

Multi-disciplinary perspective on Knowledge Quality: Dimensions of Knowledge Quality

F. Mili, L. Blackwell, A. Gokani
Computer Science and Engineering
Oakland University
Rochester, MI48309-4478

Abstract

In a knowledge economy, knowledge repositories are being created, maintained, and disseminated by most organizations. While the major technical issues of storage and access have been solved, the issue of quality control and quality assurance on the contents are still relatively unexplored. In this paper, we propose a general framework of information systems quality. The framework is articulated around three dimensions of knowledge quality: logical, structural, and perceptual. Each of the dimensions is discussed in turn to provide a guide for formulating quality properties for knowledge repositories as well as for other types of information systems.

Introduction

The case for knowledge management no longer needs to be made. The potential benefits of capitalizing on corporate knowledge and managing it are generally acknowledged and accepted (e.g., (2; 5; 9; 17; 19; 20; 25).) Although there have been published success stories, the operational aspects of knowledge management are still being researched, experimented with, and refined. Knowledge management encompasses the activities of elicitation, cultivation, broadcast, and to some extent, the use of the knowledge. The electronic and asynchronous mode of publication of the corporate knowledge is almost an inherent aspect of the knowledge management movement. The ease of collection, review, update, and communication needed for corporate knowledge all make its electronic storage and electronic broadcast a necessity. On the other hand, the on-line on-time feature of these electronic repositories raises issues of quality and reliability.

We have, in (15; 16), documented the need for a systematic, tool-supported quality assurance, and shown how such quality monitoring and enforcement is feasible within the context of knowledge repositories containing engineering design standards, regulations, and best practices. We have developed and implemented a knowledge management system that monitors knowledge quality and supports the user in identifying and

executing potential fixes. In the process of carrying out this project, we identified issues and solutions that are of interest beyond the confines of the specific application studied. In this paper, we share some of the insight gained, and organize it into a general framework for knowledge quality. In particular, we categorize quality into three different dimensions, the logical, the structural, and the perceptual. For each of these dimensions, we identify the main motivation of quality, the general underlying principle, and the application specific parameters that shape it and give it a different flavor in different application domains.

The case for Quality categorization

Although many researchers within the knowledge management community have acknowledged the lack of systematic, comprehensive quality assurance (23), few concrete proposals for pallying to this situation exist. Yet, the issue of quality and quality assurance has been researched from many angles in related areas. Theoretical results, validation support tools, and effective processes have been developed in the related disciplines of software engineering, database management, and artificial intelligence. A first examination of these results lends an impression of a great diversity and divergence in what constitutes quality and what motivates quality assurance. The approaches used in these different disciplines tend to be application-specific, language-specific, semantic-specific, and, at first examination, provide with little room for generalization and reuse. The terminology and concepts used in the different fields cover a wide spectrum including such qualities as completeness, correctness, competence, convergence, timeliness, and minimality, to cite only a few (6; 13; 14; 23). In this paper, we seek to identify some unity in the form of common underlying themes behind the variety of approaches and concerns. In particular, we define a framework of quality by identifying three complementary dimensions of quality. All three dimensions are relevant in all fields. The difference lies in their relative importance and criticality. Different fields of information science have focussed on different dimensions. This framework is useful as a guide for developing quality properties in knowledge repositories as well as in other

fields.

The Logical dimension of quality

The ultimate quality of any informational item is its correctness and veracity. The correctness of a computer system is defined relative to a specification or expectation. Program verification can be fully formalized, proved, and maintained (14). For data and knowledge repositories, veracity rather than correctness capture the logical quality. Veracity characterizes a knowledge or data repository that is a faithful reflection of the world that it is meant to represent, i.e. the repository

“states the truth, all the truth, and nothing but the truth.”

Such utopic criterion, although highly desirable, is generally impossible to verify or enforce as it makes reference to an external unaccessible entity. In other words, because veracity is an extrinsic quality, it cannot be established internally. It can only be approximated with other intrinsic properties. In practice, intrinsic criteria are identified and used instead. By being intrinsic, these criteria can be assessed without referring to some outside entity, but they are only indicators of quality. Their absence (when they are not met) is indicative of lack of quality, while their presence only gives some level of confidence in the quality of the artifact but does not constitute a full warranty. For example, when considering a formal specification, the presence of a logical “fault” such as inconsistency (7), incompleteness (6), divergence (13), or safety violation (10) is indicative of lack of quality, yet a specification with none of these faults is not necessarily valid. Similarly, the presence of type mismatches in a Pascal program are an indication of a problem, yet the absence of mismatches tells us very little.

