MASTER

Setting up a quality database for analysis of software development process information

Ahogado Alvarez, D.C.

Award date:
2013

Setting up a quality database for analysis of Software Development Process Information

Diana Carolina Ahogado Alvarez

August 2009

Master Project
M.Sc. in Business Information Systems
Department of Mathematics and Computer Science

Supervisors:
dr.ir. J.J.M. Trienekens, TU/e, TM/IS
prof.dr. R.J. Kusters, TU/e, TM/IS
ir. J. Samalikova, TU/e, TM/IS

Eindhoven University of Technology
Department of Industrial Engineering and Innovation Sciences
Software Engineering Management Group
Information Systems

Contents

Chapter 1

1 Research Project Definition

The field of data quality [STR97] has been studied as a topic of importance in the creation of Information Systems and the structuring of data resources that act as their ground. This is because the necessity of producing information that can be highly useful to the users of applications is notorious in every sector where software is employed to support their daily operations.

Data quality refers to different characteristics that make data in a data source, such as a database, more close to the real world they are representing, e.g. consistency, completeness, reliability, and accuracy [BER07], among others. Of course, the definition of the more suitable characteristics depends on the context where the data is being used, and the requirements of its utilization by the users. For this reason, many of those characteristics, called "data quality dimensions" [BAT06] have been defined and used in literature and industry, without specifying a complete standard group of them that should be added to a data source.

The addition of those quality dimensions can be done through different approaches. It is possible to design a database where not only the information about entities of the real world is represented, but also the characteristics of quality that are desired for them. Additionally, it can be that during the software development process, some activities are followed to support the work of developers in order to gather the information needed to design the database in the described way.

But also, the addition of quality is feasible to be done even when an Information System already exists and is being used. In this case once the quality dimensions are defined, it is necessary to assess the existing data by using metrics related to those dimensions and afterwards apply appropriate techniques that allow improving the quality where errors are found.

For both approaches, methodologies have been proposed that indicate how to design or improve applications and data sources [WAN93]; some of those methodologies give the notion of how to add quality dimensions, by using metrics and techniques for eliminating errors [BAT06]. Furthermore, some graphical methods have been also created in order to serve as a tool that supports the representation of data and their quality characteristics [SCA02].

The main focus of this research project is to mainly create a well founded and well documented quality database, and depict the methodology to do it. By quality database it must be understood, a database where the data stored has quality. This quality is added after analyze the initial quality level and apply the corrective actions to eliminate found errors.

Given this focus, the work results in three important deliverables; the quality database, the quality documentation of the database, and, a model that describes the process followed for the creation of the quality database.

1.1 Context of the project

The project is developed in the context of an agreement created between the Information Systems group, research group of the Faculty of Technology Management, and a consultant who works for companies on the analysis and improvement of their software development process.

The objective of this consultant's job is to solve the problems currently existing in companies, where projects for software development are too long. One cause for this problem is that data about the development activities is not stored frequently; therefore there are not enough information sources that can be consulted to understand what is wrong and what can be improved.

For his job as a consultant, he has created a procedure which is implemented in every company where he provides his services. This procedure is supported by a tool called "Measurement Database" [SIE09], which is in charge to frequently collect data about different activities performed during the development process. Once enough data has been collected, this tool is used to generate metrics about the projects, the products constructed and the process followed for that construction.

Some of the information collected is about tasks performed during the development, the context of the components that compose an application, defects found in the components, and size of the files that constitute the component. Thus, after analysis appropriate information about them, metrics are calculated in different areas such as Productivity, Quality and Timing.

As the collected data are usually stored in different data bases, the "Measurement Database" first retrieves them from those data bases, and then processes them and stores them in a predetermined format in csv archives called the "snapshots". These archives are the basis for the next step, which consists in a new processing of the data for store it in a consolidated data base that can be used to obtain the information necessary to calculate the metrics.

The metrics calculated for every company are defined according to their information needs, derived from their business goals. From the metrics some indicators are defined to give information needed to the project and organizational management.

During the last years, through the agreement made with Eindhoven University of Technology, the consultant has provided some of the data he collects from companies, in order some research projects can be executed using them. The goal of these projects has always been to understand the processes that are lying on the information provided. Some of these projects have been focused in the analysis of defect data [IB0E8] and the change control board [URE08]. The intention behind providing this information to the university, is to allow that research on the field of improvement of the software development process can be done, but also through the job made by students and researchers, to gain some inside on how he could improve the process he is following nowadays in the work with the companies.

The amount of data contained in the snapshots, which is the basis for research provided by the consultant, is too big, and also requires time to be understood and used for research. Therefore contemplated in objective of the current research project a complete analysis of the structure of these files is included, before creating the database where the data will be structured and which will be used as reference for future research projects.

1.2 Objective and research questions

The objective of this research project is *"the creation of a well-founded and well-documented quality database to structure and analyze information, and the description of the process followed to create this quality database"*.

As already mentioned in the previous section, quality database refers to a database where the information stored has quality. In this case the data provided by the consultant is evaluated and its quality improved to be stored in a well documented database where it is structured.

In order to achieve this objective, the next main questions are answered before proceeding with the practical work:

- What does data quality mean? This is to understand what it means that a data resource contains quality.
- How can data quality be measured in an existing database? Given the fact that there is already information to be used which is stored in the snapshots, it is essential to understand how to measure the quality level on it.
- How can data quality be improved in an existing database, which means, how can data cleaning be made? Once the level of quality is measured in the snapshots, it is necessary to find out some techniques to improve quality in an existing data resource.
- How can data quality be added to a database? As a data model is designed and implemented it is also necessary to know how to add quality to a data resource that is completely new.

The information analyzed and stored in the database created based on the model is related to the software development process of two companies, which in this report are called Company 1 and Company 2 for confidentiality reasons.

Given this confidentiality to be respected one of the considerations taken into account in the moment of the design of the model is making the information anonymous. Nevertheless as also was required by the provider of the data, the information though anonymous is also traceable; this means, that it is possible to find out whenever it is necessary who is the owner of the data.

1.3 Research Outline

The execution of the research project starts with a literature review that is necessary for understanding the concepts related with data quality that must be used to answer the previously formulated questions; then, the process, called work methodology, to be followed to improve quality of data in the snapshots, create the new database, and prove its quality level is explained. Such methodology is based on the theoretical aspects researched and is also aiming to show how the main questions are answered.

Once this theoretical background and the methodology are complete two main phases are followed: Creation of the quality database and test of quality level on the information stored in the database. Each of these phases is explained in the following subsections.

1.3.1 Creation of the database

During the first phase, most of the important activities to achieve the goal proposed for the project are performed. Such activities are oriented to the creation of the quality database that is finally used to structure the information provided by the consultant. These activities compose the work methodology which is one of the final deliverables of the project.

The work methodology, which will be explained further in this report, consist mainly on the understanding of the data provided by the consultant in the snapshots, to then executing a cleaning of this information in order to improve its quality, and afterwards proceed with the design, creation and documentation of the database where the quality data is stored.

1.3.2 Tests the information:

During this phase the aim is to test the quality of the database created, in order to verify whether it really contains the data quality characteristics improved and/or added in the previous phase. Some experiments are thus performed to prove that the data stored in the created database can be used. These experiments are designed and executed by using some data mining techniques.

1.4 Theoretical aspects

As already mentioned, some of the practical activities performed during the project are supported by literature. The theoretical aspects necessary for this were investigated and are enumerated here; they are the basis for proposing the work methodology followed. These theoretical aspects are:

1. Definition of quality data and setting up of rules to create data sources that contain it, or improve it in already existing ones. Rules that are applied when making the cleaning and during the creation of the database.
2. Establishment of the requirements to be followed when making a migration from a database to a Data Warehouse. This topic was researched and is documented

although during the execution of the project a Data Warehouse was not created for storing the data.

1.5 Structure of the document

The remaining part of this document is organized in 6 chapters where the development of the phases explained above is described. In chapter 2, the literature review about data quality is presented along with the theoretical aspects concerning migration to Data Warehouses. Chapter 3 explains the work methodology to be followed with the aim to improve the quality of the data and create the quality database, which will be based on the findings introduced in chapter 2. Chapter 4 is dedicated to initiate the application of the work methodology with the description of the work made by the consultant and of the structure of the data contained in the snapshots. Afterwards, in chapter 5 the practical application of the work methodology is documented, relating the steps followed and results obtained at the end of this activity. Then in chapter 6, the definition and execution of the tests to prove the quality level of the database are exposed, and the conclusions obtained from these tests are also elaborated. Finally, in chapter 7 the conclusions about the work made and results obtained are done, along with some recommendations for future work.

At the end of the report the appendix A is depicting the structure of the data stored in the snapshots. Appendices B, C and D, contain the quality documentation that is one of the deliverables of the project; appendix B describes the quality level measurements made to data in the snapshots and improvements performed, appendix C contains the graphical models and documentation of the database and appendix D contains the results of the tests performed. Appendix E presents a process model which summarizes the work methodology that was followed to create the database.

Chapter 2

2 Theoretical background

This chapter is aimed to bring a conceptual background through which the main questions formulated for the project can be answered; it will be the basis to produce the three main deliverables of the project: the quality database, its quality documentation and the process followed for the creation of the database. The first section describes essential concepts about data quality, while the second one is related to Data Warehouses.

2.1 Data Quality

In this section the following concepts about data quality are presented: meaning of data quality, how to measure data quality, how to improve it and how to add it to a database. At the end the conclusions about the ideas that are more relevant to create the quality data model are summarized.

2.1.1 Concept of Data Quality

There is not a standardized consensus about the concrete meaning of data quality means; nevertheless, it could be described as the characteristics that make the data in a database the most possibly useful and reliable for users according to their information needs. Therefore, according to [WAN93], the better the representation of the real world is made by data in a database of an information system the better is its quality.

Addition of quality characteristics to a database should be done during its design and construction, and thus the quality depends on how good is the execution of those phases [WAR96]. The designer of a database must consequently have a complete understanding of the information necessities of the users, which reflect the real world where they work, and also of the quality requirements they have for this information.

Some design deficiencies that could conduce to inconformity are [WAR96]:

- Incomplete representation: When no exhaustive representation of all states in the real world is made in the information system
- Ambiguous representation: When two or more states of the real world are represented by the same state in the information system.
- Meaningless state: states that don't represent any real world property.

2.1.1.1 Data Quality Dimensions

Data quality dimensions are a more formal way to name the characteristics of quality that data should have. They depend on the context where data are used and therefore many of them have been proposed by authors, but there is not a standard set that should be used. Some of the most common are accuracy, timeliness, interpretability, completeness [WAN93] and consistency [WAN95].

Table 1 shows dimensions proposed by several authors. In some cases, different authors give a different connotation to the same dimension; therefore after analysis to find common meanings, for every dimension a definition is given and in case it is necessary, the different descriptions given by authors are presented.

Data quality dimension	*Description*
Accuracy	[WAN95] and [WAR96] agree on accuracy as the conformity between a value recorded in the database and the real world value.
Timeliness	The value recorded in the data base is not out of date [WAN95]. Also the availability of information on time [WAR96].
Completeness	All values for a certain variable are all recorded [WAN95]. It means that every meaningful state of the represented real world is stored, or according to [WAR96] there are not missing states.
Unambiguous	It is when there is a proper representation of the states of the real world in the data. Not multiple states mapped to the same one [WAR96].
Meaningful	All the states stored in the database can be mapped to a state existing in the real world [WAR96].
Correct	All the information in the database is mapped to correct states of the real world.
Consistency	It is related to the values of data and it means that the representation of the data is the same in all cases [WAN95], [WAR96].
Reliability	It indicates whether data can be counted on to communicate the right information [WAR96].

Accessibility	It is the extent to which data is available, or easily and quickly retrievable [PIP02].
Understandability	It is the extent to which data is easy to be comprehended [PIP02].
Concise representation	It is the extent to which the data is compactly represented [PIP02].
Consistent representation	It is the degree to which data is presented in the same format [PIP02].
Believability	It is the degree to which the data is regarded as true and credible [PIP02]. When data consumers find no quality and don't know to whom the problem should be attributed, there is a problem of believability [STR97].
Free of error	It is the degree to which data is correct and reliable [PIP02].
Ease of manipulation	It is the extent to which data is easy to manipulate and apply to different tasks [PIP02].
Interpretability	It is the level to which data is in appropriate languages, symbols, and units, and the definitions are clear [PIP02].
Objectivity	It is the degree at which the data is unbiased, unprejudiced and impartial [PIP02].
Relevancy	It is the extent to which the data is applicable and helpful for the task at hand [PIP02].
Appropriate amount of data	It is the extent to which the volume of data is appropriate for the task at hand [PIP02].
Security	It is the extent to which access to data is restricted appropriately to maintain its security [PIP02].
Currency	It is the time a data item was stored [WAR96]

Table 1. Most cited data quality dimensions

2.1.1.2 Methodologies

Some methodologies have been proposed that allow designing a database adding the characteristics necessary for it to have data quality, but that are also useful to improve the quality on already existing databases. The addition or improvement in these methodologies is made through quality dimensions.

Data Quality Requirements Analysis and Modeling

In [WAN93] the idea of tagging the data is suggested as a mean to give additional information that can help users to obtain all the information they need when they retrieve it from the application that uses the database. For example, in the case that they require always the most updated information related to a bank transaction, timeliness should be considered as a quality dimension to be included in the creation of the data base.

During the design of the database all the information requirements must be modeled as entities and the quality requirements must be modeled as tags (special fields) of those entities for which a special quality dimension is desired. For example in the case of adding a tag of timeliness to a transaction, a field such as the date it was done would be appropriate.

Framework for analysis of data quality research

In [WAN95] a framework is proposed that not only considers the aspects related with the design and control of data quality dimensions in a data base, but also gives importance to the organization where the application that uses the data will be employed. The process of creation of information is comparable with the manufacturing process; consequently, it is necessary to include quality aspects in every step.

The framework has 7 elements adapted from ISO9000:

1. Management of responsibilities: create a data quality policy that adapts to all the phases of production of data products, according to quality requirements.
2. Operation and assurance costs: Constantly monitoring costs for data quality assurance.
3. Research and development: Create technical specifications for the quality requirements, including acceptance and rejection criteria.
4. Production: Constantly check the conformity of raw data with quality requirements. Correct found errors in the process of creation of these data.
5. Distribution: Plan production of data and data quality products; control their distribution and maintenance. It must be well documented.
6. Personnel management: Personnel must be trained, qualified and motivated towards the use of data quality standards.
7. Legal function: Identify safety aspect of data products to enhance product safety and minimize product liability.

TDQM

The TDQM program (Total Data Quality Management) proposed by Wang et al., provides a methodology aimed to produce high quality Information Products (IP) through the implementation of a quality policy in an organizations [WAN98]. It is based on the idea of manufacturing of products and compares it with the manufacturing of Information Products, which is a process in which quality requirements can be added to the data in an Information System.

It includes the modeling of data quality in the Entity-Relationship conceptual database model [BER07]. This methodology is composed of four phases:

- Definition: In this phase the information requirements (IP characteristics) for an application are defined and along with the quality requirements for the information. Also the components derived from the requirements and their relationships are defined and can be represented in an entity-relationship model. From the IP characteristics and the assessment of quality requirements, which indicates the necessary quality dimensions, the logical and physical models can be developed; the quality attributes are added in these models. The definition of models can be done using IP-UML [SCA02] as a graphical support.
- Measurement: In this phase metrics are defined for data quality dimensions, in order to track the level of quality of attributes in the database, e.g. the number of records that violate referential integrity. On the other hand, at a higher level, also some business rules must be observed, and therefore procedures for this are developed.
- Analysis: In this phase, the results from measurements made in the previous phase are analyzed to detect their causes.
- Improvement: In this last phase, the procedures to improve quality in the areas where problems were detected are defined.

