

A Software Engineering View of Data Quality

Mónica Bobrowski, Martina Marré, Daniel Yankelevich

Departamento de Computación, FCEyN, Universidad de Buenos Aires, Argentina

{monicab,martina,dany}@dc.uba.ar

Abstract

Thirty years ago, software was not considered a concrete value. Everyone agreed on its importance, but it was not considered as a good or possession. Nowadays, software is part of the balance of an organization. Data is slowly following the same process. The information owned by an organization is an important part of its assets. Information can be used as a competitive advantage. However, data has long been underestimated by the software community. Usually, methods and techniques apply to software (including data schemata), but the data itself has often been considered as an external problem. Validation and verification techniques usually assume that data is provided by an external agent and concentrate only on software.

In this work, we present different issues related to data quality from a software engineering point of view. We propose three main streams that should be analyzed: data quality metrics, data testing, and data quality requirements in the software development process. We point out the main problems and opportunities in each of them.

Keywords: Software Quality, Data Quality, Software Engineering.

1. Introduction

Thirty years ago, the software owned by an organization was not considered a concrete value. Everyone agreed on the importance of software, on its virtual value, but it was not considered as a good, as a possession. In those days, the value of software was defined by its cost.

Nowadays, software is part of the balance of an organization, it contributes to its value, and for almost every software project the ROI is calculated. Data is slowly following the same process. In fact, people is now talking about “the value of information.” Many organizations want to possess information. Managers know that having the right information at the right time may lead them to obtain great benefits. Moreover, organizations have information that may help them to improve their work, make decisions, and increase their profits. This information is usually stored in large databases accessed via software applications. However, it is not enough to have good applications; an organization needs good data in order to achieve its goals.

Now, how could an organization know that it has the right information at the right time? How could an organization evaluate its information? That is a matter of *data quality*. In fact, the quality of information is crucial when determining its usefulness. When quality is not achieved, information is not used, or leads to incorrect decisions, and even loss. As it is known, “decisions are no better than the data on which they are based” [Red98]. But, what does information quality mean?

In recent years, researchers have been studying data quality problems from the perspective of the data generation processes [SLW97, SLW97, WW96]. They have identified problems in data, and tried to associate them with problems in the process that lead to this data. The underlying idea is that improving the process

may lead to an improvement in data.

In [Red96], Redman gives a deep introduction to DQ issues. He points out many aspects of data quality: definition, management, policies, experiences, requirements, measurements, etc. Although his approach differs from ours (it is mainly statistical, and concentrate on the data generation process), his book offers clear examples, motivations, definitions, and useful tips.

Data quality problems are well known to practitioners. In fact, many failures of software are not due to poor quality of the systems, but to inconsistencies or other problems in data. The quality of data has a great impact on the usefulness and overall quality of software systems.

However, the mainstream of Software Engineering ignored data quality issues up to day. Validation and verification techniques exist and have been used for software processes and products, but few has been done related to data. The only concern for computer engineers regarding data quality has been the extraction of data for data warehouses. In the context of Data Warehousing, an European project investigated quality and in particular the requirements on data needed to implement a data warehouse [JV97].

In our view, software engineers must take into account data quality issues in the design, validation, and implementation of software systems. Moreover, standard techniques can and should be applied to these problems. In this paper, we present different issues related to data quality from a software engineering point of view. We point out three main streams that should be analyzed and the main problems and opportunities in each of them. These themes are:

- Data quality metrics

Measuring the quality of the information will help us to know its value, and also its pitfalls. We will be able to know how valuable our information is, and also what we need to improve in order to increase quality. Moreover, measuring quality would clarify the goals of a quality improvement strategy or process. We agree with the well-known proverb: "if you can't measure it, you can't manage it."

- Data quality and testing

Usually, testers and engineers assume that the data (in a production environment) is correct, and test the system considering its behavior. However, as we have said, this is not the case in the real world. When a new system is incorporated to an existing environment, the data it uses must be analyzed to understand its usefulness. Moreover, an old system may be using corrupted data. We believe that the verification of a system must include the verification of the data it works on. Besides, we believe that many testing techniques can be adapted in order to be used to test data.