In general, given an information system expressed in some model M , the intrinsic logical qualities of the system are the expression of the absence of violation of the model’s axioms and rules. In other words, the quality properties are anchored in a set of axioms that are inherent to the model or the language. We illustrate this from a variety of systems.

Relational Databases: In the relational model of databases, there are two primary axioms intrinsic to the model, the entity integrity, and the referential integrity, stating respectively, that relations are sets, and thus allow no repetitions; and that associations must be expressed with valid references. In addition to the models axioms (or integrity constraints as they are generally called), the model allows domain-specific, user-defined integrity constraints. The expression of these constraints is of interest to the extent that the system (DBMS) automatically monitors and enforces these quality axioms.

Object Oriented Databases: The object oriented model has much richer semantics than the relational model. As a result, it is associated with a larger

number of axioms. The axiomatization of the object model received particular attention within the domain of database schema evolution. Because object databases are used in domains where the schema is expected to change frequently, the need for a system supported monitoring and enforcement of some level of schema correctness is critical. All proposals related to object schema evolution center around the definition of a set of axioms (or invariants) (8; 11; 22; 24). Most of these logical consistency properties center around the classification hierarchy ensuring it remains a partial order, and that classification associations are consistent with its axiomatization.

Knowledge Bases: Knowledge bases can be seen as formal versions of task-specific knowledge repositories. Even though they are represented in a formal language and benefit from careful, manual crafting, knowledge bases do also raise the need for an automatic monitoring of quality. In addition to consistency with the semantics of the formal logics, other qualities of completeness (coverage) have also been used.

Knowledge Repositories: When the knowledge is represented in an informal language, inconsistencies and contradictions are hard to detect. In (16), we have used an augmented object model to represent engineering design knowledge, and defined a set of axioms defining the semantics of the model.

In summary, logical quality is the extent to which the artifact is expressed “correctly” within the model or language. The extent to which logical quality can be enforced and monitored is a direct factor of the extent to which the language or model is formalized. It is much harder to detect a contradiction in a statement expressed in some natural language, for example, than it is to detect a contradiction in the same statement expressed in a formal language. When the model’s axioms are formal, the monitoring of quality can be performed in an automatic manner and integrated within the knowledge management tool.

The Structural dimension of quality

In the same way that logical quality is the first thing that comes to mind when we think of knowledge base quality, structural quality is the first that comes to mind when we think of software quality. Structural quality is also prevalent in database. Structural quality is to a large extent an evolution- and maintenance-related quality. When systems or artifacts are meant to be designed once, validated, and then used forever unchanged, logical quality is all that is needed. For software artifacts, however, this is hardly ever the case. As a result, there are generally at least two layers of quality that are used. The first layer, logical quality reflects the fact that a system is correct/valid/operational. The second layer, structural quality, is needed to reflect the fact that the logical quality of the artifact is relatively

easy to preserve as the system evolves and changes over time.

In the programming realm, software quality refers to such qualities as readability, modularity, abstraction, high cohesion of individual units or subsystems, and low coupling between different units. In software engineering, we find the same concepts and themes recurring at a larger scale. The concepts of high coherence and low coupling are applied at the subsystems level. The concepts of modularity and abstraction are prevalent as well, promoting reusing existing artifacts and building reusable ones (4; 6; 3).

In the area of database, where data repositories are updated by users who have limited views of the overall contents of the database, the system's oversight over the quality of the data is one of its most important missions. As a result, the concept of structural quality is well developed. In relational databases, structural quality is defined as the lack of redundancy. The lack of redundancy is reflected by the general guiding principle that "each item of information figures in one and only one place." This requirement ensures that a change in one item in the database will not conflict with other information elsewhere within the same database (1).

The above discussion identifies two subthemes within the structural dimension of quality: modularity, and absence of redundancy. They reflect two complementary approaches to managing the unavoidable interdependencies that exist within a system. Modularity manages inter-dependencies by clustering related information together and separating it from other relatively independent information. Elimination of redundancy manages inter-dependencies by taking advantage of the expressive power and the inference power of the language to capture the same information in the most economical way. We discuss each of these two complementary sub-themes briefly.