2.1.2 Measurement of Data Quality in an existing data model

Simple Ratio

In [PIP02] some techniques are proposed for performing objective assessments of data quality. One of them is the use of a *simple ratio* which implies to perform some simple mathematical operations using the quantity of registries in the data base, with the aim to know how good or bad the data in the data base is regarding different quality dimensions. For example, for measuring how much free of error is the data, the number of units of data in error must be divided over the total number of unit data, and the result must be subtracted from 1. The more the result obtained is close to 1, the more the quality of the data related to the error it contains is.

Methodologies for assessment

In order to make the necessary evaluation of the data quality of an information system with regard to data quality, specific assessment methodologies have been created [BAT06]. Usually the steps followed in these methodologies are:

1. Choose the relevant dimensions that are going to be used to measure the quality of the data bases and the data flows in the IS, and the metrics that are necessary for this procedure. The dimensions can be classified into one of four categories: sound, useful, dependable, and usable, and they are classified in order to provide a context for every one of them and for their consequent evaluation.
2. Make subjective judgments of the measures obtained which are made by experts.
3. Compare the values obtained during measurement with values that are already established as acceptable; or performing a benchmarking with best practices providing at the end suggestions for improvements.

Data Quality Dimensions and metrics

Dimensions are defined in a qualitative way only providing a description of what they mean, and therefore metrics must be associated to them in order to give a measure. For the metrics there are measurement methods indicating where the measurements are taken, what data they include, the measurement device and the scale on which results are reported. Some dimensions and types of measurements associated are presented next.

Accuracy:

Accuracy indicates how close value v of an attribute in a record is to the real value v' that it aims to represent. There are two types of accuracy, syntactic and semantic [BAT06].

Syntactic accuracy is the closeness of a value v to the elements of a domain D, i.e. that the value belongs to that domain. This kind of accuracy is measured by comparison functions, which evaluate the distance between v and the values in the domain; for example the edit distance that measures the number of steps to convert a string like "jon" into "john".

Semantic accuracy measures how close a value of an attribute in a record is from the real value that it should have; for example when the data about a person contains a name "James" that is syntactically right but nevertheless the real name is "John", there is a semantic accuracy error. Semantic accuracy is better measured with a <yes, no> or a <correct, not correct> domain.

Completeness

Completeness is "the extend to which data are of sufficient breath, depth and scope for the task at hand" [BAT06]. There are three types of completeness: schema completeness that indicates the degree to which concepts of the real world and their attributes are not missing from the schema; column completeness, which measures the missing values for a column in a table; and, population completeness which evaluates missing values with respect to a reference population.

One of the ways of characterizing completeness in a relational model is by the presence/absence and meaning of null values: It is important to understand why a null value is present in a table, if it is because it exist but is unknown, or it does not exists, or because it may exist but it is not known whether it actually exists or not. The second case would not be considered as incompleteness.

There is a special case of this characterization called Closed World Assumption, in which is sure that only the values present in a relational table represent facts of the real world. In this case it is possible to define completeness with different levels of granularity: value completeness (the presence of null values in some fields of a tuple), tuple completeness (completeness of the tuple with respect to the values of all its fields), attribute completeness (number of null values of a specific attribute in a relation), and relation completeness (presence of null values in a whole relation).

Consistency

Consistency allows discovering the violation of semantic rules over a set of data, like tuples in a relational table or records in a file; the integrity constraints are an example of those rules, which must be satisfied by all instances of a database schema. Integrity constraints may be defined for schemas and for instances.

There are two types of integrity constraints, intrarelation integrity constraints that regard single attributes or multiple attributes of a relation, and interrelation integrity constraints, which involve attributes of more than one relation. Most of the integrity constraints are considered dependencies among which there exists a Key dependency, which enforce that there are not duplicated values within the relation; other option is the inclusion dependency which states that some columns of a relational instance are contained in other columns of the same instance, or the columns of another instance.

2.1.3 Improvement of data quality in an existing data model

After the measurement of data quality has been done many quality problems are detected. Quality problems occur when capturing, gathering or importing information; some of them are duplication of data, or not standardized format or schema of the sources from which the data comes [BER07].

In [BAT06] a number of quality activities to correct errors and thus improve data are described. Some of them are explained next.

Error localization and correction

Error localization and correction are useful every time data have been collected from error-prone sources or acquired from sources whose reliability is not known at all. There are three steps to follow: localize and correct inconsistencies, localize and correct incomplete data, localize outliers (data that are anomalous with respect to other data).

1. Localize and correct inconsistencies.

The localization of errors is done through the use of edit rules, which indicate the semantic rules that should be complied by data in the tuples, e.g. Role = professor and AnnualIncome < 100.000.

- The activity of localizing errors by means of edit rules and correcting them according to these rules is called edit-imputation problem. When applying this technique it is desired to achieve that the data in each record satisfy edit rules by changing the fewest fields possible.

The model, proposed by Fellegi and Holt [WIN06] provides a way to find the minimum number of fields to change in order to respect all the edit rules. There is an important assumption in this method, which is that implicit edit rules are known. An implicit edit rule is derived from explicitly defined edits. For example:

edit1: Age > 15 and MaritalStatus = married

edit2: MaritalStatus = married and Relationship-to-Head-of-Household = spouse

An implicit edit, as may easily be checked, is

edit3: Age > 15 and Relationship-to-Head-of-Household = spouse

2. Incomplete data

There are two cases of incompleteness, one when data is not complete in the context of relational tables, and the other in the measurement of phenomena during a period of time.

In the context of relational databases, the problem of finding the number of attributes to be modified is related with finding the number of attributes that are missing. "Thus, the goal that becomes critical is to maintain the marginal and joint frequency distributions of the attributes. If the attributes to be considered are A1, A2,. . . , An, an assumption can be made that attributes are missing monotonically, that is, Ai is not missing only if Ai−1, Ai−2, . . . , A1 are not missing. In this case, a regression method can be performed recursively, generating valid values from A1 to An".

In the case of time series there are two types of incompleteness, truncated data and censored data. Truncated data corresponds to records that are dropped from an analyzed dataset. Censored data corresponds to data that is known not to have been collected before a certain time t1 (left censored data) or after a certain time t2 (right censored data). This time series problem could maybe used to detect whether information is not complete for a period of time in the snapshots, e.g. data for a problem concerning requirements is included for a period, but not the data for testing.

3. Discovering outliers

An outlier is a value usually larger or smaller than other values in a dataset, which could exist due to one of different cases:

- It was incorrectly observed, recorded, or entered in the database.

- It comes from a different population, in relation to other values.
- It is correct but represents a rare event.

It is important to distinguish between the cases when there are data glitches, corresponding to the two first cases, and cases when there are correct but rare data, as in the third case. It helps to follow the method for managing outliers, which consists on discovering outliers, and then decide whether they are rare data or they are data glitches.

Some methods useful for the detection of outliers are:

- Control charts: Several data samples are collected, and then statistics, such as mean and standard error are computed and analyzed.
- Distributional outliers. According to this method, outliers are seen as points which are in a region of low density.

Object Identification

The quality activity of object identification is related to the case when information related to the same object is stored in different sources, where some attributes are common among the sources and others are particular of every one of them. The techniques used to deal with the activity of object identification depend on the type of data used to represent objects. There are three types:

- Simple structured data, which correspond to pairs of files or relational tables.
- Complex structured data, which are groups of logically related files or relational tables.
- Semi-structured data, such as pairs of xml documents.

Most of the methods proposed in the literature for record linkage consist of the five following steps [BER07]:

1. Pre-processing for coding, formatting and standardizing the data to compare
2. Select a blocking method to reduce the search space by partitioning the datasets into mutually exclusive blocks to compare.
3. Select and compute a comparison function: this step consists of measuring the similarity distance between the pairs of records for string matching.
4. Select a decision model: this step consists on assigning and classifying pairs of records as matching, non-matching or potentially matching with a method that can be probabilistic, knowledge-based or empirical.
5. Validation of the method and feedback.

Standardization

This is a way to change the values of the existing data according to standard formats e.g. change from Channel Str. to Channel Street. It is usually performed as a preprocessing activity in error localization, data integration and mainly object identification.

2.1.4 Addition of data quality to a new data model

During the design of a data model a conceptual schema represents the requirements for an application; this is translated into a logical schema, where the queries and transactions are expressed [BAT06].

Conceptual models (schemas)

In conceptual models quality can be added by extending the Entity Relationship model based on the attributes of the entities.

One option is to create a data quality schema composed of:

1. The original data schema with its entities and corresponding attributes.
2. Additional entities with the attributes <DimensionName, Rating>, which represent quality dimensions and their corresponding ratings; rating corresponds to the possible corresponding values from measurements
3. The relationships between attributes of the normal entities, and the corresponding entities that represent their Data Quality Dimensions.
4. A DataQualityMeasure entity employed to represent metrics for dimensions and its relationship with entities, attributes and dimensions. It has an attribute Rating, which values depend on the specific dimension modeled.
5. The relationship between the attributes and their related Data Quality Dimension entities, and their Data Quality Measure entities with a new representation structure that extends the Entity Relationship model, and relates entities and relationships.

The figure 1 presents a graphical example of this approach:

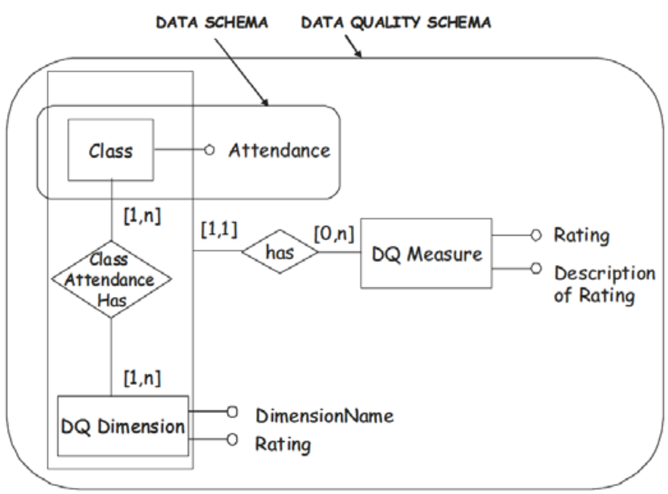

Figure 1. Extension of a conceptual data model [BAT06]

Logical models (schemas)

One possibility to give quality to a logical model is to extend the relational model adding quality values to each attribute, resulting in a quality attribute model. Those quality values represented by quality indicators are linked to the attributes through a quality key, and they indicate the value of every quality dimension for every attribute in an entity; there is also a value for every dimension that summarizes the values associated to the attributes to the whole dimension. The figure 2 shows an example of this approach:

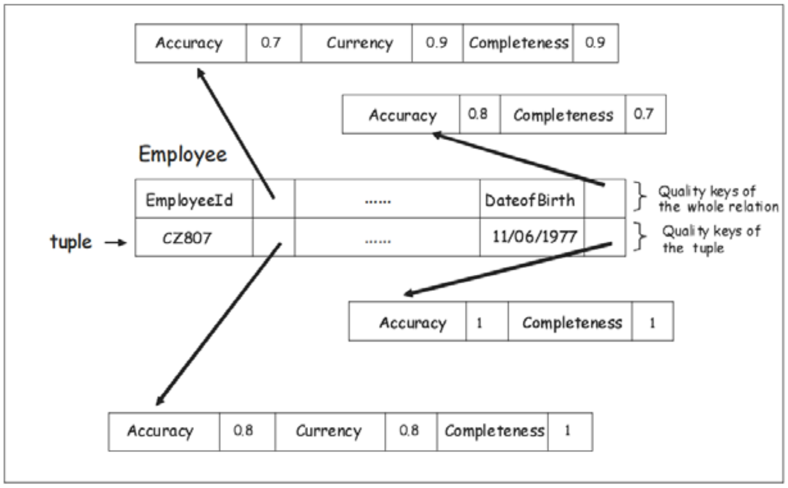

Figure 2. Example of a quality attribute model [BAT06]

2.1.4.1 Quality dimensions for a schema

Interpretability is a general dimension that can be added to give quality to any schema (data model) during its creation [BAT06]. It consists on the creation of documentation and metadata to correctly interpret the meaning and properties of the data sources. The types of documentation that should be available are:

- The conceptual schema of the database.
- The integrity constraints that hold among data.
- A set of metadata for information about the resource including creator, subject, description, publisher, data, format, source, and language.
- A certificate describing available measures of data quality dimensions and schema dimensions.
- Information on the history and provenance of the data.
- Correctness with respect to the model: concerns the correct use of the categories of the model in representing requirements.
- Correctness with respect to requirements: Correct representation of requirements in terms of the model categories.

- Minimalization: Every part of the requirements is represented only once in the schema, which is useful to avoid redundancy.
- Completeness: It measures the extend to which a conceptual schema includes all the conceptual elements necessary to meet some specified requirements.
- Pertinence: It is a measure of how many unnecessary conceptual elements are included in the conceptual schema.
- Readability: It means create diagrams and schemas in an entity relationship model that are clear enough for their intended use.
- Normalization: Normalization in the relational model is related to the structure of functional dependencies. In the case of this project it will be enough to reach the third normal form.

2.1.4.2 Graphical representation

Regarding the management of data quality in Information Systems there is a graphical model called the Information Production Map (IP-MAP) [BAL98] which allows analyzing the production of information as a process comparable with the normal manufacturing process in a company. In this model several graphical constructs are used to compose the model that illustrates the process. This is useful for understanding who the owners of the process phases are, understand the organizational boundaries and estimate time and quality metrics associated with the production process.

The IP- MAP model has been extended to include more characteristics for representation of other aspects related with the production of information; for example, the IP-UML [SCA02] is a modeling formalism created extending UML with a data quality profile based on IP-MAP. The data quality profile consists of three different models:

- Data analysis model: represents the data that are important for consumers as its quality is critical for the organization's success. It has labeled classes that represent the raw data, the component data and the information products (elements of IP-MAP).
- Quality analysis model: contains elements that represent quality requirements of data, related to quality dimensions. In order to model the dimension-related requirements, two stereotypes are introduced: A quality requirement class that represents the quality requirements that can be specified on a quality data class, and a quality association class that associates quality requirement classes with quality classes.
- Quality design model: specifies the perspective in which processes are described together with the exchange of data, by combining the UML activity diagrams with the UML [STE06] object flow diagrams. The stereotyped

activities, actors and dependencies from UML are added to represent IP-MAP elements.

2.1.5 Theoretical aspects to be used

The concepts necessary to understand the meaning of quality data have been given, answering the first main question of this project.

For the other questions, how to measure data quality, how to improve it and how to add it, it was understood that the basic way to manage quality is through quality dimensions. In this manner any activity oriented to the creation of the quality database, will be based on this idea.

Some general definitions of many of these dimensions were given, but in three specific cases, Accuracy, Completeness and Consistency, more extended descriptions have been presented. These descriptions give a suggestion about what aspects can be measured in the data to assess their quality level, and how to improve it.

In order to measure the quality level and to improve it, different approaches and methodologies have been proposed. Some of them, such as the TDQM methodology, present a series of steps that give a complete guide considered very useful in the context of the project. This methodology is intended to orientate the process of creation of information systems providing the necessary activities to include quality in the data in every step of the process. Given such a scope, it must be adapted to be used in this project.