- Data quality in the software development process

The dimensions in which data quality is analyzed (for instance consistency, accuracy) can be considered data quality requirements for a project, and can be assessed from the beginning of the software development process in the same way that we have functional and non-functional requirements. In fact, they are a particular sort of non-functional requirements. So, we want to deal with them from the beginning of the process, and incorporate them to our specification, our design, and our system implementation. In this case, our system should give us a warning when the data is in close or already in a "bad quality" situation with respect to our requirements. And hence we can prevent our system from entering such a situation.

In Section 2 we discuss what data quality means. In Section 3 we describe typical data quality problems. Section 4 presents data quality metrics: why, what, and how to measure. Section 5 is devoted to data testing. Section 6 points out why data quality requirements should be incorporated to the software development process. In section 7 we present our conclusions and future work.

2. What is Data Quality?

It is difficult to give a universal definition of what quality means. When we talk about quality we do not always refer to the same concept. We will try to exemplify this issue. Suppose that you are planning a trip to a foreign country. You have to choose an airline to fly. Which one do you prefer? Of course, you will prefer the airline that offers the *best* quality. But, what does quality mean? You want to arrive in time, you want comfortable seats, you want a helpful crew, you want a quiet trip, and you want low prices. These attributes

(punctuality, comfort, helpfulness, peace, low prices) constitute your notion of quality in this particular context. Even in the trip situation, someone else may not be concerned about the price, but be very worried about the meals served. So his notion of “airline quality” is different from yours. It may differ not only in the attributes taken into account; the relevance of each item may be different. Moreover, you could have different notions of “airline quality” for different trips.

This example shows that quality is not an absolute concept. The word *quality* by itself has not a unique connotation. We have to make assumptions on which aspects apply on a particular situation. In the case of data quality, we may want to take into account only specific attributes with some specific relevance, depending on the particular context we are analyzing. In our view, the quality of data in the context of software systems is related to the benefits that it might give to an organization.

As we have said, the quality of data depends on several aspects. Therefore, in order to obtain an accurate measure of the quality of data, one have to choose which *attributes* to consider, and how much each one contributes to the quality as a whole. In what follows, we present several attributes that we think may determine the quality of our data. These attributes or *dimensions* have been taken from [WW96, SLW97] following the point of view of the value of the data, i.e., our pragmatic view of data quality.

We present an informal definition for each of the attributes considered. This selection is not exhaustive, but is representative enough for our purposes.

Completeness	Every fact of the real world is represented. It is possible to consider two different aspects of completeness: first, certain values may not be present at the time; second, certain attributes cannot be stored.
Relevance	Every piece of information stored is important in order to get a representation of the real world.
Reliability	The data stored is trustable, i.e., it can be taken as true information.
Amount of data	The number of facts stored.
Consistency	There is no contradiction between the data stored.
Correctness	Every set of data stored represents a real world situation.
Timeliness	Data is updated in time; update frequency is adequate.
Precision	Data is stored with the precision required to characterize it.
Unambiguous	Each piece of data has a unique meaning.
Accuracy	Each piece of data stored is related to a real world datum in a precise way.
Objectivity	Data is objective, i.e., it does not depend on the judgment, interpretation, or evaluation of people.
Conciseness	The real world is represented with the minimum information required for the goal it is used for.
Usefulness	The stored information is applicable for the organization.
Usability	The stored information is usable by the organization.

Notice that dimensions may be related to others. For example, the amount of data may be important only in conjunction with correctness (lot of incorrect data has no sense, and even may damage the organization), usability (inefficient access to data due to the size of the database is worthless), and so on. In some way, these attributes complement each other.

3. The Data Quality Problem

- I can't use this application. Look, I know that wells in this field are at most 3000 feet, and the depth of this well in the system is 4500! This system is useless.

- The system works perfectly. Of course, if wrong data is being loaded, what can the computer do?

- I don't know. I just say that it is not good for me. It makes me loose more time looking for data than before.

- It is not our problem. The system works, we detect wrong data when it is loaded –and in the cases YOU specified-. It is a problem of the users: you should tell them to use it right.