Modularity is the hardest of the two to formalize and quantify in general. There are two dimensions to modularity: The language or model in which the system is expressed must support modularity; and the designer must make use of this facility. The underlying guiding principle is to divide the system into relatively self contained and reusable subsystems, and maximize the locality of changes. Different programming paradigms emphasize different conceptual modules (functions vs. objects for example). Individual modules can be analyzed, reasoned about, and modified relatively independently of the context in which they are used. This requires systems to have high cohesion (to be self contained) and high level of abstraction (to be reusable). These same qualities also contribute to the locality of changes. The added characteristic of low coupling further enhances locality. Software engineering metrics are used to quantify these qualities (12).

In summary, the general guideline in modularity is to maximize self- containment of components and maximize locality of changes. In some applications, quantified or qualified parameters can be defined to character-

ize modularity. In those cases this quality criterion can be automatically assessed and monitored by a system.

The second subtheme of structural quality, i.e. elimination of redundancy, has generally been captured more formally. This may be in part, because it has been applied to more formal structures such as databases and knowledge bases. The guideline of eliminating redundancy would be trivial if redundancy were limited to the replication of identical information. Generally, redundancy occurs in more subtle forms. In relational databases, a data item d is deemed redundant if it already exists in the database, or if it can be derived from other data in the database. A set of inference rules based on the concept of functional dependencies (reflexivity, transitivity, augmentation) is defined. These rules are used to define the informational content –implicit as well as explicit– of a database. A data item d is then deemed redundant with respect to given database if it can be derived from the data explicitly stored in the database. In knowledge base systems, absence of redundancy is also used as a quality criterion for knowledge.

The fact that the definition of absence of redundancy is intimately tied to the inference rules of the model or language used is a reflection of the fact that the richer are the semantics of the model, the more economical is the expression of the information. In other words, the more one can infer from a statement, the fewer statements one has to make. Informal languages do have a high expressive power, but the derived information is often ambiguous. In the realm of knowledge management, knowledge is often expressed in semi formal languages. The detection of redundancy or contradiction cannot be done with certainty. Heuristics can be used to assist users in detecting and correcting problems.

In (15), we have defined a set of "normal forms" for the engineering design knowledge based on the rules of inferences dictated by the associations used. In addition to the classification association and its inheritance, we have used part-feature and assembly-component associations.

In sum, the underlying principle of elimination of redundancy is to ensure that "each item of information is expressed in only one place." This is materialized by the definition of

- A set of inference rules that dictate how to derive new information from information explicitly stored.
- A corresponding set of quality criteria capturing the absence of redundancy.

The Perceptual dimension of quality

This third dimension is relevant in those cases where the artifact is "read" by humans. This is generally the case in knowledge management settings. Understandably, perception is the least quantified and the least explored of the three. Even though it can be totally subjective, this dimension is critical in a knowledge management setting where corporate workers are expected to trust

a knowledge repository and use it as a basis for making decisions.

Whereas in the other two dimensions we were able to draw on research and experience from other software disciplines, we have found little to build on for this dimension. Our study of this dimension draws primarily on our experience with validating and testing an engineering design knowledge repository with its users. We have inventoried requests and inquiries from users and used them to outline a set of perceptual quality criteria. We have found, for example, that users liked predictability and repetition. For example, users expressed the desire to see the same classification hierarchy for similar parts. This has led us to artificially inflate some of the hierarchical structures to provide users with a more uniform and more symmetric view of the knowledge repository. This led to a competition between the two criteria of structural quality, dictating to represent information minimally, and perceptual quality, dictating to represent information in the way that the users expect. In our experience, we have managed to provide users with their preferences by creating external views without disrupting the stored version of the knowledge.

Summary, Conclusion

The issues of knowledge quality and user's perception of knowledge quality are sine qua non conditions of success of knowledge management. Because knowledge repositories are designed to be real-time reflections of the state of the art and the state of the knowledge, quality is best controlled -at least in part- in a systematic, system-supported manner. This position paper presents a framework for defining, monitoring, and controlling quality. It ties together research work on quality from different software disciplines, and builds on practical experience to identify relevant issues.