The adaptation of the methodology consists on using only the activities of every phase that are related to the improvement of quality, and to the creation of the data model, since there is not an information system to be created. More concisely, the activities from the methodology to be taken into account are those related with defining quality dimensions to be measured in the data, measure the current quality level, and improving it, as well as designing the database taking into account new possible quality dimensions for the information to be stored there.

Besides, the idea of adding the general concept of interpretability to the database will serve to reach the goal of making it well documented. And the support given by graphical models such as IP-UML is appropriate to analyze and give an initial structure to the data model that will be used for the creation of the database.

2.2 Data Warehouse

In this section, a brief concept of Data Warehouse is given placing special attention to the way preparation of data must be done for migration to this kind of system. This topic is treated here only with the aim of giving an idea of what this technology can be used for, and state that in future work, the data stored from snapshots into the new database could be migrated to a Data Warehouse.

A Data Warehouse is a system aimed to support the decision processes in organizations. This is made by storing data from different sources such as organizational databases, legacy systems, files, or external databases, and providing information out of those sources according to the requirements of the users.

The presence of different data sources introduces a heterogeneity characteristic that brings with it issues like semantic difference among the data. This characteristic of heterogeneity is one of the most important issues to be addressed when creating and maintaining a data warehouse, given that data stored there must be standardized before introducing it into the warehouse schema. That standard structure depends on the requirements of the users and on the business rules of the organization where the warehouse is intended to be used [JAR03].

The management of heterogeneity is also important because the nature of the warehouse has a tendency to change continuously, given that information requirements and business rules are evolving with time; therefore the design process is made iteratively to maintain the warehouse updated to comply with those changes, making it flexible [GAR98].

Additionally, designers must manage the production of an enterprise model for the data warehouse, followed by the derivation of the logical structure of relations.

2.2.1 Architecture

The architecture of a data warehouse is composed by several layers [JAR03]:

- The lowest layer is composed by all the heterogeneous data sources that provide the initial set of data in different formats.
- The central layer contains atomic data and lightly summarized data in a set of integrated databases called the global data warehouse. Therefore the volume of information here is high. The schema in this level is oriented towards query efficiency at the cost of schema normalization.
- The third layer contains highly aggregated data from the global warehouse, e.g. data marts or OLAP databases, which are accessed by the final users of the system. Here the data is less voluminous.

There is a fourth layer present in some cases which is called Operational Data Store (ODS). This layer is located between the original data sources and the global warehouse, and contains a set of materialized views with low granularity aggregation that summarize the data in the data sources. This data is constantly changing and is always up to date with the last changes occurred in those data sources. Data cleaning and aggregation occur in this level.

In order to create a good architecture a close collaboration between IP people and business users can be very helpful. This is because once the design has been done, it must be validated within the organizational context, and also because the maintenance of the warehouse, which implies constant changes, is also directly related to the requirements of users [GAR98].

Also, according to [ANN06] strategic and tactical requirements must be taken into account for the design of the warehouse. Strategic requirements are high performance indicators that allow taking high level decisions, while tactical requirements are functional objectives expressed by end users.

During early stages of the data warehouse design the designers must perform two tasks in parallel. One is collecting the requirements of information from the users, and the other is the analysis of the structure and content of the existing data sources and their intentional mapping to the common data warehouse model. The crucial deliverable is the mapping of the attributes of the data sources to the attributes of the data warehouse tables [VAS02].

2.2.2 ETL process

Once the architecture of a data warehouse has been created, the activities to be performed to introduce the data there are performed. These activities consists on loading, transforming, cleaning and updating data from the data sources, and also on integrating the data into the data source for resolving inconsistencies among different sources [Jar03]. In order to facilitate the execution of the named activities, Extraction-Transformation-Loading tools have been created.

The creation of metadata is a key concept during the creation of the warehouse, since it acts as a blueprint of all the objects that compose the warehouse, like a table, a column or a query. This metadata manages all the process of extraction, transformation and loading of the data performed by the ETL tools, and it serves as a pointer that allows locating objects and data into the warehouse [GAR98].

As it is stated by [VAS02], the task of defining the process that guides all the activities performed by an ETL requires modeling, design and methodological foundations.

The ETL process consists on extracting the data from data sources and then creating some snapshots out of them. Then those snapshots are propagated to an area called the "Data Staging Area" (DSA) where they are cleaned and transformed, to finally store them in the data warehouse data stores, e.g. fact tables and dimension tables.

The conceptual model proposed by [VAS02] is aimed to model the initial phases of the design, with a particular focus on the interrelationships of attributes and concepts, and the necessary transformations that need to take place during the loading of the warehouse.

They call transformation to the process of restructuring the schema and values, or even to the selection and transformation of data. In the model the relationships in the original sources are mapped to relationships in the data warehouse. Also constraints and transformation composition are captured. The design model in [VAS02] follows some steps that conduce to the attribute interrelationships; that model is the conceptual part of the overall ETL process.

Also as this is considered an expensive process, given that it is designed and performed once the data warehouse has been created, some authors such as [MAZ03] even propose graphical models based on UML to assist on the execution of the design of the process.

2.2.3 Quality

Given the many information necessities expressed by the users of the data warehouse there are also different quality requirements that should be fulfilled. For managers of organizations it is important to assess the importance of these requirements, and decide which of them should be given priority during the implementation of the warehouse. This kind of trade-off is due to the fact that there is a limited amount of resources which make not possible to take the measures necessary to satisfy all the possible required quality aspects.

For example a manager must decide which part of the information is more important; a set of data that supports several low level organizational activities, or a set of data that supports only one highly important organizational activity [BAL99].

Once the needed quality improvements have been selected, it is necessary to identify the data sources that support the related organizational processes, to see whether they already exist or not and thus identify potential quality problems. For example that the data set exist but can not be obtained for any reason, which creates a problem of accessibility.

Various projects can be undertaken to improve the quality of the data, such as solving the syntactic differences among customer data records. Solve differences among different sources, or the mechanisms to gather the data that is stored in the data warehouse.

Quality must be implemented in all phases of data warehousing: planning, implementation and maintenance. And as the data in data warehouses are supporting different activities, the manager must make and assess the trade-offs to decide which data sets and activities must be enhanced to have more quality.

2.2.4 Theoretical aspects to be used

As it has been explained, a Data Warehouse is used to structure information from different sources with the aim to support organizational decisions.

The process for structuring the information from different sources and migrate it to a Data Warehouse requires among the main steps, standardization and improvement of its quality. These activities are usually performed by ETL tools once the design of the Data Warehouse has been performed and the structure of its data source is known.

In this project there is not a Data Warehouse definition already made. Nevertheless, the activities performed to structure the data and improve their quality, can be considered as equivalent to some of the activities performed by an ETL tool.

That consideration can be done because the data provided as work resource, which was retrieved from different databases which belong to different companies, has been given a first structure and standardization in the snapshots. Now, through the process followed in this project, quality is improved and a new and more concrete structure is given to the data.

The resulting database can be thus considered as an intermediate resource for a Data Warehouse where the information from different companies will be stored with a unique standard structure; this is because the data stored in the database is ready to be used for analysis and won't possibly need more procedures of standardization or quality management.

Some additional transformations to the data could be needed considering that the data source employed in the Data Warehouse could have a different way to structure the information, given the information requirements used to design it. In that case the information stored in the database created in this project, would be transformed to be adapted to the structure of the data source in the Data Warehouse.

It can be concluded then that the execution of this project is going in an appropriate direction when considering the creation of the Data Warehouse for the future storage and analysis of the information.

Chapter 3

3 Work Methodology

In this chapter, one of the final deliverables of the project, the work methodology to be followed in order to create and document the quality database, is explained. This methodology aims to answer the main questions stated in chapter 1. The activities proposed are based on the results obtained during the literature review, which gave key concepts that are adapted here according to the specific requirements of the project.

Two phases compose the methodology: the first of the following sections describes the phase in which the creation of the database is performed; then, the second section explains the way the resulting data base is tested.

A process model that depicts the steps of the methodology is presented in Appendix E, as a complement to the description given in this chapter. The documents that are result of its application are also enumerated there.

3.1 Creation of the database

During the first phase, most of the important activities to achieve the goal proposed for the project are performed. Such activities are oriented to the creation of the database that is finally used to structure the information provided by the consultant; this database has been built following a sequence of steps that conduce to the generation of quality data, by first executing a cleaning of the original information, and then proceeding with the design and construction of the database.

The steps followed in this phase are detailed in the next subsections.

3.1.1 Understand the data

To facilitate the creation of the database it is necessary to start by realizing which the structure given by the consultant to the information in the snapshots is. Then, after comprehending the underlying structure and relationships it is also important to detect how the data about the software development process was stored into this structure, with the aim to discover possible problems in it. For this, the understanding of the work done by the consultant with companies, in order to get acquainted with the approach he uses to retrieve and analyze their data is useful.

The execution of this first step is supported by the documents provided by the consultant and communication through meetings and emails during the development of the research project.

3.1.2 Data Cleaning

The data cleaning goal is to assess the quality of the data stored in the snapshots and perform the activities that can be necessary to improve it. As it has been told before, one of the goals in the revision of literature was to obtain a clear idea about measurement and improvement of quality; after researching, several descriptions concerning these concepts were found along with approaches aimed to serve as a guide to implement them.

The ideal approach would be one that comprehends all the activities involved in measuring, improving and adding quality in the data, and that clearly indicate how to do this through the use of quality dimensions.

The approaches found and explained in the literature review chapter are:

- Data Quality requirements and analysis [WAN93]: which is focused in the design of a database adding special tags to model quality requirements.
- Framework for analysis of data quality research [WAN95]: which gives importance to the design of a database adding quality dimensions, but also to the organizational context where it will be used.
- TDQM (Total Data Quality Management) [WAN98] which is a methodology that indicates how to implement data quality policies for the creation of information products in information systems. This is done following four steps: definition, measurement, analysis and improvement, which have already been explained in chapter 2.

After an analysis to conclude which of these approaches complains better with the requirements for the creation of the quality database, the TDQM methodology was selected. This is because thorough the four steps that compose it, there are activities oriented to measure and improve quality, but also to create a new database where information of quality can be stored. Therefore it is more complete than the two other approaches that are more focused only on the creation of the database.

Since the TDQM is intended to orientate a complete implementation of quality in the information systems used to produce information products at organizations, it has to be adapted to the context of this project, where only a database must be created and information of quality stored there.

The adaptation of that methodology is done both for the data cleaning and for the creation of the database. In the case of the cleaning, the four steps of the methodology are adapted for the improvement of quality in the snapshots. In the next subsections the four steps to follow are explained.

3.1.2.1 Definition:

During the definition phase of the TDQM [WAN98] methodology the requirements of information and quality for an application are defined. For the cleaning phase only quality requirements must be defined in order to analyze and improve the data in the snapshots. The following activities must be performed:

1. Definition of the quality dimensions [WAN93] that are used to perform the quality assessment of the attributes chosen from the snapshots.
2. After determining the dimensions, the metrics related [BAT06] which are the basis for the assessment are established. Usually in the TDQM methodology the metrics are defined during the measurement step, but in the adaptation made for this project it has been chosen to do it during this step in order to have a clear idea of what must be measured and how, before starting with the measurements.
3. Selection of the data attributes from the snapshots that will be assessed to determine their quality level and improve it in case it is necessary. This attributes are those considered as more relevant for the production of information from the data contained in the snapshots.
4. Get information from the business context in order to define which metrics can be applied to the selected attributes according to the availability of this information.
5. Once the information has been provided, make the final decision about metrics that will be applied and over which attributes.

There are different dimensions according to which quality of information can be evaluated, and the selection of them depends on the context where the information is used. As there is not a standard set of dimensions, some of the most common used can be those chosen for the cleaning of the snapshots.

3.1.2.2 Measurement:

Once the dimensions and their related metrics have been selected, the next phase of the work methodology is to perform the necessary measurements on the attributes in the snapshots. During this activity, according to TDQM, besides measuring the quality level of the information, some business rules are observed. For the context of this project there are neither information requirements nor business rules to comply, so this part of the TDQM is not applied.

TDQM doesn't indicate how to specifically perform the measurements, so it has been decided to follow the approach of localization of errors described in [BAT06] with the goal of finding errors related to the quality dimensions selected, and then count them according to the metrics.

The activities proposed for this phase are:

1. Perform a quantitative assessment. Use an algorithm to localize errors in the attributes previously chosen according to what the metrics defined for every dimension are aimed to measure, and count them.
2. Calculate the level of quality. Once the number of errors is calculated for every metric, it will be possible to use a simple ratio [PIP02] which is a mathematical

calculation that takes the number of data in error and divides it by the total number of data, to finally subtract the result from 1. The final value is a percentage that indicates the level of quality of the information in every dimension; the closer this value is to 1, the higher the quality level is. The use of this ratio is an addition made to the original proposal made by TDQM.

3.1.2.3 Analysis:

In the analysis phase of TDQM the measures obtained with the metrics, are used to investigate the cause of the quality problems. This is done with the help of business experts.

In the case of the data cleaning, this phase is oriented to compare the quality levels calculated with a predetermined bound, to find out how good is the quality of the data regarding a desired level. This activity is therefore a qualitative assessment for which the steps to follow are:

1. Define a standard quality level. A definition of quality values that are acceptable for metrics is done with the help of the consultant. He can indicate for the attributes that are being evaluated during the cleaning, which are the expected quality levels; also, the allowed discrepancy between them and the quality level of the data stored in the snapshots. This level must be a numerical value, so it can be compared with the values obtained through the metrics.
2. Analyze the results of the metrics, by comparing them against the standard parameters of quality that have been specified. The comparison between them and the values obtained in the measurements is done to establish which improvements are necessary for the data.

3.1.2.4 Improvement:

In the improvement phase of TDQM, the procedures necessary to increase the quality level of the data are applied. In the case of improvement of a complete information system, activities such as aligning the creation of information products with the information needs are performed. Nevertheless, as already said in this case it is only necessary to improve the quality of the data in the snapshots, so the correction of errors should be done by applying specific techniques, which are adequate according to the quality dimensions used for the assessment.

One technique that could be used is edit-imputation [BAT06], which is an activity that implies the application of edit rules to the data with the aim of decreasing their inconsistency. An edit rule is an expression that indicates a constraint for the range or type of values that can be stored in a field of a tuple, e.g. “Age > 15 and MaritalStatus = married”, indicates that there could be an inconsistency in case the field Age contains a value less or equal than 15 and the field MaritalStatus has the value married.

The steps to follow in this activity are:

1. Decide whether improvements must be done according to the results of the analysis.
2. In case there are not necessary improvements, this activity is finished.
3. In case there are necessary improvements, it is necessary to get information from the business context. This information is the data considered correct for the values that contain errors, and must be asked to the consultant, since he has a better knowledge of the context where the information was obtained. This restriction is made given the fact that definition of edit rules must be done based on constraints of the business, and also under the use of an established reference domain where values allowed for the attributes assessed are known [BAT06].
4. In case the information is not available, no improvements can be performed. This must be documented and the possible causes of the errors must be explained. The activity must be finished in this point.
5. In case the information is available, the proposed improvements must be performed.
6. Once the proposed improvements have been done, the new quality level of the attributes for which the errors were corrected must be measured with the simple ration [PIP02] and documented.

3.1.3 Design of quality database

After the cleaning of the data has been performed, the data in the snapshots are already complying with quality characteristics that will make it more suitable for the creation of information. It is not possible to talk about a 100% quality level, but it would be acceptable to make an improvement of the data in order to reach a quality level advised by the consultant, regarding the parameters established as allowable in the phase of analysis during the cleaning activities.