This dialog, at least in spirit, happened in many places many times. Data quality problems are real problems in most information systems. With different degrees of criticality and deepness, these problems are being treated in many organizations. In most cases, in an ad-hoc way.

For instance, let us consider mailing lists. How many times do you usually receive a brochure for a conference? How many combinations of first, second, and last name have you seen your name on envelopes? This simple example shows how expensive data quality errors can be: mailing can be quite expensive, and using a faulty list, a mailing campaign can be many times more expensive. Moreover, the lost caused by wrong advertising goes long beyond the cost of mailing: customers and potential customers do not trust someone that is not even capable of keep his/her data right. The image of the organization suffers offering and using wrong data.

However, the particular case of names (and, mainly, occidental names) has been extensively studied and many heuristics have been proposed for the problem of determining whether two names correspond to the same person (in general, in the presence of more data, like date of birth, addresses, etc.). Commercial products and algorithms are available to attack (not to solve) this problem. Even though, this particular problem is cause of misuse of systems in several different contexts.

For instance, criminal identification systems determine if a person has criminal records. This information is used, for example, by judges (to decide whether the person has to be punished, and how), and by organizations (to decide whether to hire him). These systems are critical because, in some sense, our future may depend on the quality of the data and the procedures used to recover it. Although a high quality application is used to access the data, the identification is based on several attributes, and sophisticated algorithms are used to match them, it turns out that wrong conclusions can be obtained when bad quality data is present in the system. It has been found that 50-80% of computerized criminal records in the U.S. were found to be inaccurate, incomplete, or ambiguous [Tay98]. This poor quality data may imply send people to jail or not to hire them.

Data quality problems are not only related to pattern matching of persons or organizations. Such problems arise in many different contexts, and the consequences can be disastrous. The cultural change imposed by the use of computers in many different environments, only makes the problem worse. In fact, people trust computers and utilize them as the main source of data: digital information is used minute by minute to take important decisions that affect people lives.

Recently, data warehouse and data mining projects exposed many data quality problems in big enterprises. When the information collected was analyzed or was checked for integrity, some “hidden” problems were detected. For instance, data from different sources was detected to be inconsistent in data warehousing.

The usual (implicit or explicit) position of software professionals facing data quality issues is that “this is not our problem.” Somehow, information professionals are not responsible of dealing with information.

On the other hand, we have two ideas that contradict that belief. First, some data quality issues are caused by poor design of software systems. In particular, the effect of poor interface design on data quality is direct. For instance, in many cases users of a complex interface with mandatory values have the tendency to choose a random value. If there is a list of values available, users choose the first of the list or the default value.

For example, by studying last year information the managers of a hospital discovered that most of the patients suffered from hemorrhoids. The resources of the next year were assigned on this basis. The number of beds,

nurses, and other resources needed were determined using this information. However, it came out that “hemorrhoids” was the default choice at the check-in application, and clerks selected it because it was difficult to look for the correct choice. This bad data -due to a poor interface design- had terrible consequences on the hospital finances [Tay98].

Another way in which poor design may affect the quality of the data is by failing in a complete analysis of business rules or by not taking into account data quality issues during the requirements analysis phase. In fact, if data quality is a risk, the design of the system must take measures to minimize that risk.

A rule of thumb [Orr98] proposes to improve data quality by increasing the use of the data. Data that is not used cannot be maintained. We agree with this rule. However, several times it has been used to illustrate that problems in the quality of information are not caused by poor design. This is not true. The process of creating and using data must be subsumed in the design of the system. The data flow, the organization of the processes, and the overall design must be created with this data life cycle in mind [Red96]. Not to do so is a modeling and design fault. During the analysis phase, the processes that are automated must be analyzed not only for efficiency: data quality is also a driver when designing the processes and the use of the applications.

The second idea is that many data quality problems can be prevented and deal with by using standard software engineering techniques – adequately adapted or revisited. For instance, configuration management techniques could be used to solve problems with out-of-date data or versioning of information. Standard metric definition techniques could be used to define useful data quality metrics. These ideas are addressed in more detail in the following sections of this work.

