information about planning, allowing to generate project tracking reports.
- Documents and sections of documents represent a model of the documentation of a project and the structure of documents.
- Software artifacts, such as requirements and design, coding and testing artifacts represent the deliverables from engineering tasks in the software life cycle.
- Deliverable templates related to each task in a project support quality assurance and project tracking reports.
- Stakeholders involved in tasks and documents that shall receive the deliverables.

Meanwhile, the traceability model is in charge of storing the traces that establish the relationships between configuration items. A partial list of traces is described below.

- Dependency traces model the relationship between configuration items, where a configuration item is dependent on other. We provide a number of semantics for dependency (e.g. parent to child, aggregation, complement) as well as degrees for the dependency (e.g. full, partial).
- Document and section traces establish the contents of documents and their structure. Contents are always configuration items (e.g. software artifacts, text, figures).
- Rationale traces allow attaching an explanation to a configuration item.
- Output traces establish relationships between tasks and document templates.
- Drive, Implementation, Test and Verification traces allow attaching different software artifacts to other. Usually these traces are useful to generate quality assurance reports, as well as to support Change Management.

The information retrieval component is responsible of locating configuration items that match the criteria given by a user search. User search are defined as patterns that depend on the search type. We provide the following search types to support different goals:

- Locate individual configuration items matching a pattern, such a rationale, content or name.
- Locate configuration items involved in traces to a given configuration item.
- Locate configuration items involved in a specific kind of trace to a given configuration item. This is particularly useful to calculate impact trees derived from a change (i.e. to support change management).
- Locate software artifacts that satisfy a set of requirements. This feature has been introduced to support software reuse.

It is also feasible to infer knowledge from the information provided by traces. The main goal is to find new traces between configuration items, in order to reduce the effort of software engineers while carrying out traceability tasks. This feature is also important to find incongruence in the models of a given project, and to locate traceability lacks, as well as to support quality assurance tasks. These are the reasons why we defined two different components, Inference Engine and Search Engine.

Traceability Management component is in charge of carrying out the functionalities defined in the reference model, excepting information retrieval and interoperability. Some of these functionalities are:

- Create and delete repositories
- Create and delete projects
- Replicate elements and reuse assets
- Create, alter and delete traces
- Create, alter and delete documents and sections

Interoperability component supports the operations which allow integrating the traceability management system to external tools (e.g. CASE, configuration management or documentation tools). Information exchange is based on import and export features through XML and XMI files. An improvement of the features in which we are working on is to create web services. Web services could make feasible to share information in different tools in soft real time.

Related Work

Traceability usefulness, in the area of Software Engineering, has been demonstrated by most related literature. Processes such as change management obtain substantial benefits from traceability [10],[6]. Traceability links allow establishing relationships between different items, or knowledge assets, and they are of interest for the organizations [26]. The capability to reuse software assets, such as documents, models or code, is also a question closely related to traceability [7], as well as domain and product analysis [8].

However a high effort and cost is inherent to traceability management, as described in [9], and new perspectives on which elements are more relevant or require more detailed attention, in order to lower the effective costs are arising, such as the Value Based Software Engineering [5].

The most popular choice when automating traceability consists of the development of systems and frameworks stating clearly the information related to traceability links, and the way to implement them in a standardized manner, instead of depending on specific features provided by vendors.

Sherba et al. [28] provide a traceability management system consisting of the integration, through parsers, of different tools, sharing the project components and models in a common repository. The main problem is to maintain this duplicated and coupled structure. In [3] we can find some suggestions on how to avoid the problem, together with a compromise on the granularity or detail level required in this kind of systems.

More efforts in this sense have conducted to establish standardized formats for documents and models, most of them based in the eXtensible Markup Language, such as XML Metadata Interchange [23].

In [19] a metamodel for traceability management and a set of processes related to software traceability, based in patterns, can be found. Alarcón et al. [1] describe a software engineering environment, considering an integrated traceability system, in which documents generated within the environment are stored in an XML compatible format. Also many efforts in introducing tags within the source code, to provide traceability, have been described in earlier years [12].

Alves-Foss et al. [2] suggest the use of XMI to represent UML designs, and JavaML for the source code. A set of DTDs and transformations make it feasible to translate the models into source code and vice versa.