The activities for the creation of the quality database proposed next are based on the definition phase of TDQM [WAN98]. As it was described in the cleaning section, this methodology can be applied for the improvement of information systems, and also, as in this second activity, for the creation of information systems that include quality characteristics.

The specific task to be performed here is the creation of the database, then answering the question about how to add quality to a new data source. It starts by following two steps:

1. Define the information needs which will be represented as the entities in the database. Create a data analysis model based on this definition.
2. Define the information quality requirements which are associated to the attributes of the main entities. This definition must be made along with the consultant. In case there are new quality requirements, and thus quality dimensions, create a quality analysis model based on the definition made.

For example, an attribute could be the date of execution of a test in an entity test, and then the information quality requirement could be the timeliness of the information on that date. Every quality requirement can be translated into a measurable characteristic, and in the case of the timeliness for the date there could be a quality attribute called age, which indicates how old the information about the test is.

3. Design the entity relationship model of the database. This is made based on the data analysis model, and then quality analysis model (in case this last one exists).

Some remarks to be done about the enumerated steps are made in the following subsections.

Special requirements

It is important to mention here that as a special requirement for the representation of the information in the database, the confidentiality of the data must be kept. Therefore, it is necessary to define during the creation of the database a mechanism that will allow to make anonymous the information but that in case that it is necessary, let the user of the information make a mapping to find out the identity of the entities. In the case of this project, the name of the projects, systems and subsystems must be maintained anonymous.

Other aspect that will be taken into account is the granularity of the information. As it is described in the reference documentation provided by the consultant, the data collected during the software development process can be analyzed at different levels: Projects, subprojects, teams; systems, subsystems, groups of components, components; given this fact, the data in the snapshots has been stored at the lowest levels, which means it has a fine granularity, and for the creation of information related to the higher levels, such as projects or systems, it is necessary to perform an aggregation of data.

In the case of the database, the granularity given by the snapshots is preserved, since it is considered a proper way to retrieve information at different levels. For example, in the case of requiring information about a component of a specific subsystem it will be easy to create a query that retrieve it from the entity that represents it; and in the case of requiring information about a whole project, a more advanced query that allows retrieving information from different entities related to the project could be necessary. This level of granularity also respect the correctness desired with respect to the adequate use of the elements of an entity-relationship model, since for example the information about components of a system can be stored in entities independent from the entity that represent the system they belong to.

Graphical method

A graphical method suggested in TDQM to represent the analysis models of the database is presented by Information Production MAP (IP-MAP) [BAT06]. The IP-MAP graphical model allows the production of information starting from the knowledge of requirements and quality characteristics desired. Therefore it can be used to analyze the whole process of creation of information in an Information System, but in the context of this project, it is only necessary to define the entities that will compose the database, and their quality characteristics. For this representation the model defines the use of entities representing concepts of the real world and entities representing the quality attributes of those first entities.

As the approach given by IP-MAP has been included in IP-UML [SCA02], which is an extension of UML with the quality profile of IP-MAP, two of the models suggested by that modeling language are used. Nevertheless, since the main focus is on the representation of the entities and their quality, only some of the elements proposed by these models are employed. The models are:

- Data analysis model: To represent the entities of real world that will be included in the database. This is the model to be made in step 1.
- Quality analysis model: To represent the quality requirements defined for every entity. It is only created in case there are specific quality requirements suggested by the consultant. This is the model to be made in step 2.

Interpretability

One important dimension that must be added to the model and documentation of the database is interpretability [BAT06], with the aim to obtain a well documented and correct model. This correctness is related with the necessity of including all the concepts related to the information in the snapshots that are supposed to be represented, and do it avoiding redundancy. Therefore the following aspects must be taken into account in steps 3:

- Integrity constraints that hold among data.
- Metadata about the schema including creator, subject, description, publisher, data, format, source, and language.
- Correctness with respect to the model. This concerns the correct use of the categories of the model in representing requirements. For example the entities in created in the Entity Relationship model should be created only for concepts that have a unique existence in the real world and then have a unique identifier.
- Minimalization. Every part of the information in the snapshots is represented only once in the schema, which will avoid redundancy.
- Completeness. All the information in the snapshots will be represented in the model.
- Pertinence. Not including in the model unnecessary conceptual elements.
- Readability. Create diagrams and schemas that are clear enough. Regarding the diagrams it means to make drawings following aesthetic criteria such as crossing lines the less possible. For schemas the simplicity of representation, which means create them as compact as possible for representing the concepts.

- Normalization. Normalization in the relational model is related to the structure of functional dependencies. In the case of this project it will be enough to reach the third normal form.

3.1.4 Implementation of the database

Once the design of the model for the database has been finished two steps must be followed

1. Definition of the DBMS to be used.
2. Creation of the database and migration of the data from the snapshots.

Previously to the storage of the data from the snapshots into the data model some preprocessing of data can be performed as a preliminary preparation step, in case it is considered necessary. Two activities to follow here are:

- Standardization [BAT06]: Change existing values according to standard formats, e.g. the use of capital letter in the names of systems. It is necessary to define whether there are some standards that should be complied by the data and decide whether this activity is necessary then.
- Data linkage [Fel69]: For every entity identify data which is stored in different snapshots, with the aim of unify all the information about it and then proceed to store it in the database.

3.2 Tests to the information:

As it is necessary to prove that the data from the snapshots stored into the database contains the quality dimensions that were defined during the previous phases, some experiments must be done. Thorough them information can be retrieved out of the data to prove that it can be used. The steps to follow during the test phase are:

1. Define tests to be done.
    - First it must be established which are the objectives when analyzing the data. During this activity it is decided which aspects of the data provided by the companies are desirable to be analyzed. These aspects constitute the basis of the test cases created and executed in order to prove that the data model complies with the quality characteristics proposed.
    - Then it must be defined how the tests will be performed using data mining and/or process mining techniques
2. Apply selected techniques to make analysis and determine the level of quality of the data stored in the database. The results obtained after the completion of this activity are the basis for conclusions and recommendations about the work done.

3.3 Documentation

The quality documentation of the database is one of the three deliverables of this project. It is a result of all the activities performed following the proposed work methodology, and consists of the following documents:

- The report that describes the quality level of the data that were evaluated and improved during the cleaning phase, in order to inform the users of the database which is the quality level of the information stored there.
- The graphical models:
    - Data analysis model: This shows the entities of the real world that are represented in the database.
    - Quality analysis model: This shows the quality requirements of the entities depicted in the data analysis model. It is only created in case new quality requirements are defined by the consultant.
- An entity relationship model that represents the entities and their relationships. The entities are derived from the data analysis model, and in case there are additional quality requirements, the corresponding attributes or additional entities are also represented. In this model it is taken into account to apply characteristics that give it interpretability: Integrity constraints, correctness, minimalization, completeness, pertinence, readability and normalization.
- The report where the entities of the entity relationship model are described along with their attributes and relationships. It is also included metadata about the schema in order to improve the interpretability of the documentation.
- The report where the results of the tests performed to the quality data once stored in the database are described.

Chapter 4

4 Description of procedure to obtain data

This chapter is aimed to present a brief description of the work made by the consultant who provided the data, and of the structure of the data which are stored in the snapshots. With this description the application of the work methodology proposed in chapter 3 is initiated.

4.1 Measurement Database

As it was mentioned in chapter 1, the work of the consultant consists on analyzing information about the software development process in companies, in order to help on its improvement. For this labor, he has developed a procedure which is supported by a tool called "Measurement Database". The objective of this tool is to collect data to calculate project, product and process metrics.

The measurement database architecture is composed by three layers [SIE03]:

- Data collection layer, where the data from different sources in the projects are collected and filtered creating then the snapshots. The snapshots are csv files where the information is structured and are used to pass information to the data storage layer.
- Data storage layer, where the data from the collection layer is stored; here some views of the data can be prepared in order to show them in the data presentation layer.
- Data presentation layer, which provides access to the data stored in the data storage layer through reports and graphs.

The snapshots created in the collection layer are the tool of work provided for this project. The data in the collection layer is collected periodically, so there are snapshots of the data created every time the procedure is performed, and thus historical data is available.

4.1.1 Granularity of the data

For the storage and analysis of the data retrieved from the companies, data are considered at different levels of granularity. These levels are [SIE04]:

- Products are composed by systems, where each system is a version of the product. Each system is made of subsystems, while in turn each subsystem consists of groups of components. Finally, every group of components is built of components. Within the context of the projects analyzed by the consultant, the hierarchy described has 3 to 4 levels.

- A program is a sequence of projects where a project follows another. Projects are organized in a hierarchy of subprojects, and the subprojects in the last level are called teams. Teams are responsible for working on the development of components, which are part of products.

- A team constructs a component by following a process. Every process is composed by a number of steps, called activities, and each activity produces a work product (such as a specification, design, source, tests among others) through a series of tasks. The creation of every work product is based on previous work and refinement of the information.

Some effort is spent on the tasks performed to produce work products; also, during their production defects are injected, which are reported on the components, not on the work products. The defects are solved by creating new versions of work products.

The data in the snapshots are stored at the lowest granularity level, which means for example that all the information about requirements, tests, defects, etc. is associated to the components. Then, when information related to the elements that are composed by these components is necessary, aggregation procedures must be executed to get the desired result.

The aggregation can be done to create information for the following views:

- Product view that contains information about systems, subsystems and groups, which can be done aggregating by components.

- Project view that contains information about projects, subprojects and teams, which can be done aggregating information by activity.

- Process view which is obtained aggregating data by activity type.

Metrics are generated for every one of the three views. The figure 3 shows the data model employed by the consultant, which reflects those concepts:

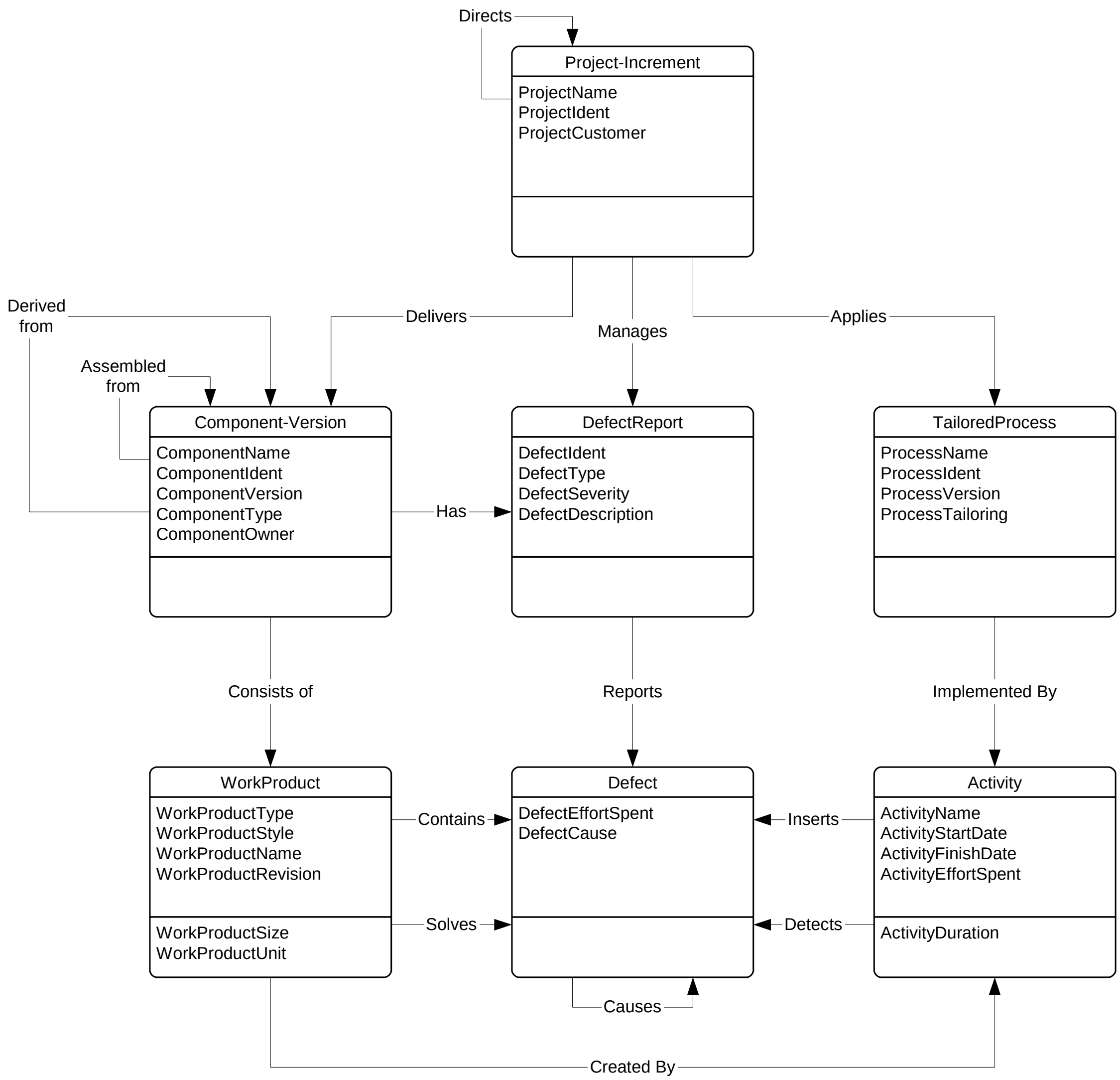

Figure 3. Data model used in the Measurement Database [SIE03]

4.1.2 Data categories

The collection of data in the snapshots is made in the following categories:

- Architecture data
- Project data
- Review data
- Size data
- Defect data
- Case data

- Change data
- Issue data
- Requirements data
- Risk data
- Test data

The structure of every one of this categories and the type of information they store are described in the Appendix A of this document.

Chapter 5

5 Cleaning phase and database design

In this chapter the application of the work methodology proposed in chapter 3, which is based on some of the activities proposed by TDQM methodology, continues. Here the activities followed for data cleaning and the creation of the database are described

5.1 Definition

In this section the definition of the quality dimensions and the metrics to use during the cleaning phase is done. Then the fields of the snapshots that will be assessed according to those metrics are presented.

5.1.1 Dimensions and metrics

Table 1 presents the selection of the data quality dimensions that are used for the assessment of the data stored in the snapshots. Also, the metrics that are proposed for the evaluation are described for every dimension; these metrics are suggested based on the definition of what the related dimension means.

For every metric there is an explanation of the measurement method to be followed; it is also discussed whether additional information to the already contained in the snapshots should be provided by the consultant, in order to make possible to employ such methods. This helps to evaluate the feasibility of applying every one of the methods.

The general idea of every measurement method is to find errors in the data, to then count the number of them found and finally calculate a ratio that gives an idea of the quality level of the attributes evaluated.

Dimension	*Metrics*	*Measurement method*
Accuracy [BAT06]	*Number of syntactic errors*	Create and use rules to find out whether values in attributes are correctly spelled with respect to values in a reference domain. E.g. the stored value is "Jon" and it should be "John". The application of this metric is only possible if the consultant has enough information about the business context, and can provide a reference domain about the values that are allowed for every attribute.