Acknowledgements

This research has been supported by the Daimler Chrysler Corporation and the Michigan Research Excellence Fund.

References

- [1] S. Abitebul, R. Hull and V. Vianu *Foundations of Databases* Addison Wesley 1995.
- [2] E. Boling, W. Cai, J.P. Brown, and J. Bolte "Knowledge Base Development: The life cycle of an item in the Indiana University knowledge base" *Technical Communication*, 4th quarter, 2000, pp.530-543.
- [3] E. J. Braude *Software Engineering: An Object Oriented Perspective* John Wiley & Sons. Pub. 2001.
- [4] F. P. Brooks "No silver bullet: Essence and accident of software engineering" *IEEE Computer* (20)4, 1987.
- [5] Chun Wei Choo. *The Knowing Organization*. Oxford University Press, 1998.
- [6] P. Devambu and M.A. Jones "The use of description logics in KSBE systems" *ACM Transactions on Software Engineering and Methods*, (6)2, 1997, pp.141-172.
- [7] J. Grundy and J. Hosking "Inconsistency management for multiple-view software development environments" *IEEE Transactions on Software Engineering* (24)11, November 1998, pp. 960-981.
- [8] A. Formica *et al* "An efficient method for checking object-oriented database schema correctness" *ACM Transactions on Database Systems* (23)3, 1998, pp. 333-369.
- [9] W. E. Halal, editor. *The infinite resource: Creating and leading the knowledge enterprise*. Jossey-Bass Publishers, 1998.
- [10] C. Heitmeyer *et al* "Using abstraction and model checking to detect safety violations in requirements specifications" *IEEE Transactions on Software Engineering* (24)11, 1998, pp. 927-948.
- [11] W. Kim and H-T Chou "Versions of Schema for Object-Oriented Databases," *Proceedings of the 14th VLDB conference, 1988*
- [12] M. Klein *et al* "Attribute-Based Architecture Styles", *Proceedings of the First Working IFIP Conference on Software Architecture*, San Antonio, Texas, 1999, pp. 225-243.
- [13] A.V. Lamsweerde, R. Darimont and E. Letier "Managing conflicts in goal driven requirements engineering" *IEEE TSE*, (24)11, 1998.
- [14] A. Mili, J. Desharnais, F. Mili *Computer Program Construction*. Oxford Press, 1994.
- [15] F. Mili, K. Narayanan, V. Atluri. Defining and Monitoring Knowledge Integrity. In *Proceedings of the FLAIRS Conference, May 2000*, Orlando, Florida, AAAI Press.
- [16] F. Mili *et al* "Knowledge Modeling for Design Decision," *Journal of AI in Engineering*, Kluewr Pub. 2001.
- [17] Rick Mullin. Knowledge management: A cultural evolution. *Journal of Business Strategy*, September/October 1996.
- [18] Krishnakumari Narayanan. *Knowledge Modeling for Engineering Design*. PhD thesis, School of Engineering, Oakland University, 2000.
- [19] Ikujiro Noraka and Hirotaka Takeuchi. *The knowledge-Creating Company*. Oxford University Press, 1995.
- [20] Daniel E. O'Leary. Enterprise knowledge management. *IEEE Computer*, 31(3):54-61, March 1998.
- [21] Daniel E. O'Leary. Developing a theory-based ontology for "best practices" knowledge bases. In *AAAI Workshop on Knowledge Management*, 2000.

- [22] R. Peters and Özsu, M. "An Axiomatic Model of Dynamic Schema Evolution in Objectbase Systems" *ACM Transactions on Database Systems*, (22)1, pp. 75-114, 1997.
- [23] R. Ruggles. Knowledge tools: Using technology to manage knowledge better. Technical report, Ernest & Young LLP Center for Business Innovations, April 1997.
- [24] M. Snoeck and G. Dedene "Existence dependency: the key to semantic integrity between structural and behavioral aspects of object types" *IEEE TSE* (24)4, April 1998, pp.233-251.
- [25] Sidney G. Winter. Knowledge and competence as strategic assets. In Treece David, editor, *The Competitive Challenge: Strategies for Industrial Innovation and Renewal*. Harper & Row Publishing, New York.