4. Measuring Data Quality

The first step to improve data quality and to define methods and techniques is to understand what “good quality” and “bad quality” is. Hence, we need to measure data quality to be able to know how valuable the information is, and how to improve it. Measuring the quality of the information will help us to know its value, and also its pitfalls. We will be able to know how valuable our information is, and also what we need to improve in order to increase quality. Moreover, measuring quality would clarify the goals of a quality improvement strategy or process. We agree with the well-known proverb: “if you can’t measure it, you can’t manage it”. In fact, it is not possible to make serious empirical analysis of techniques or strategies if there is no agreement on how the results will be evaluated.

We propose to measure information quality using metrics defined using traditional software engineering techniques. Metrics have been deeply studied in software engineering [FP97], so we want to take advantage of it.

In [BMY98] we present a framework for defining and using data quality metrics. The outcome of this work is a suitable set of metrics that establish a starting point for a systematic analysis of data quality. We identify the attributes we want to measure, and obtain a set of metrics and techniques to calculate them. This is a starting point for a systematic analysis of data quality, that may lead to improve the quality of the data in an organization.

We base our work on the GQM methodology [BR88]. GQM is a framework for the definition of metrics. GQM is based on the assumption that in order to measure in a useful way, an organization must:

- specify goals,
- characterize them by means of questions pointing their relevant attributes,
- give measurements that may answer these questions.

We have chosen this framework because it is a top down approach that provides guidelines to define metrics, without a priori knowledge of the specific measures. There are other approaches for metric definition, e.g., [BBL76, MRW77]. We have chosen GQM because of its simplicity, its adequacy to our problem, and because it is well known and proven in software engineering applications [Van98].

Following GQM, we first are able to state which dimensions characterize our notion of data quality. Then, we can ask questions characterizing each dimension, without giving a precise (formal) definition -that is

sometimes impossible-, only focusing on their relevant characteristics from our point of view. Finally, we give metrics (some objective, some others based on people appreciation) to answer these questions, giving us a more precise valuation of the quality of our data.

We cannot measure data and ignore how it is organized. Certain quality characteristics are related to the organization of data, i.e., to the *data model*, and not to data itself. The data model might affect some data quality attributes, since it defines the way data is accessed and maintained. We want to identify and measure those attributes too, and complement measures of data with information on how it is organized. As a consequence, we defined two kinds of metrics: set of data metrics, and data model metrics.

Once we have defined our data quality metrics (i.e., what and how to measure) we want to use them. We can simply take our relational database, identify the dimensions we are interested in, choose the appropriate metrics and techniques depending on specific considerations, apply them, and analyze the results. This is a useful approach, specially when the system is already in production, the database is implemented, there is a lot of data loaded, and we want to have a picture of the current situation in order to decide what to improve. We may even add information about the quality of the data to the meta model, as part of its definition. This way it may be easier to check and evaluate the quality of the data at a certain point. In [JV97], this approach is followed in the data warehouse case.

Once we have measured the quality of our data with respect to the chosen dimensions, we can decide whether or not our current data satisfies our quality expectations. Moreover, we will know in which dimension it fails (although we do not know why), with respect to which specific aspect, and we have a measure of the “badness.” So we can concentrate our efforts in solving that particular problem, and we can decide if it is convenient to do so - may be data is not so bad, and the solving effort is worthless.

This procedure only deals with measuring the quality of data at certain points, and can help in deciding which corrective or preventive actions to implement. In order to reach and maintain high levels of data quality, it has to be part of a broader plan, that takes into account all the aspects of data quality in the organization (see [Red96]).

Another approach is to see the dimensions we are interested in as data quality requirements (see Section 6). These requirements can be assessed from the beginning of the software development process, in the same way that we have functional and non-functional requirements. So, we want to deal with them from the beginning, and incorporate them to our specification, our design, and our system implementation. In this case, our system should give us a warning when the data is in close or already in a “bad quality” situation with respect to our requirements. And hence we can prevent our system from entering such a situation. Metrics may be used here to establish the requirements and check them at different stages of the software development process.