	Number of semantic errors	Create and use rules to find out whether value of an attribute is correct, i.e. it should be "Charles", but is "John". The application of this metric is only possible if the consultant has enough information about the business context, and can provide a reference domain about the values of that are allowed for every field.
	Number of duplicated values	Use of an algorithm to check whether values that should be unique for an entity are stored more than once in the file.
Completeness [BAT06]	*Tuple completeness: Number of values missing in a tuple*	Check number of values that are missing in the fields of the tuples in the snapshots. In order to use this metric it is necessary to determine the reason why null values exist: • Exist but are unknown. • Do not exist. • Exist but it is unknown whether they exist or not. The recognition of null values that are an error and those that are not is only possible if the consultant can provide more information about the business context.
	Attribute completeness: Number of null values of a specific attribute in a relation	Check the number of missing values for an attribute in a snapshot. In order to use this metric it is necessary to determine the reason why null values exist: • Exist but are unknown. • Do not exist. • Exist but it is unknown whether they exist or not.

Consistency [BAT06]	*Number of violations to semantic rules*	Use of algorithm that checks whether values of attributes comply with semantic rules. Find the number of errors regarding rules. The application of this metric is only possible if the consultant has enough information about the business context, and can provide constraints for the data that can be translated into semantic rules.
	Number of outliers	Use of an algorithm to detect outliers in numerical fields. Outliers are data that are anomalous with respect to other data; they can be correct but exceptional values, or values incorrectly recorded. It is possible to identify values that are not common among the data, but it would be necessary to have some information provided by the consultant in order to know whether those values are correct or not.

Table 2. Data quality dimensions for data cleaning

5.1.2 Fields to be assessed

As it was explained in the definition of measurement methods in table 1, it is necessary have more information about the business context and reference domains, so it can be feasible to apply such methods. It was established from conversations with the consultant, that given the big amount of data it was not possible to provide reference domains for many of the attributes.

He also explained that during the process he performs to store the data in the snapshots a number of repairs are performed for known problems. During calculations, everything that can not be mapped on the expected range of values is mapped to the value “OTHER”.

Given these facts the conclusions about the feasibility of applying the measurement methods are:

- Accuracy:
    - Number of syntactic errors: This method is applied only for the attributes for which the reference domains are known. These reference domains are taken from the documentation of the snapshots that is described in Appendix A.

- Number of semantic errors: This method is not applied for any of the attributes. The reason is that even when there are reference domains, there is not an exact knowledge of which the correct values that should be stored in every field are.
- Number of duplicated values: This method is applied to evaluate duplicated values in the attributes that represent the unique identifiers of entities. E.g. the id of the Tests.

- Completeness:
  - Tuple completeness: This measurement method is not applied, given that there is a big amount of data and it is not possible to know for every tuple when the absence of a value can be qualified as incompleteness. It is more practical to measure the level of incompleteness at a higher granularity level.
  - Attribute completeness: This method is applied to measure the number of null values of a specific attribute. E.g. to evaluate the number of incomplete values in the attribute DefectCost of the defects.
- Consistency:
  - Number of violation to semantic rules: Since there is not additional information from the business context, only a few edit rules are defined to be evaluated on the attributes of the snapshots. These rules are defined based on the documentation that is available in Appendix A.
  - Number of outliers: There is not information that could indicate when values contained in numeric fields are outliers, or are correct but unusual. Therefore this rule is applied to identify negative values in the attributes that should contain positive values, such as the effort spent in the correction of a defect.

In the section “Attributes and metrics” of appendix B the attributes of every snapshot category to be evaluated are listed, along with the metrics to be used for the evaluation.

5.2 Measurement

After defining the dimensions and metrics for cleaning and the attributes to be cleaned, the corresponding measurements were performed.

These measurements were made through an algorithm that detects the errors in the values stored for the attribute evaluated. The algorithm is briefly explained next for every metric evaluated:

Accuracy: Syntactic errors

In order to measure the quality level of an attribute selected in a category of information, e.g. "Priority" in the category Requirements Data:

1. For every snapshot :
   a) The values of the attribute to be assessed are read, and every one of them is compared with the values that belong to the reference domain.
   b) In case the value is misspelled or doesn't belong to the reference domain it is counted as an error.
   c) Once all the values of the attribute that are stored in the snapshot have been assessed, the number of errors is counted, and the simple ratio (which indicates the quality level) is calculated for the snapshot. It is calculated by dividing the number of errors by the total number of values evaluated, and then subtracting that value from 1.
2. Once all the snapshots have been checked, the average number of errors is calculated summing up the number of errors of all the snapshots, and dividing the result by the number of snapshots. Also the average simple ratio is calculated by summing up the simple ratio obtained in every snapshot, and then dividing by the number of snapshots.

Accuracy: Duplicated Values

1. In order to measure the quality level of an attribute selected in a category of information, e.g. "defectId" in the category Defect Data:

For every snapshot :

   a) For every value of the attribute in the snapshot it is checked whether it is unique by comparing it with the other values.
   b) In case it is found that the value is duplicated, this duplication is counted as an error.
   c) Once all the errors have been counted the simple ratio (which indicates the quality level) is calculated for the snapshot, dividing the number of errors by the total number of values evaluated, and then subtracting that value from 1.

2. Once all the snapshots have been checked, the average number of errors is calculated summing up the number of errors of all the snapshots, and dividing the result by the number of snapshots. Also the average simple ratio is calculated by summing up the simple ratio obtained in every snapshot, and then dividing by the number of snapshots.

Completeness: Attribute completeness

In order to measure the quality level of an attribute selected in a category of information. E.g. "System" in the category Architecture Data:

1. For every snapshot :

a) Every value of the attribute is checked to verify whether it is empty.

b) In case the value is empty this is counted as an error.

c) Once all values of the attribute in the snapshot have been checked, the number of errors is counted and then the simple ratio (which indicates the quality level) is calculated. It is calculated by dividing the number of errors by the total number of values evaluated, and then subtracting that value from 1.

2. Once all the snapshots have been checked, the average number of errors is calculated summing up the number of errors of all the snapshots, and dividing the result by the number of snapshots. Also the average simple ratio is calculated by summing up the simple ratio obtained in every snapshot, and then dividing by the number of snapshots.

Consistency: Edit rules

In order to measure the quality level, it is checked that the attributes related to the edit rule contain values that are correct according to this, e.g. the attributes "StartDate" and "FinishDate" in the rule StartDate < FinishDate.

1. For every snapshot :

   a) In every tuple the values associated with the attributes are read and for them the edit rule is verified.

   b) In case the values don't accomplish with the rule this is counted as an error.

   c) Once the rule has been evaluated for all the tuples in the snapshot, the number of errors is counted, and then the simple ratio of the snapshot is calculated. This is done by dividing the number of errors by the total number of tuples evaluated, and then subtracting the result from 1.

2. Once all the snapshots have been checked, the average number of errors is calculated summing up the number of errors of all the snapshots, and dividing the result by the number of snapshots. Also the average simple ratio is calculated by summing up the simple ratio obtained in every snapshot, and then dividing by the number of snapshots.

Consistency: Outliers

In the case of the outliers, it is evaluated the presence of negative numbers in the values of attributes that should contain positive values, e.g. "ReworkEffort" in the category Review Data.

1. For every snapshot :

   a) Every value of the attribute to be evaluated is checked to verify whether it contains a negative number.

   b) In case a negative number is found, it is counted as an error.

   c) Once all t he values have been assessed, the number of errors is counted and then the simple ratio is calculated for the snapshot. It is done by dividing the number of errors by the total number of values evaluated, and then subtracting the result from 1.

2. Once all the snapshots have been checked, the average number of errors is calculated summing up the number of errors of all the snapshots, and dividing the result by the number of snapshots. Also the average simple ratio is calculated by summing up the simple ratio obtained in every snapshot, and then dividing by the number of snapshots.

Once all the snapshots have been checked, the average number of errors is calculated summing up the number of errors of all the snapshots, and dividing the result by the number of snapshots. Also the average simple ratio is calculated by summing up the simple ratio obtained in every snapshot, and then dividing by the number of snapshots.

The resulting averages of the measurements can be found in the section "Measurements" of appendix B; these results indicate the quality level of every attribute in the dimensions selected to evaluate them.

5.3 Analysis and Improvement

In this section the analysis of the data obtained during the measurement phase is done. In order to decide which of the attributes evaluated should be taken into account to improve their quality level, it was necessary to define first a reference boundary. This is used to define which attributes have an acceptable quality level.

To define the mentioned boundary the consultant was inquired, given the fact that he has a better knowledge of the context where the information was retrieved. After this consult, it was established that the quality level of the information depends on the type of use of the data. For the consultant the objective of retrieving the data from the companies and analyze it, is to give the users of the information an insight in the process and the project status. Therefore it is correct to allow the presence of errors in the data, since it will show to the users where in the process the mechanisms of creation of data should be improved.

Given the considerations provided by the consultant, it was decided to define a boundary of 80% as an approximate good quality level for the attributes. Based on it in the following subsections the attributes for which quality of data should be improved are mentioned. In some cases, for some of the attributes that obtained more than 80% there is also an analysis of the possible reason for the corresponding value.

In every case there is also an explanation about whether the necessary improvements can be done or not, given the knowledge of the business context.

5.3.1 Company 1

5.3.1.1 Completeness

Defect

The completeness dimension was assessed for the following attributes. Since there are not known values that can be used as reference, no improvements are possible to be done:

Subcategory 1

- DefectEstimate: 0%
- DefectCost: 0%
- Injected: 0%
- Detected: 0%
- DefectAnalysis: 44.88%
- DefectResolution: 2.7%
- DefectEvaluation: 78.49%
- DefectFinish: 78.49%

Subcategory 2

- DefectCost: 0%
- Injected: 0%
- Detected: 0%
- DefectAnalysis: 21.68%
- DefectResolution: 2.64%
- DefectEvaluation: 71.40%
- DefectFinish: 71.40%

Subcategory 3

- Defect_type: 6.31%
- Caused_during: 79.38%
- Act_total_eff: 57.31%
- Analysed_time: 31.95%
- Resolved_time: 47.46%
- Evaluated_time: 41.08%

In the case of the defect data there is not a known reason why the null values exist, but it can be identified that these data are related with effort, costs, phases of the project, and dates involved on the resolution of the defects; therefore, it could be said that the most probable reason for incompleteness in this case is that information existed but it was not stored.

5.3.1.2 Accuracy: Syntactic errors

Some of the following attributes obtained a quality level lower than 80% in the Accuracy dimension when evaluating syntactic errors. For these attributes, reference domains were used to check whether the values stored are correct. It was found that in many cases the values stored don't belong to the reference domains which were established from the information in the documentation of the snapshots. This explains the presence of the errors.

Given this fact, it was asked to the consultant whether the values that were found should be also included in the reference domains, with the aim to add them and perform a new assessment. According to his answer improvements could be done for some of the attributes since the correct values were indicated; for some others no improvements were made, since it is not sure that the error values belong to the corresponding reference domains.

For the values not corrected it must be taken into account that according to the consultant, the errors that appear in a set of snapshots taken on an early date are improved with time in snapshots taken on posterior dates.

Defect

Subcategory 1

- DefectStatus: In the case of this attribute, the found value that doesn't belong to the reference domain is "WontFix". According to the consultant this value could be replaced by "Rejected", which belongs to the reference domain. Before making the change the quality level of the attribute was 97.94%; after making the appropriate change the new quality level is 100%.

Subcategory 2

- DefectStatus: In the case of this attribute, the found value that doesn't belong to the reference domain is "WontFix". As in the previous category, the value was changed to "Rejected". The quality level before performing the change was 92.35%; after the improvement the new quality level is 100%

- Severity: 79.56%

  The found values that don't belong to the reference domain of this attribute seem to be values that correspond to the value stored in the attribute Priority. For example, in the cases the value "M" appears in this attribute in a record, the value "Medium" appears in the attribute Priority for the same record; the same happens with the values "L" and "Low", and "H" and "High". It is not known which should the correct values stored in the cases where “L”, “M” and “H” appear, therefore there is not possibility to make a correction to improve the quality level.

- Priority: 76.56%

  The found values that don’t belong to the reference domain of this attribute are values that belong to the reference domain of the attribute Severity. It seemed that there was a switch on the values of these attributes in the records where the errors were found, but after analysis of the values, this hypothesis was discarded. The values in the attribute Severity were correct in the cases where the attribute Priority was not. There is not knowledge of the correct values that should be stored instead of the error values; therefore the improvement on the quality level is not possible.

Subcategory 3

- Priority: In the case of this attribute the errors found were not mostly related to values that don’t belong to the reference domain used. Some of the values stored were syntactically incorrect, for example the value “Hign” was stored instead of the value “High”. The quality level of this attribute was initially 94.51%. Once the corresponding improvements in the error values were made, the new quality level is 99.68%.
- Crstatus: 92.25%

Review

- State: 99.93%

Size

- Unit: 0%

The only value stored in all fields is “ncsl”. Despite the reference domain is known, there is not certainty of which are the values that should be stored in every field. Therefore there are not possible improvements to be done.

5.3.1.3 Consistency: Edit rules

Defect

Subcategory 3

The following fields were evaluated for checking the consistency of the edit rule:

- DefectAnalysis < DefectResolution. The quality level obtained was 74.17%.

It is possible that the dates that are wrong were known but they were incorrectly stored. Given that there is not additional knowledge of the business context, it is not possible to determine the correct values of these dates. Therefore the quality level of these attributes can not be improved.

5.3.2 Company 2

5.3.2.1 Completeness

For every of the following attributes there are not reference values that can be used to correct the null values, therefore there are not possible improvements to be done.

Case

- CaseFinish: 0%

The most probable reason for the existence of null values in these fields is that the dates of finalization existed but they were not stored in the database.

Change

- ApprovalDate: 64.68%
- Raised: 74.78%
- Assigned: 23.30%
- Completed: 17.06%
- Approval: 54.85%
- Approved: 64.67%
- Closed: 36.17%
- Cancelled: 10.54%
- Rejected: 1.8%

Since the data related with these fields corresponds to the different dates when the change management process happened, the most probable reason for the incomplete fields is that the corresponding information existed but it was not stored.

Defect

- Severity: 38.77%
- DefectFinish: 79.36%

The most probable reason for the presence of incomplete values is that the data for these fields existed but it was not stored.

Issue

- Completed: 57.45%
- Closed: 69.68%

Since the data that correspond to these fields is associated with some of the fields where changes of state in the revision of the issues happened, it could be concluded that most probably the information was known but not inserted.

Requirements

- Priority: 49.05%
- ReqPreparation: 46.82%
- ReqExecution: 27.99%
- ReqFailed: 9.11%
- ReqPassed: 23.15%

The most probable reason for the existence of these incomplete fields is that the information existed but it was not stored. This is because the data corresponds to the dates where the change of status of the requirements happened, and also to the priority it was given to them for being implemented.

Risk

- Raised: 66.29%
- Assigned: 63.13%
- Completed: 53.06%
- TargetDate: 64.49%

Since those fields correspond to the dates when the state of the evaluation of the risks happened, the most probable reason for the incompleteness is that the information of the dates existed but it was not stored.

Test

- TestPreparation: 0%
- TestExecution: 0%
- TestFailed: 0%
- TestPassed: 0%

Since there are not reference values that can be used to fill the null values, no improvements can be done. Again, the most probable reason for the existence of those values is that the information about the dates when the execution of the test happened was known, but this was not stored in the database.

5.3.2.2 Accuracy: Syntactic errors

As it happened with the accuracy errors described for company 1, the evaluation of accuracy for the following attributes showed that there were values stored which doesn't belong to the reference domains used. It was also asked to the consultant about the correctness of these values, in order to know whether they should be added to the corresponding reference domains, and then perform new assessments.