Another significant problem in traceability systems is to determine the proper information retrieval and processing. Huffman et al. [13] applied information retrieval techniques to create automated candidate traceability link lists. Marcus et al. [22] used latent semantic indexing to detect links between product documentation and source code, and Spanoudakis [29] established a set of heuristic rules to analyze links between different elements that resulted in patterns to determine which candidates where valid.

Regarding elements and relationships, the best analysis we can find is that one by Ramesh and Jarke [25], in which we can find a complete classification on traceability links and the data that should be considered in reference models. Different classifications, which provide more information and types of links can be found in [21], with the goal of supporting conformance analysis and inconsistency finding. In [20] we can find a model to support traceability management for UML projects, including rationales and stakeholders, as well as many software artifacts.

Tryggeseth and Nitro [30] classify the relationships in different categories keeping in mind a double structure related to application structure and documentation, while Riebisch [27] takes into account the link types depending on the structure of requirements documents. Von Knethen [31] establishes a difference between traces linking elements in the same abstraction level and those between elements in different abstraction level. Sherba et al. [28] describe some examples of links that are useful to determine some of the types of relationships between elements.

Relating the knowledge that should underlie to each link, Ramesh and Jarke [25] suggest to consider six dimensions: *What?, Who?, Where?, How?, Why?* y *When?*.

Conclusions

Introducing traceability as a part of the methodological definition of the development process could help in avoiding the historical problems of traceability application, marginally considered in the Software Engineering processes. The support to other organizational processes could result in decreasing the effective costs on applying traceability. The specification of ISO 24744 [18] could have been a nice chance to introduce traceability as an essential part during the definition of software development methodologies. Integrating traceability in such a standard, which considers all software process aspects, from documentation to tasks, as well as human resources involved, should have supposed a great step.

If we expect to reach such an ambitious goal, it is necessary to track any knowledge asset. It supposes to include, under configuration management, any element that shall persist in time. Then, it becomes necessary to extend configuration management to all areas in the organization, not only to those related to engineering processes. In this sense it is essential to establish a common framework in the organization regarding configuration management and traceability, considering also the proper minimum granularity required.

We also discussed how information contained in a traceability management system is not only useful for software processes, but also to other processes. Multiple uses of this information, especially through information retrieval and data mining, will result in long and short term benefits in organizations. In such case scenario it could be worthwhile to implement and introduce traceability management in the industry, as well as lowering the effort required to manage and maintain these kinds of repositories.

Bibliography

[1] Pedro P. Alarcon et al. Automated Integrated Support for Requirements-Area and Validation Processes Related to System Development. Proceedings of the 2nd International Conference on Industrial Informatics, IEEE. 2004, pp. 287-292.

[2] Jim Alves-Foss, Daniel Conte de Leon y Paul Oman. Experiments in the Use of XML to Enhance Traceability between Object-Oriented Design Specifications and Source Code. Proceedings of the 35th International Conference on System Sciences, pp. 3959-3966. IEEE. 2002.

[3] P. Arkley, P. Mason, S. Riddle. Position paper: Enabling Traceability. Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 61–65. ACM. 2001.

[4] J. Birkinshaw, T. Sheenan. Managing the Knowledge Life Cycle. Engineering Management Review, volume 31, issue 3, p. 19. 2003.

[5] Barry Boehm. Value-Based Software Engineering. Software Engineering Notes, vol. 28, issue 2, section Article abstracts with full text on line, pp. 3. ACM. 2003.

[6] L.C. Briand, Y. Labiche, L. O'Sullivan. Impact Analysis and Change Management of UML Models. Proceedings of the 19th International Conference on Software Maintenance. IEEE. 2003.

[7] Andrea de Lucia et al. Enhancing an Artefact Management System with Traceability Recovery Features. Proceedings of the 20th International Conference on Software Maintenance, pp. 306-315. IEEE. 2004.

[8] Alexander Egyed, Paul Grünbacher. Automating Requirements Traceability: Beyond the Record & Replay Paradigm. Proceedings of the 17th International Conference on Automated Software Engineering, pp. 163-171. IEEE. 2002.

[9] Alexander Egyed, et al. A Value-Based Approach for Understanding Cost-Benefit Trade-Offs During Automated Software Traceability. Proceedings of the 3rd International Workshop Traceability in Emerging Forms of Software Engineering, pp. 2-7. ACM. 2005.