5. Testing and Data Quality

Software systems were often analyzed as if they start from scratch. Only recently the idea of using COTS is being incorporated in formal description of the development process. This is even stronger in the case of the data used by these systems. The idea of testing a system concentrates on testing its functionalities: never the data that it is supposed to work with – even if it makes assumptions on what is the state of the data. The phrase “garbage in/garbage out” only expresses the idea of “data is not a problem of software systems.”

If a system is started from scratch, some of these assumptions can be accepted. However, in the daily practice of our profession, most systems are incorporated on top of existing systems or collaborating with existing systems. Many projects use data generated by other systems, in many cases by systems that are not operative anymore.

In our view, it is important to check whether the data satisfies the requirements of the system or, in other words, that the quality of data reaches the minimum level required for the system to work properly. This activity can be done before the system is developed (in order to take corrective measures or include extra components during the development), before installing the system (in order to check how it will work and to prevent problems during its use) or, independently of any system, just to measure the quality of the data.

This verification can be done in three different ways:

- Complete validation of all data.

- Statistical indicators of mean, variance, intervals, etc.; or random selection with an associated confidence.
- Use testing techniques to define and execute test cases.

The first way is clear: validate the whole data, using automatic and manual verification. This is not equivalent to clean the database or files, because the cost of repairing a data error can be many times greater than the cost of detecting it. However, in most cases complete validation is unfeasible. In this extent, it is not different of complete validation of programs [How76]: in many cases the domain of programs are finite and could, in theory, be validated for all inputs. Even though, complete verification is not done, because it is too expensive, too complex, or unnecessary. Only the thought of checking a 2,000,000 registries database to see if any customer has changed his address is scaring.

It is important to discuss the difference between the last two options. There are testing techniques based on statistics, and the activity of testing is strongly related to statistical analysis. However, there is a subtle difference between taking values that describe distribution of data on one hand, and choosing particular cases that satisfy particular criteria on the other. When we propose testing as a technique to validate data quality, we think that it is possible to define *testing criteria* for data quality based on the quality dimensions of interest. In some cases, it might be even possible to define the notion of *coverage*, and to construct *test cases* to satisfy a particular coverage criteria [My79].

For example, suppose that we are interested in measuring how accurate our data is with respect to time (timeliness). Let us assume that we know which attributes are time dependent. If we have a way to know if specific values are outdated, we may define a test over the data to estimate the number of records that are outdated. Hence, in order to implement these testing activities we need to use a selection criterion (to reduce the number of test cases to be evaluated) and an *oracle* (to know if specific values are outdated). We believe that in this case, the selection criterion should use the specific information about timeliness of data, improving the results obtained by using sampling. Testing for other qualities should use different information to select data.

Data testing has the flavor of structural testing –because the structure of data will probably play a basic role in defining the criteria- and aspects of functional testing. The type of coverage used to check data quality will be fundamental to create new testing techniques –or adapt existing techniques for new goals. A lot of work must be done to define adequate notions, and those notions must be validated by empirical data (and, possibly, by high quality data!) before proposing concrete techniques. However, it is clear that testing, as presented in this section, has many advantages over statistical analysis. One of the advantages is that we do not need to define the rules that guarantee that a particular piece of data is of high quality explicitly: for each case we can determine whether the output passes or fails the test. The only difficulty is to choose the right tests. But we know how to do that to test programs: the same ideas should apply here.

6. Data Quality in the Software Development Process

The requirements of a software system are usually divided in two groups: functional requirements and non-functional requirements. Functional requirements include the services the system is expected to provide, while non-functional requirements place constraints on the way those services must be provided [Som94]. Examples of non-functional requirements are programming languages (“the system must be implemented using C++”), performance (“the expected response time is 2 seconds”), standards (“the development process must be ISO compliant”), interoperability (“the system must communicate with the accounting system”). Moreover, non-functional requirements may be classified according to the kind of constraints they impose. So, we have process requirements (constrain the development process), product requirements (constrain the final product), and external requirements [Som94].

Besides, when describing non-functional requirements at early stages of the software development process, we assume that they should be verifiable, that is, we want to be able to decide whether the system architecture, the design, the implementation, the process model, etc., satisfy them.