According to the answer obtained, the errors that appear in snapshots taken on an early date are corrected in snapshots that were taken on posterior dates. It is not sure that the found error values belong to the reference domain, therefore no improvements were made.

Case

- CaseStatus: 1.21%

Change

- ChangeStatus: 39.30%
- ChangeState: 88.97%

Defect

- DefectStatus: 87%
- Priority: 83.46%

Issue

- IssueStatus: 69.68%
- IssueState: 94.5%
- Criticality: 99.24%

Risk

- RiskStatus: 67.12%
- RiskState: 90.1%

5.3.2.3 Consistency: Edit rules

Project

The following attributes were evaluated to check whether they comply with the edit rule:

- Psn_Start < Psn_Finish. The quality level obtained was 58.27%

Requirements

The following fields were assessed to check whether they comply with the edit rule:

- ReqExecution < ReqFailed. The quality level found is 62.13%

In both cases it is possible that real values of the wrong dates were known but the values were not correctly inserted. As there is not additional knowledge of the business context, no improvements are possible.

5.4 Database design

After analyzing the information stored in the snapshots and understanding the granularity levels used by the consultant to structure the data, the Data Analysis model was produced. This model is based on some of the stereotype classes suggested by IP-UML and is shown in figure 4.

Since no additional quality dimensions were defined for the data that will be stored in the database, there is not a Quality Analysis model to be elaborated. The entities that are finally represented in the design model which is the Entity Relationship model are based on the classes of the Data Analysis model.

Figure 5 presents the Entity Relationship model, where only the entities and their relationships are depicted. The documentation of the entities and attributes can be found in Appendix C.

Regarding the documentation and the model, it is taken into account to add the interpretability dimension to make them readable, correct, complete and pertinent.

Also during the design of the database it was decided that in order to make anonymous the information required by the consultant, the names of the Companies, Systems, Subsystems, Groups of components, Components, Tasks, Programs, Projects, Teams, Resources and Deliverables will be fictitious. There is an external file where the real names can be consulted and mapped to the data stored in the database.

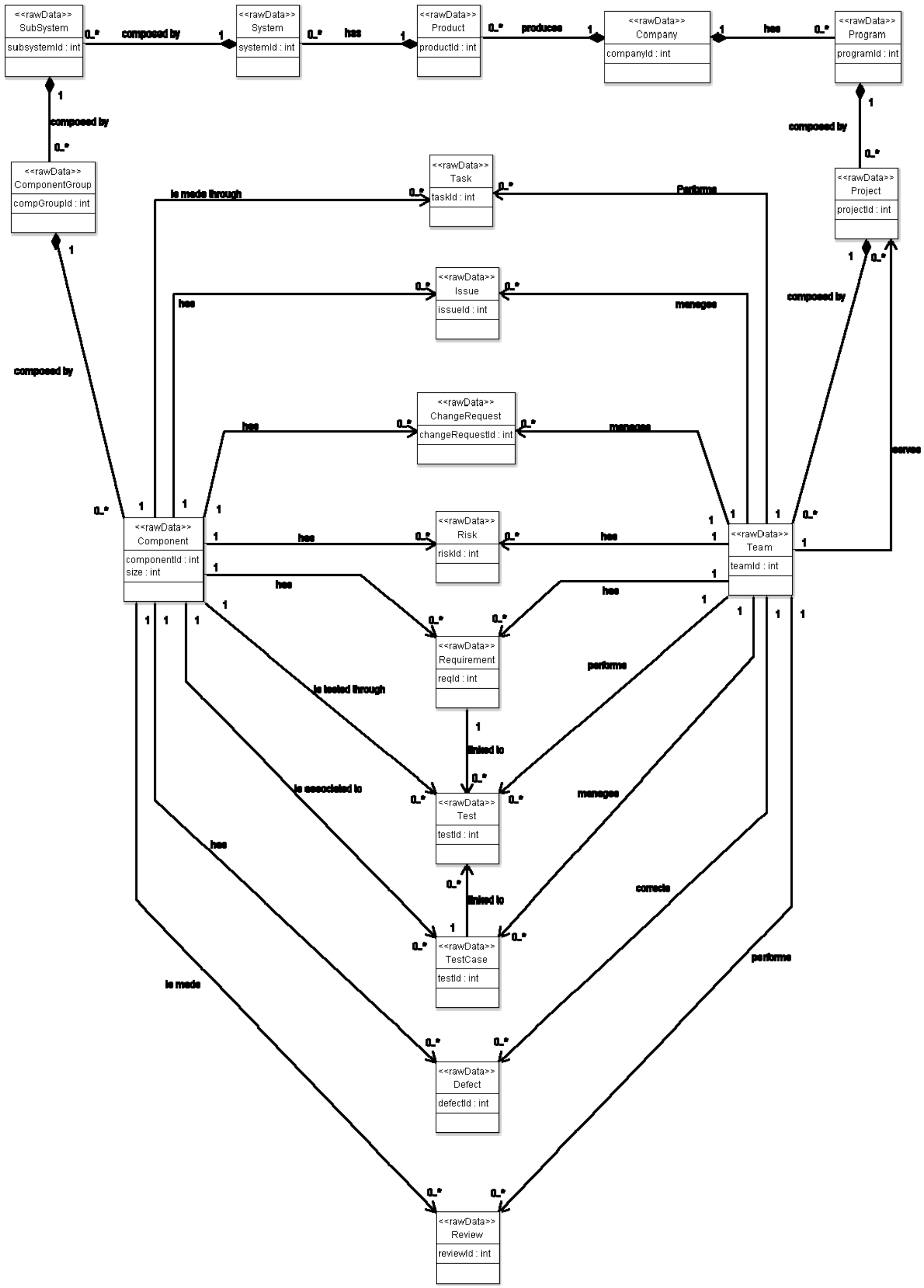

Figure 4. Data Analysis model

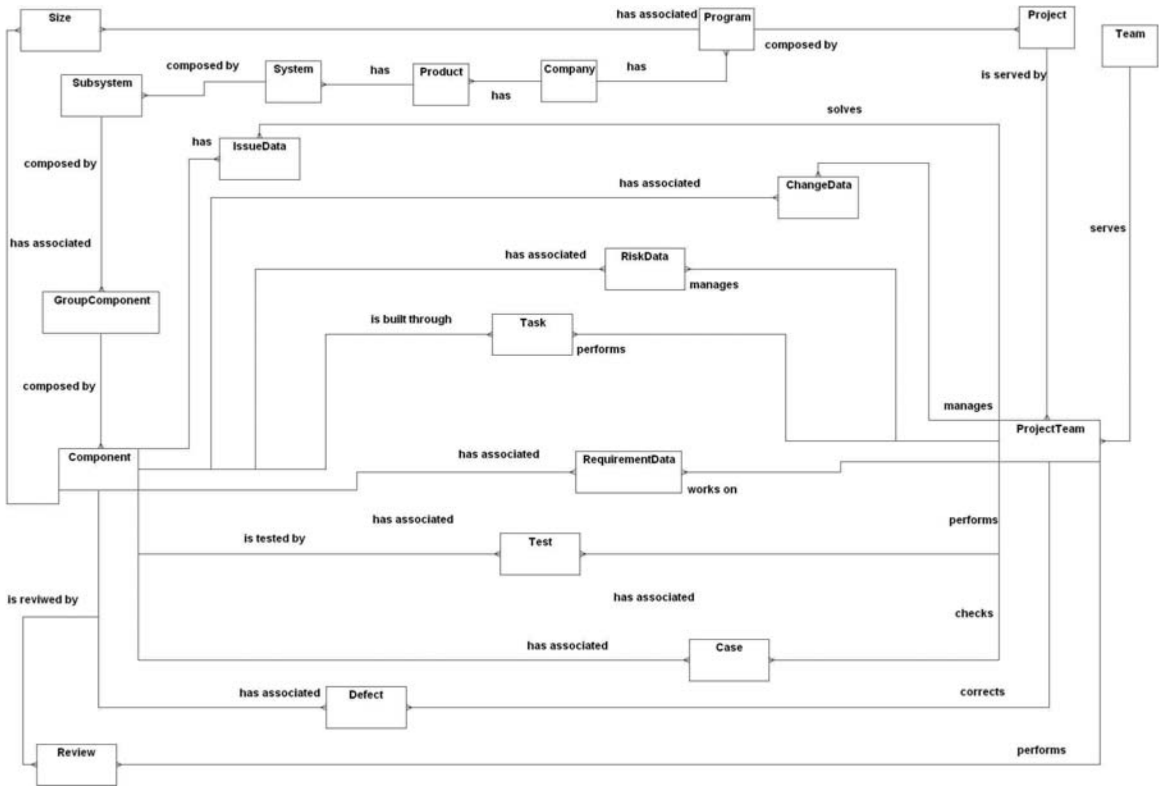

Figure 5. Entity Relationship model

Chapter 6

6 Tests

In this chapter the tests performed over the data stored in the database are described. They are based on real metrics that the consultant obtains in his job with companies. The goal is to demonstrate that the created database can be used to retrieve data to be processed, in order to generate information about the software development process of the companies.

That information could be for example related with effort of people working on different activities, or duration of tasks, among others. It would be therefore employed to calculate the metrics that are useful to analyze how different tasks during the development of applications are being performed.

The results of the tests are presented in Appendix D.

6.1 Description of the test

Through his job with the companies, the consultant uses the data stored in the snapshots to produce metrics that indicate levels of productivity, quality and timing in the software development process. The determination of the metrics to be calculated depends on the goals of every company. For example, for a company it could be essential to know what the effort implied in requirements management activities is, with the aim to improve the way they are performed; in that case the metrics will be related with requirements management.

The tests described in table 3 are planned based on the metrics proposed in [SIE07] and [SIE04]. Every one of them is aimed to measure one aspect of quality on the software development process.

Quality characteristic	*Metric*	*Description*	*Definition*
Maintainability-Changeability	Change duration	How long does it take to implement a change?	Σ change.duration / # change
Maintainability - Compliance	Compliance	To what extend has the required functionality been provided	% req.status [met]

Reliability - Maturity	Defect Severity	How severe were defects found?	% defect.severity [very high+urgent]
Performance - estimation	Effort distribution	How is the effort distributed over project activities?	Σ task.actualwork [task.tasktype]
Performance - effectiveness	Review Coverage	To what extend have deliverables been reviewed	Σ review.size [accepted]/ Σ document.size
Earned value analysis	Actual Cost of the Work Performed ACWP(t)	Cumulative work spent on tasks actually completed , i.e. the sum of the Actual Work of all tasks that have Actual Finish <= t.	Σ task.actual [completed] upto time t

Table 3. Metrics for planning tests

6.1.1 Change duration

Steps to calculate the metric were implemented in an algorithm:

1. Retrieve the dates when snapshots about change information were taken.
2. For every date retrieve assigned date and closed date of changes that have closed status and are associated to the program with Id 18.
3. For every change calculate the duration in number of days it took to implement it.
4. Sum up the results of duration of all changes.
5. Divide the result of the sum by the total number of changes that were evaluated.

The results are presented in figure 1 of appendix D.

6.1.2 Compliance

The steps to obtain the metric were implemented through an algorithm:

1. Retrieve the dates when snapshots about requirements information were taken.
2. For every date calculate the number of requirements that exist and are associated to program with id 11.
3. For every date calculate the number of requirements that are in state closed and are associated to program with id 11.

4. Calculate the percentage of met requirements dividing the number of requirements in state closed, by the total number of requirements, and then multiplying by 100.

The results are presented in figure 2 of appendix D

6.1.3 Defect Severity

The steps to obtain the metric were implemented through an algorithm:

1. Retrieve the dates when snapshots about defects information were taken.
2. For every date calculate the number of defects associated to program with id 2.
3. For every date calculate the number of defects associated to program with id 2 and which severity is S = show stopper/ blocker, or A = major function affected.
4. Calculate the defect severity dividing the number of defects with severity S or A by the total number of defects, and then multiplying by 100.

The results are presented in figure 3 of appendix D.

6.1.4 Effort Distribution

The steps to obtain the metric were implemented through an algorithm:

1. Retrieve the dates when snapshots about tasks information were taken.
2. For every date obtain the actual work of tasks which type of activity is REQ (Requirements), DSG (Design) or TST (test) and are related to program with id 8.
3. For type of activity sum up the actual work of all the related tasks.

The results are presented in figure 4 of appendix D.

6.1.5 Review Coverage

The steps to obtain the metric were implemented through an algorithm:

1. Retrieve the dates when snapshots about review information were taken.
2. For every date calculate the sum of the sizes of reviews associated to project with id 49.
3. For every date calculate the sum of the sizes of reviews associated to project with id 49 and which state is accepted.
4. Calculate the review coverage by dividing the result of the sum of sizes of accepted reviews, by the sum of sizes of all the reviews; then multiplying the result by 100.

The results are presented in figure 5 of appendix D.

6.1.6 Actual Cost of the Work Performed

The steps to obtain the metric were implemented through an algorithm:

1. Retrieve the dates when snapshots about task information were taken.
2. For every date calculate the sum of actual work of all the tasks associated to program with id 8, and which actual finish date is previous to the date of the snapshot.

The results are presented in figure 6 of appendix D.

6.2 Conclusion

The tests described and performed have been useful to confirm that the structure given to the original data provided by the consultant in the database, is appropriate to generate information about the software development process.

Six tests were performed, every one related with the generation of a selected metric. For every one of them, an algorithm was created using queries over the database and then making some type of processing required to calculate the related metric.

The results obtained and shown through different graphics, were useful to obtain an idea of the time and effort spent on the execution of some activities related with areas of software development such as requirements and risks management. Initial conclusions were derived out of the results, and also new inquiries about the reason of the behaviors observed, or the reason why in some opportunities the information generated seems to be not accurate enough.

It can be concluded that in the future many ideas could be proposed to analyze the different areas of the software development process for which data is stored in the database. Researchers will be able to detect how the related activities were performed, detect possible errors, propose new questions about the results found and try to solve them by creating the needed queries, or consulting with the provider of the data.

Chapter 7

7 Conclusions and Future work

The goal of this project was the creation of a quality database where information about the software development process from two companies was structured, and stored after going through a procedure of quality improvement. This was made to accomplish with the objective of the consultant who provided the data, who intends to contribute with academic research, and obtain also a feedback on techniques he could use to make better his own work procedures.

As final result, three deliverables are available: the quality database, the quality documentation of the database, and a work methodology which was followed for their creation.

The work methodology has been proposed as a result of literature review on quality data. It is intended to explain the steps that should be followed to improve the quality on the data provided by the consultant, and then create a database where to store these data and keep their quality.

Through the data cleaning phase of the work methodology, the data provided by the consultant was analyzed in three quality dimensions: consistency, completeness and accuracy. Some of the attributes of the snapshots were selected for that analysis, and their quality was measured using some metrics proposed for every dimension. Afterwards, improvements that were possible according to the availability of business context information were performed. The final results obtained after these activities have been documented to be used as reference of the quality of the data.

Once the cleaning phase finished, the analysis and design of the database were made, and finally the database was implemented. The work methodology proposes the creation of the database taking into account possible new quality dimensions. In this particular case, the consultant didn't suggest new quality dimensions, so no special entities or attributes were created apart from those used to store the information provided by the companies. Some special considerations made were to keep anonymous the main data about the companies and give interpretability to the database and the documentation of the same.

The documentation contains the description of the entities that were created, their relationships, data types and kind of information they store.