[10] Stephen G. Eick et al. Does code decay? Assesing the Evidence from change management data. IEEE Transactions on Software Engineering, vol. 27, no. 1. IEEE. 2001.

[11] Hector Garcia. Documentation and Traceability in Software Projects (in spanish). Research Work. Carlos III University of Madrid. 2007.

[12] Ernesto Guerrieri. Software Document Reuse with XML. Proceedings of the 5th International Conference on Software Reuse, pp. 246-254. IEEE. 1998.

[13] Jane Huffman Hayes, Alex Dekhtyar, James Osborne. Improving requirements tracing via information retrieval. Proceedings of the 11th International Conference on Requirements Engineering Conference, pp. 138-147.IEEE. 2003.

[14] Suhaimi Ibrahim et al. Implementing a Document-Based Requirements Traceability: A Case Study. Proceedings of the 9th International Conference on Software Engineering and Applications, pp. 124-131. IASTED. 2005.

[15] International Organization for Standardization. Information technology – Software life cycle processes. ISO/IEC 12207:1995. ISO. 1995.

[16] International Organization for Standardization. Software Process Improvement and Capability Determination. Juego de normas ISO/IEC 15504. ISO. 2003 and 2004.

[17] International Organization for Standardization. Information technology – Software Engineering Environment Services. ISO/IEC 15940:2006. ISO. 2006.

[18] International Organization for Standardization. Software Engineering – Metamodel for Development Methodologies. ISO/IEC 24744:2007. ISO. 2007.

[19] Justin Kelleher. A Reusable Traceability Framework using Patterns. Proceedings of the 3rd International Workshop in Emerging Forms of Software Engineering, pp. 50-55. ACM. 2005.

[20] Patricio Letelier. A Framework for Requirements Traceability in UML-based Projects. Proceedings of the1st International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 32-41. ACM. 2001.

[21] Jonathan I. Maletic et al. Using a Hypertext Model for Traceability Link Conformace Analysis. Proceedings of the 2nd International Workshop Traceability in Emerging Forms of Software Engineering, pp. 47-54. ACM. 2003.

[22] Andrian Marcus, Jonathan I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. Proceedings of the 25th International Conference on Software Engineering, pp. 125-135. IEEE. 2003.

[23] Object Management Group. MOF 2.0 – XMI Mapping Specification, v. 2.1. OMG. 2005. Available at <http://www.omg.org/technology/documents/formal/xmi.htm> [ref. february 11th, 2007].

[24] Balasubramaniam Ramesh. Factors Influencing Requirements Traceability Practice. Communications of the ACM, vol. 41, issue 12, pp. 37-44. ACM. 1998.

[25] Balasubramaniam Ramesh, Matthias Jarke. Towards Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, vol. 27, issue 1, pp. 58-93. IEEE. 2001.

[26] Balasubramaniam Ramesh. Process Knowledge Management with Traceability. IEEE Software, vol. 19, issue 3, pp. 50-52. IEEE. 2002.

[27] Matthias Riebisch. Supporting Evolutionary Development by Feature Models and Traceability Links. Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, pp. 370-377. IEEE. 2004.

[28] Susanna E. Sherba, Kenneth M. Anderson, Maha Faisal. A Framework for Mapping Traceability Relationships. Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 32-39. ACM. 2003.

[29] George Spanoudakis. Plausible and Adaptative Requirements Traceability Structures. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, pp. 135-142. ACM. 2002.

[30] Eirik Tryggeseth, Oystein Nitro. Dynamic Traceability Links Supported by a System Architecture Definition. Proceedings of the International Conference on Software Maintenance, pp. 180-187. IEEE. 1997.

[31] Antje von Knethen. A Trace Model for System Requirements Changes on Embedded Systems. Proceedings of the 4th International Workshop on Principles of Software Evolution, pp. 17-26. ACM. 2001.

Authors' Information

Héctor García – Adjunct Professor. Technical University of Madrid. E.U. Informática. Ctra. de Valencia Km. 7. E28031 Madrid. e-mail: hgarcia@eui.upm.es

Eugenio Santos – Professor. Technical University of Madrid. E.U. Informática. Ctra. de Valencia Km. 7. E28031 Madrid. e-mail: esantos@eui.upm.es

Bruno Windels – Researcher. Technical University of Madrid. E.U. Informática. Ctra. de Valencia Km. 7. E28031 Madrid. e-mail: bwindels@eui.upm.es