As mentioned in previous sections, we claim that data quality issues are non-functional requirements that may be incorporated into the software development process. In fact, we may add a fourth kind of requirements: *data requirements*. These requirements must be placed at requirements and specification time, and they will constrain the following steps of the development process. In this way, the system constructed will satisfy

the expected levels of data quality, and we may be able to verify these data quality requirements in the system.

Very often, system developers claim that their job do not consist in understanding what the systems they develop are used for, neither the context in which the systems will be used. They just build systems that meet the requirements of the users; the users have to ensure the quality of the data in the databases [Orr98]. According to our view, if data quality requirements are formulated together with other non-functional requirements, the developer has to guarantee that those requirements are met, and consequently, that the expected levels of data quality are achieved and maintained. Of course, it is not always possible to have an a-priori knowledge of all the aspects regarding data quality, but at least a subset should be available. And the analyst is responsible for asking and obtaining this information.

Users have expected levels of data quality in mind. In fact, they obviously want data in the systems to be used, and this alone constitutes a requirement. Sometimes they have more specific demands, concerning accuracy, timeliness, security, accessibility, etc., of data. These requirements are functional by no means, since they are not related to the services the system provides. However, they are related to how the services will be implemented. For example, if a requirement is placed on the security of the data so that certain data is not accessible to every one, the system must comply with this security requirement in order to satisfy the user expectations. Hence, data quality requirements are non-functional requirements. To include data quality requirements from the beginning of the development process may help to improve the quality of the data during system usage, because the system will be designed to take care of the quality of the data according to the user needs.

Moreover, in information systems the expectations on data quality can be even stronger than the expectations on a particular functionality or operation that the system may perform.

Existent approaches consider data to be independent from the applications that use it [Red96]. This is essentially true. Organizations have information that may be used by many systems, although the data has entity by itself. However, systems are build to use this data in agreement with the rules and needs of the organization. Thus, they have to preserve the consistency of the data, make it accessible, extract useful information, maintain its quality, etc. It follows that applications must take care of data quality. And, as we already know from software development models, it is better to have them in mind from the beginning. It is always more expensive to modify existent systems in order to deal with the quality of the data, to preserve it, or improve it. It is cheaper to include them at the starting point of the development process and verify them at each stage, including the final implementation.

In order to have verifiable requirements, we would rather use formal notations that may allow us to use automatic tools to perform verification of requirements. As opposed to other requirement languages, a language for data quality specification should be decidable and quite simple.

Data quality requirements may be formulated in terms of data quality dimensions [SLW97]. Different requirements may be placed over the same set of data and data model. Hence, we want to be able to decide whether a set of requirements is sound. Moreover, we may place different requirements over different subsets of data.

There are other approaches that deal with requirements on the data at early stages of the software development process [Red96]. However, they do not follow a software engineering approach. In fact, they do not incorporate them as part of standard software development processes; they do not look for a formal, simple, and verifiable notation to describe data quality requirements; they do not apply traditional software engineering techniques to data quality problems.

Strong, Lee, and Wang [SLW97] describe common existing problems with data. They identify their source, the dimensions affected, and the impact on the organization. They propose general solutions to these problems, for instance, as guidelines to the process development and management. They do not formalize the expected data quality levels, and cannot verify if they are achieved. This analysis can be done when the problems are detected, and the experience could be used in future developments.

Redman [Red96] proposes to understand the customer needs prior to the software development. He translates user requirements into technical requirements written in natural language. Some of these requirements are formulated in terms of specific conditions over certain data (for example, “the new address must be in the

system within two weeks”). He determines which dimensions are affected by each requirement. To do this, he uses an impact matrix, where the impact of each requirement is rated as “high”, “medium”, or “low”. It is hard to verify if the technical requirements really correspond to the user requirements, since they are both informal. Also, it is difficult to verify if the technical requirements are satisfied. Moreover, with such a limited scale, it is hard to determine the desired quality levels precisely, and consequently, to verify their achievement.