Regarding the tests to prove that the structure given to the information is useful, some calculations of metrics were performed. Those metrics are based on the actual measurements that the consultant makes through his job, in order to generate indicators of how the software development process is being followed, and then suggest improvements. Many of them are specifically oriented to analyze a certain area of the process which the company has the goal to improve.

The results obtained with the measurements are useful to interpret some facts concerned with the development in the companies, but also show that in several cases the data could be incomplete and thus calculations were not possible to be done for all the dates. It must be taken into account that the indicators are not 100% reliable since neither the data has a 100% quality level.

Talking about the benefit of the work performed during this project for future researchers, it can be said that they will have an inside on the quality and structure of the data, and will be able to use it in order to analyze the software development process of the companies. That analysis could be done through the calculation of metrics, or the application of methods such as process mining that can help to understand the procedures followed to execute different activities; for example those involved in requirements management.

Additionally, among the future perspectives for using the database and the results provided by the project, is the creation of a data warehouse to store the information. This can provide the consultant an idea about how to introduce also the use of a data warehouse in his own job.

A literature review about the topic of data warehouses has been presented in order to give the perspective of how it could be implemented in future work. As it was concluded, the work made during this project is part of what is necessary for the creation of a data warehouse; this is because activities for quality improvement, standardization and structuring of the data performed here, are part of a normal data warehouse design and implementation. Therefore the database is an intermediate result which can be easily adapted for the migration of the data to the data structure of a warehouse.

The work methodology is also useful for the consultant because he could consider complementing the current procedures he follows to analyze the data, with the activities proposed to perform data quality measurement and improvement. Those activities are useful for him to get an inside of the errors that are made when data is stored by people in companies; therefore after performing quality measurements, he could indicate to them the kind of improvements that they should do in order to avoid those errors when storing the information.

Once the improvements to the methods to store data would be made, the consultant could have more quality data stored in the snapshots. This would be useful to generate more accurate calculations of the metrics he uses to indicate enhancements that must be done in the software development process of the companies. For example on the procedures they use to correct defects, which could be making them to spend more effort than estimated.

Additionally, the work methodology can also be useful for giving an idea to the consultant about how to structure the data after it is stored in the snapshots. Currently he already uses a database to make this structure, and he employs the information there to create the metrics that will give to the companies, indicators of their performance during the software development process. Nevertheless, taking into account the perspective of adding more quality to the data, the consultant could figure out whether there are quality dimensions that he could add to the information in his database, in order to make it more useful for his job; with this in mind he could add some new entities to his database, which would be used with the aim to add the considered quality dimensions.

As already explained the database created during this project will be used by other researchers in future work. Having in mind the quality figures presented in chapter 5, they will be able to indicate the level of reliability of the results they obtain with their job. The consultant could also use these results to understand the errors made by the companies that provided data for this project.

Finally, the work methodology proposed here, can be followed in the future by other students in case they receive new data from the consultant and want to improve their quality and store it in the database for their job.

8 Appendix A: Snapshots Documentation

In this appendix a description of the structure of the snapshots for both companies is made. In both cases the snapshots are classified in different categories according to the type of data they store. For every type of snapshot it is indicated the type of information it contains, the list of the fields that compose it, the description of every field, and the data type stored in every field.

The snapshots from Company 1 are shown in tables of section 1, and the snapshots from Company 2 are shown in tables of section 2. The information used for the description was obtained from original documentation of the design of the tool that generates the snapshots, and comments made to complement given by the provider of the information.

8.1 Company 1

8.1.1 Architecture data

The architecture data represents the common key among all the snapshots.

Field name	*Description*	*Values – Data type*
Data Set	Identification of the dataset in the snapshot. Dataset = set of snapshots from the same source at regular intervals. Project – All teams – Type of data	String
History Date	Date the snapshot was taken	Date
Product_SubSys	Short indicator for the subsystem/cmpgroup/component. Will be mapped to the proper item in the database.	String
System	System name – System = topmost deliverable; often Program name and System name look alike.	String

SubSystem	Subsystem from which the snapshot was taken – Continuous database the snapshot is taken from. Due to multisite cooperation multiple databases can be included as source. Subsystem = major part of system; system = assembly of 1 or more subsystems.	String
CmpGroup	Name of the components group. Cmpgroup is major part of subsystem; subsystem = assembly of 1 or more cmpgroups	String
Component	Name of the component. Component is major part of cmpgroup; cmpgroup = assembly of 1 or more components	String
External	Field not present in all snapshots. Indicates that some part of a system is not created by the program / project / team but delivered by or bought from an external party	Integer

Table A1. Architecture Data

8.1.2 Defect data

The information in the snapshots that correspond to the Defect data is related with the defects, tests requirements, tests cases, tests steps, and bugs that were used or originated during the test phase in the software development process at both companies.

In this group there are three types of snapshots which have different fields, which are described in the following tables. The information used for the documentation was obtained from the document [SIE08] provided by the consultant, and from the master thesis [URE08] and [IBE08].

Subcategory 1

Field name	***Description***	***Values – Data type***
Data Set	Identification of the dataset in the snapshot – Dataset = set of snapshots from the same source at regular intervals. Project – All teams – Type of data	String
History Date	Date the snapshot was taken	Date
Project	from configuration – Name of the project Project has 1 or more teams to serve it	String
Team	from configuration – Name of the team in charge of the creation of a component Team has 1 or more projects to serve	String
ProdSubSys	from configuration – Short indicator for the subsystem/cmpgroup/component. Will be mapped to the proper item in the database	String
System	System name – System = topmost deliverable; often Program name and System name look alike.	String
Version	from QC –Version of the system or subsystem etc.	String
DefectId	from QC – Identification of the defect being documented	Integer
DefectType	from QC – Description of the defect. This data is not always available. The data contain real defects (PR), changes to the requirements (CR) and impact of normal work (IR). The classification is not always known immediately	String
DefectState	based on status from QC – This is the same as CrStatus but mapped onto a standard set of	String

<table>
<tr><td></td><td>states.<table>
<tr><td>Main State</td><td>Substate</td><td>Meaning</td></tr>
<tr><td>Deferred</td><td>On hold</td><td>Not solved</td></tr>
<tr><td rowspan="4">Open</td><td>Submitted</td><td>Reported</td></tr>
<tr><td>Analysis</td><td>Investigation</td></tr>
<tr><td>Resolution</td><td>Fixing</td></tr>
<tr><td>Evaluation</td><td>Verifying the fix</td></tr>
<tr><td>Closed</td><td>Closed</td><td>Closed after fixing</td></tr>
<tr><td rowspan="4">Rejected</td><td>Duplicate</td><td>Already reported</td></tr>
<tr><td>Nonrepro</td><td>Can’t solve, not reproducible</td></tr>
<tr><td>Rejected</td><td>Won’t solve, live with it</td></tr>
<tr><td>By design</td><td>Shouldn’t solve, intended behavior.</td></tr>
</table></td><td></td></tr>
<tr><td>DefectStatus</td><td>from QC – State is the main state, open, closed, deferred or rejected. Status is the substate of the main state. To keep them separate is easier when handling this data in queries.</td><td>String</td></tr>
<tr><td>DefectEstimate</td><td>from QC – Not in all snapshots. Estimated cost to repair the defect (PR) or to implement the change (CR) or the task (IR). In many cases there is a symbol “?”, instead of data.</td><td>String</td></tr>
<tr><td>DefectCost</td><td>from QC – Actual cost to repair the defect (PR) or to implement the change (CR) or the task (IR). In many cases there is a symbol “?”, instead of data.</td><td>String</td></tr>
<tr><td>Severity</td><td>from QC –
S = show stopper/ blocker
A = major function affected</td><td>String</td></tr>
</table>

	B = minor function affected C = cosmetic D = all other How much the defect/change affects the performance or behavior of system	
Priority	from QC – Priority given to the defect for its treatment. [From defects project] Ordering to address things in the project: 1 = Low 2 = Medium 3 = High 4 = Top How soon we want the issue to be solved	String
Injected	from QC – This variable explains in which phase the defects has been caused: 1 = Requirements definition and specification 2 = Architectural design 3 = Implementation 4 = Integration 5 = Qualification 6 = Not applicable There is data of this kind but not much, in many cases there is a symbol "?"	String
Detected	from QC – When was the defect detected. This variable explains in which phase the defects has been discovered: 1 = Requirements definition and specification 2 = Architectural design 3 = Implementation 4 = Integration	String

	5 = Qualification 6 = Not applicable There is data of this kind but not much, in many cases there is a symbol "?"	
DefectStart	status transition date from QC – Creation of the defect	Date
DefectAnalysis	status transition date from QC – State set to in-analysis [sub of analysis]. Date when the analysis started.	Date
DefectResolution	status transition date from QC – State set to in-resolution [sub of resolution]. Date when the defect entered to resolution	Date
DefectEvaluation	status transition date from QC – state set to in-evaluation [sub of evaluation]. Date when the defect entered to the evaluation process.	Date
DefectFinish	status transition date from QC – state set to closed or to rejected	Date

Table A2. Defect Data 1

Subcategory 2

Field name	***Description***	***Values – Data type***
Data Set	Identification of the dataset in the snapshot – Dataset = set of snapshots from the same source at regular intervals. Project – All teams – Type of data	String
History Date	Date this snapshot was taken	Date
Program	Program – architectural components. This is the name of the programme of which the project is a part.	String

<table>
<tr><td></td><td colspan="3">Program = collection of Projects</td><td></td></tr>
<tr><td>Project</td><td colspan="3">from configuration – Name of the project
Project has 1 or more teams to serve it</td><td>String</td></tr>
<tr><td>Team</td><td colspan="3">from configuration – Name of the team in charge of the creation of a component
Team has 1 or more projects to serve</td><td>String</td></tr>
<tr><td>ProdSubSys</td><td colspan="3">from configuration – Short indicator for the subsystem/cmpgroup/component. Will be mapped to the proper item in the database</td><td>String</td></tr>
<tr><td>System</td><td colspan="3">System name – System = topmost deliverable; often Program name and System name look alike.</td><td>String</td></tr>
<tr><td>Version</td><td colspan="3">Version of software. Data not always available</td><td>String</td></tr>
<tr><td>DefectId</td><td colspan="3">Id of the defect</td><td>Integer</td></tr>
<tr><td>Defect Type</td><td colspan="3">Description of the defect. This data is not always available.
The data contain real defects (PR), changes to the requirements (CR) and impact of normal work (IR). The classification is not always known immediately.</td><td>String</td></tr>
<tr><td rowspan="7">DefectState</td><td colspan="3">based on status from QC – This is the same as CrStatus but mapped onto a standard set of states.</td><td rowspan="7">String</td></tr>
<tr><td>Main State</td><td>Substate</td><td>Meaning</td></tr>
<tr><td>Deferred</td><td>On hold</td><td>Not solved</td></tr>
<tr><td rowspan="4">Open</td><td>Submitted</td><td>Reported</td></tr>
<tr><td>Analysis</td><td>Investigation</td></tr>
<tr><td>Resolution</td><td>Fixing</td></tr>
<tr><td>Evaluation</td><td>Verifying the fix</td></tr>
</table>

	Closed	Closed	Closed after fixing	
	Rejected	Duplicate	Already reported	
		Nonrepro	Can't solve, not reproducible	
		Rejected	Won't solve, live with it	
		By design	Shouldn't solve, intended behavior.	
DefectStatus	from QC – State is the main state, open, closed, deferred or rejected. Status is the substate of the main state. To keep them separate is easier when handling this data in queries.			String
DefectCost	from QC – Actual cost to repair the defect (PR) or to implement the change (CR) or the task (IR). In many cases there is a symbol "?", instead of data.			
Severity	This variable explains the impact of the defect: 1 = S(Showstopper / blocker) 2 = A (Major Function affected) 3 = B (Minor Function affected) 4 = C (Cosmetic) 6 = D (All Others) How much the defect/change affects the performance or behavior of system			String
Priority	Priority given to the defect for its treatment. [From defects project] Ordering to address things in the project: 1 = Low 2 = Medium 3 = High 4 = Top			String

	How soon we want the issue to be solved	
Injected	This variable explains in which phase the defects has been caused: 1 = Requirements definition and specification 2 = Architectural design 3 = Implementation 4 = Integration 5 = Qualification 6 = Not applicable Data not available	String
Detected	This variable explains in which phase the defects has been discovered: 1 = Requirements definition and specification 2 = Architectural design 3 = Implementation 4 = Integration 5 = Qualification 6 = Not applicable Data not available	String
DefectStart	status transition date from QC – Creation of the defect	Date
DefectAnalysis	Status transition date from QC – State set to in-analysis [sub of analysis]. Date when the analysis started.	Date
DefectResolution	Status transition date from QC – State set to in-resolution [sub of resolution]. Date when the defect entered to resolution	Date
DefectEvaluation	status transition date from QC – state set to in-evaluation [sub of evaluation]. Date when the defect entered to the evaluation process.	Date

DefectFinish	status transition date from QC – state set to closed or to rejected	Date

Table A3. Defect Data 2

Subcategory 3

Field name	*Description*	*Values – Data type*
Data Set	Name of the dataset. Project name + date of snapshot	String
History Date	Date this snapshot was taken	Date
System	System name	String
Subsystem	Subsystem from which it was taken the snapshot - Continuous database the snapshot is taken from. Due to multisite cooperation multiple databases The commercial database application uses this name with a different meaning than consultant does Subsystem = major part of system; system = assembly of 1 or more subsystems	String
Problem_number	Unique id within the continuous database Maps onto defect id	Integer
Product_name	used differently by each project	String
Product_subsys	Name of the subsystem where the defect is stored. Product name. Short indicator for the subsystem/cmpgroup/component. Will be mapped to the proper item in the database.	String
Version	Version of the defect. 1 (always)	Integer

Release	Release label. Product name + Release Number	String
Priority	Priority given to the defect for its treatment. Priority of the defect to be solved? [From defects project] Ordering to address things in the project: 1 = Low 2 = Medium 3 = High 4 = Top	String
Severity	Level of severity of the defect – S = show stopper/ blocker A = major function affected B = minor function affected C = cosmetic D = all other	String
Defect_type	Description of the defect. This data is not always available.	String
Problem_type	Depending the hierarchy of the defect, the defect can be parent or child Not of interest; is a check for correctly getting the right records from the source database	String
Request_type	Explain the needs of the resolution of the defect: PR = Problem report CR = Change Request IR = Implementation Request	String - with three possible values
Crstatus	Current state of the defect in the resolution process Analysed Concluded	String

	Created Duplicate Duplicate_analysed Duplicate_concluded Duplicate_evaluated Evaluated In_analysis In_evaluation In_resolution Later_release Not_reproducible On_hold Rejected Resolved Submitted	
Caused_during	when was the defect injected – This variable explains in which phase the defects has been caused: 1 = Requirements definition and specification 2 = Architectural design 3 = Implementation 4 = Integration 5 = Qualification 6 = Not applicable Phases described in snapshots: Alpha testing, Architecture, Beta testing, Component testing, Design, Implementation, Integration testing, Not Applicable, Requirements, Scenarios. The data is not available in all cases.	String
Discovered_ during	when was the defect detected – This variable explains in which phase the	String

	defects has been discovered: 1 = Requirements definition and specification 2 = Architectural design 3 = Implementation 4 = Integration 5 = Qualification 6 = Not applicable Phases described in snapshots: Alpha testing, Architecture, Beta testing, Component testing, Design, Implementation, Integration testing, Not Applicable, Requirements, Scenarios.	
Act_total_eff	Estimated total effort spent on solving the defect.	Float
Create_time	Creation of the record	Date
Submitted_time	Date of submission of the defect. State set to submitted.	Date
In_analysis_time	State set to in_analysis. The analysis started. Data not always available.	Date
Analysed_time	State set to Analysed. Date when the analysis ended. Data not always available.	Date
In_resolution_ time	State set to in_resolution. Date when the defect entered to resolution. Data not always available.	Date
Resolved_time	State set to resolved. Date when the resolution ended. Data not always available.	Date
In_evaluation_ time	State set to in-evaluation. Date when the defect entered to the evaluation process. Data not always available.	Date

Evaluated_time	State set to evaluated. Date when the evaluation ended. Data not always available.	Date
Modify_time	Latest change date of the defect's status. When it is closed or closured.	Date
Modifiable_in	Name of the subsystem (local database of the responsible party) where the changes will be carried out.	String
Discovered_on	The project = MTR-A	String
Team	Team in charge of handling the defect. Field not present in all snapshots.	String
Program	Program – architectural components. This is the name of the programme of which the project is a part Program = collection of Projects Field not present in all snapshots.	String
Scope	Not used.	