7. Conclusions and Future Work

Problems in the quality of information are real problems in almost all organizations that use large databases. In this work we have discussed the characteristics of the data quality problem, in particular related to other quality topics usually considered in the software engineering field. Moreover, we have proposed three specific lines in which particular techniques could have a direct impact on how organizations deal with these problems.

This work presents more problems than solutions, it is just a particular point of view to attack data quality problems. In order to obtain concrete results, more work must be done in each of the three themes proposed. In particular, empirical validation is mandatory to check the adequacy of the methods and techniques proposed. Actually, this work can be used as a research agenda, and the lines presented are the basis of our research program on data quality.

The start point of this research program is the definition of metrics for data quality. Without a clear knowledge of what and how to measure, it is difficult to attack the underlying problems or to define objective experiments to check improvement after the use of new techniques [BMY98]. A particular issue related to this point is the value of data. At some point in our program, we would like to have a notion of value of information (in the sense of dollar value or market value), probably related to its use.

Data testing and the incorporation of data quality in the software development process are both issues that must be investigated before defining practical techniques.

One of the main conclusions of this work is that software engineers cannot ignore data quality in the day to day practice, and that many among the best practices of the field can be adapted to work with data quality.

Acknowledgements

This research was partially supported by the ANPCyT under ARTE Project grant PIC 11-00000-01856, and the ANPCyT under grant PIC 11-00000-0594.

References

- [BR88] Basili, V.R., Rombach, H.D.: The TAME Project: Towards Improvement-Oriented Software Environments, *IEEE Transactions on Software Engineering*, vol. 14, no. 6, June 1988.
- [BMY98] Bobrowski, M., Marré, M., Yankelevich, D.: *Measuring Data Quality*, submitted for publication.
- [BBL76] Boehm, W., Brown, J.R., Lipow, M.: Quantitative Evaluation of Software Quality, *Proceedings of the Second International Conference on Software Engineering*, 1976.
- [FP97] Fenton, N.E., Pfleeger, S.L.: *Software Metrics - A Rigorous & Practical Approach*, 2nd edition ITP Press, 1997.
- [How76] Howden W. E.: Reliability of the Path Analysis Testing Strategy, *IEEE Transactions on Software Engineering*, vol. 2, 1976.
- [JV97] Jarke M., Vassiliou Y.: Data Warehouse Quality: A Review of the DWQ Project, *Proceedings of the Conference on Information Quality*, MIT, Boston, October 1997.
- [MRW77] McCall, J.A., Richards, P.K., Walters, G.F.: *Factors in Software Quality*, Rome Air Development Center, RADC TR-77-369, 1977.

- [My79] Myers G. J., *The Art of Software Testing*, Wiley, New York, 1979.
- [Orr98] Orr, K.: Data Quality and Systems Theory, *Communications of the ACM*, Vol. 41, No. 2, pp. 66-71, Feb. 1998.
- [Red96] Redman, T.: *Data Quality for the Information Age*, Artech House, 1996.
- [Red98] Redman, T.: The Impact of Poor Data Quality on the Typical Enterprise, *Communications of the ACM*, Vol. 41, No. 2, pp. 79-82, Feb. 1998.
- [SLW97] Strong, D., Lee, Y., Wang, R.: Data Quality in Context, *Communications of the ACM*, Vol. 40, No. 5, May, 1997.
- [SLW97] Strong, D., Lee, Y., Wang, R.: 10 Potholes in the Road of Information Quality, *IEEE Computer*, August 1997.
- [Som94] Sommerville, I.: *Software Engineering*, Addison-Wesley, 1994.
- [Tay98] Tayi, G.K.: Research Seminar on Data Quality Management, Universidad de Buenos Aires, July 22th, 1998.
- [Van98] Van Latum F., Van Solingen R., Oivo M., Hoisi B., Rombach D., and Ruhe G.: Adopting GQM-Based Measurement in an Industrial Environment, *IEEE Software*, pp. 78-86, January-February 1998.
- [WW96] Wand, Y., Wang, R.: Anchoring Data Quality Dimensions in Ontological Foundations, *Communications of the ACM*, Vol. 39, No. 11, November, 1996.