Table A4. Defect Data 3

8.1.3 Project data – "Effort Data"

The data stored in the snapshots in the category Effort Data is related with the time, effort, budget spent in projects for software development, and it has been retrieved from different databases. The documentation of the snapshots is based on the documents [SIE08], [SIE02-1] and [SIE02-2].

Field name	*Description*	*Values – Data type*
Data Set	Identification of the dataset in the snapshot	String
History Date	Date the snapshot was taken	Date
Unique_ID	from PSN – Unique ID of the task within the	Integer

<table>
<tr><td></td><td colspan="2">MsProject file, maintained over time (i.e. the same snapshots unless task is deleted and replaced)</td><td></td></tr>
<tr><td>Outline_
Number</td><td colspan="2">from PSN – The structural ordering of the task in the file, 1 comes before 2. 1.1 is the first child, etc.</td><td>String</td></tr>
<tr><td>Milestone</td><td colspan="2">Is the task a Milestone? Yes/No. YES if task is a milestone.</td><td>String</td></tr>
<tr><td>Summary</td><td colspan="2">Is the task a summary task? Yes/No. A summary task is a parent and as such, it is the sum of all its child tasks. YES is the task is a parent.</td><td>String</td></tr>
<tr><td>Name</td><td colspan="2">from PSN – The description of the task in MsProject.</td><td>String</td></tr>
<tr><td>Flag 10</td><td colspan="2">Indicates task completion percentage (either 0 / 100). If it is 0 the task is ongoing. Yes/No (100/0)
For milestones
• set to yes when milestone has been successfully passed
For non-summary tasks
• set to yes when a task is complete; the task will then be included in the work performed during Earned Value calculations</td><td>Integer</td></tr>
<tr><td>Baseline_Start</td><td>Start from Baseline 1 in PSN when available</td><td rowspan="5">The start, finish and scheduled work are the current plan.

When the plan is approved, a baseline copy is saved of the scheduled start, finish and work (hours).

The actual start, finish and work reflect</td><td>Date</td></tr>
<tr><td>Baseline_Finish</td><td>Finish from Baseline 1 in PSN when available</td><td>Date</td></tr>
<tr><td>Baseline_Work</td><td>Work from Baseline 1 in PSN</td><td>Float (minutes)</td></tr>
<tr><td>Start_Date</td><td>Scheduled start from PSN</td><td>Date</td></tr>
<tr><td>Finish_Date</td><td>Scheduled finish from PSN</td><td>Date</td></tr>
</table>

Scheduled_ Work	Scheduled work [Actuals+ETC] from PCteam and PSN	progress. When the actual start is set the scheduled start is set to the actual. For ongoing tasks (Flag 10=0) the finish date is set to the snapshot date. The actual work is always the effort spent between start and finish.	Float (minutes)
Actual_Start	from PCteam		Date
Actual_Finish	from PCteam. Only recorded when the task is 100% complete. Included only for completeness sake.		Date
Actual_Work	from PCteam		Float (minutes)
Text4	The type of activity involved, see explanation. PDSL Activity Type* Project Start System Proposed System Defined Code Complete System Complete System Accepted Project End Project Start Project Implementation Approval SW Components Specified SW Components Available System Validated System Release Approval Project Complete Kick Off Concept Start Product Range Start Design Release		String

	Commercial Release Mass Production Release PDSL Activity Type: • Requirements • Design • Coding • Testing • Management • Support • Problems • Other OR PDSL Milestone: • PS • SP • SD • CC • SC • SA • PE OR OSRP Milestone: • PS (=PS) • PIA (=SP) • SWCS (=SD) • SWCA (=SC) • SV • SRA (=SA) • PC (=PE) OR SPEED Milestone: • KO (=PS) • CS (=SP) • PRS (=SD) • DR (=SC) • CR (=SA) • MPR (~PE) From * For milestones: • **Teammilestone** • **projectmilestone**	

	For non-summary tasks (see Quality Manual) • **REQ** for Requirements/ Specification • **DES** for Architecture / Design • **ARCSUP** for Architecture Support • **IMP** for Implementation / Coding • **ITS** for Integration Test Specification • **ITI** for Integration Test Implementation • **ITE** for Integration Test Execution • **QTS** for Qualification Test Specification • **QTI** for Qualification Test Implementation • **QTE** for Qualification Test Execution • **MGT** for Management/Planning/Tracking/Meeting • **SUP** for CM/QA/Training • **REW** for Problem Solving • **Other** for non-project activities	
Text5	Deliverable. If non-empty indicates that the task has a deliverable with it.	String
Text15	This field is only used in some snapshots. e.g. Healthy – living – Is it the Program. This is the name of the programme of which the project is a part. Program = collection of Projects It is not present in al files	String
Text16	It is the name of the project. Project has 1 or more teams to serve it For all tasks	String
Text17	Is it the name of the team For all tasks Team has 1 or more projects to serve	String
Text25	For all tasks. System Name, e.g. C-STEP System is composed by subsystems	String
Text26	For tasks uniquely related to a single subsystem • Subsystem Name, e.g. SV for Service Layer	String

	Otherwise • (same as) System Name Subsystem is composed by cmpgroups	
Text27	For tasks uniquely related to a single component group. • Component Group Name, e.g. SV_HDMAN Otherwise • (same as) Subsystem Name Cmpgroup is composed by components	String
Text28	For tasks uniquely related to a single component • Component Name Otherwise • (same as) Component Group Name Note: for storage projects related to DVD+RW the Component field is always set to the same value as the Group field because defects are reported only to the level of Component Groups	String
Text29	Release There is not data available in some files.	String
Resource_ Names	From PSN – The name of the persons working on the task, often suppressed for privacy reasons.	String

Table A5. Project Data

8.1.4 Review data

Some of the information used to document the snapshots in this category was obtained from the document [IBE08]. The data contained in these snapshots is related with the activity of reviewing documents used in different phases of the software development cycle.

Field name	*Description*	*Values – Data type*
Data Set	Identification of the dataset in the snapshot – Dataset = set of snapshots from the same source at regular intervals. Project – All teams – Type of data	String
History Date	Date the snapshot was taken	Date
InitiationDate	Day the tasks for review started	Date
KickOffDate	Date the review activity started	Date
LoggingMeetingDate	Date for meeting in the process of review	Date
ClosureDate	Date the review finished	Date
DefectId	Review id	Integer
Project	Name of the developed software project Project has 1 or more teams to serve it	String
Team	Name in charge of writing the document under review. Team has 1 or more projects to serve.	String
System	System name System = topmost deliverable; often Program name and System name look alike.	String
ProdSubsys	Name of the subsystem where the review is being made. Short indicator for the subsystem/cmpgroup/component. Will be mapped to the proper item in the database.	String
Pool	From which resource pool the moderatos of the review is coming	String
WorkProductTitle	Title of the document under inspection / review	String

ActivityType	The type of activity involved in each review: 1 = Requirements (REQ). 2 = Design (DES) 3 = Coding (IMP) 4 = Integration Test Specification (ITS) 5 = Integration Test Implementation (ITI) 6 = Other	String
NofParticipants	Number of persons executing the review	Integer
EntryEffort	Effort spent on entry phase	Integer
KickOffEffort	Estimated effort spent on start activities	Integer
PreparationEffort	Estimated preparation effort spent on this review, reading the documents and preparing a list of mistakes.	Float
MeetingEffort	Estimated effort in the review meeting	Integer
ReworkEffort	Estimated Rework effort	Integer
VerificationEffort	Estimated effort for the review of the rework made	Integer
ReviewSize	Number of logical pages or lines of code (LOC) that the review has.	Integer
MajorDefects	The most important defects that must be solved in the review.	Integer
MinorDefects	The least important defects that must be solved in the review.	Integer
Type	Explain the needs of the review PR = Problem report CR = Change Request IR = Implementation Request	String
Severity	Level of severity of the defect being reviewed	String

	S = show stopper/ blocker A = major function affected B = minor function affected C = cosmetic D = all other	
ExternalWorkProduct	0 = internal -1 = external	Integer
State	Outcome of the review process: document is Accepted Cancelled Rejected Rework	String
Unit	Unit Of measurement lines or pages	String
LeadTime	Time it took to review and correct a document	Integer
Moderator	Name of the moderator of the review	String
TargetDateVerification	Date scheduled for the verification of the rework	Date
TargetDateRework	Date scheduled for the rework	Date
TotalEffort	Estimated total effort spent on the review. It is the sum of EntryEffort, KickOfEffort, PreparationEffort, MeetingEffort, ReworkEffor, VerificationEffort	Float
PreparationRate	Average Effort per page spent on preparation	Float
RemovalRate	Average Defects removed per page	Float
AverageSize	Review Size / Number of participants	Float
DefectCost	Total cost of review / major defects solved	Float

SaneID	Outcome of sanity checks	Integer
SaneCD	Outcome of sanity checks	Integer
SaneLT	Outcome of sanity checks	Integer
SaneNP	Outcome of sanity checks	Integer
SaneTE	Outcome of sanity checks	Integer
SanePE	Outcome of sanity checks	Integer
SaneTD	Outcome of sanity checks	Integer
SaneSZ	Outcome of sanity checks	Integer
SanePR	Outcome of sanity checks	Integer
SaneDC	Outcome of sanity checks	Integer
Sane	Outcome of sanity checks	Integer
Recent	Outcome of sanity checks – whether data element is in the expected range	Integer

Table A6. Review Data

8.1.5 Size Data

The data in the snapshots that belong to this category give information about the number size of the code developed during the software development process.

Field name	*Description*	*Values – Data type*
Data Set	Identification of the dataset in the snapshot. Dataset = set of snapshots from the same source at regular intervals. Project – All teams – Type of data	String
History Date	Date the snapshot was taken	Date
Program	Program – architectural components. This is the name of the programme of which the project is a part. Program = collection of Projects	String

System	System name – System = topmost deliverable; often Program name and System name look alike.	String
ProdSubSys	Short indicator for the subsystem/cmpgroup/component. Will be mapped to the proper item in the database.	String
BaseRootPath	Path to the root directory of the base version	String
NewRootPath	Path to the root directory of the new version	String
Unit	Unit of measure to count the code lines	String
Total	Total number of code lines	Integer
Blank	Number of blank lines in the code file	Integer
Comment	Number of comment lines in the code file	Integer
Deleted	Number of lines deleted in the code file	Integer
Equal	Number of unaltered identical lines	Integer
Moved	Number of lines moved in the code file	Integer
Modified	Number of lines modified in the code file	Integer
Added	Number of lines added to the code in the file	Integer
Source	Number of lines in the original source Equal + Moved+ Modified + Added	Integer
Delta	It is equal to the number of lines Modified + added	Integer
File	Name of the file with the code	String
Type	Type of file. Depends on the programming language	String
MatchPath	There is not data available in snapshots Is a file has been moved from one place to another this is the other location	

MatchFile	There is not data available in snapshots Is a file has been given another name, this is the other name	
Patho	The relative path below the baserootpath / new rootpath where the file resides	

Table A7. Size Data

8.2 Company 2

8.2.1 Architecture Data

The architecture data represents the common key among all the snapshots.

Field name	***Description***	***Data type***
DataSet	Identification of the dataset in the snapshot – Dataset = set of snapshots from the same source at regular intervals. Project – All teams – Type of data	String
HistoryDate	Date the snapshot was taken	Date
Product_SubSys	Short indicator for the subsystem/cmpgroup/component. Will be mapped to the proper item in the database.	String
System	System name – System = topmost deliverable; often Program name and System name look alike.	String
SubSystem	Subsystem from which the snapshot was taken – Continuous database the snapshot is taken from. Due to multisite cooperation multiple databases can be included as source. Subsystem = major part of system; system = assembly of 1 or more subsystems.	String

CmpGroup	Name of the components group. Cmpgroup is major part of subsystem; subsystem = assembly of 1 or more cmpgroups	String
Component	Name of the component. Component is major part of cmpgroup; cmpgroup = assembly of 1 or more components	String
External	Indicates that some part of a system is not created by the program / project / team but delivered by or bought from an external party	Integer

Table A8. Architecture Data

8.2.2 Case Data

The information for the documentation of the snapshots in this category was obtained from the document [SIE08]. The data in these snapshots is related to the test cases used in the tests made to the software during the software development process.

Field name	*Description*	*Values – Data type*
DataSet	Identification of the dataset in the snapshot. Dataset = set of snapshots from the same source at regular intervals. Project – All teams – Type of data	String
HistoryDate	Date the snapshot was taken	Date
Program	from configuration –This is the name of the programme of which the project is a part. Program = collection of Projects	String
Project	from configuration – Name of the project Project has 1 or more teams to serve it	String

<table>
<tr><td>Team</td><td>from configuration – Name of the team in charge of the creation of a component

Team has 1 or more projects to serve</td><td>String</td></tr>
<tr><td>ProdSubSys</td><td>from configuration – Short indicator for the subsystem/cmpgroup/component. Will be mapped to the proper item in the database</td><td>String</td></tr>
<tr><td>System</td><td>System name – System = topmost deliverable; often Program name and System name look alike.</td><td>String</td></tr>
<tr><td>CaseId</td><td>from QC – Identification of the test case</td><td>Integer</td></tr>
<tr><td>CaseState</td><td>based on status from QC – This is the same as CrStatus but mapped onto a standard set of states.
<table>
<tr><td>Main State</td><td>Substate</td><td>Meaning</td></tr>
<tr><td>Deferred</td><td>On hold</td><td>Not solved</td></tr>
<tr><td rowspan="4">Open</td><td>Submitted</td><td>Reported</td></tr>
<tr><td>Analysis</td><td>Investigation</td></tr>
<tr><td>Resolution</td><td>Fixing</td></tr>
<tr><td>Evaluation</td><td>Verifying the fix</td></tr>
<tr><td>Closed</td><td>Closed</td><td>Closed after fixing</td></tr>
<tr><td rowspan="3">Rejected</td><td>Duplicate</td><td>Already reported</td></tr>
<tr><td>Nonrepro</td><td>Can't solve, not reproducible</td></tr>
<tr><td>Rejected</td><td>Won't solve, live with it</td></tr>
</table></td><td>String</td></tr>
</table>

		By design	Shouldn't solve, intended behavior.	
CaseStatus	from QC – State is the main state, open, closed, deferred or rejected. Status is the substate of the main state. To keep them separate is easier when handling this data in queries.			String
LinkedTests	from QC – Identification of a related test			Integer
CaseStart	status transition – Date when the execution of the test case started			Date
CaseFinish	status transition – Date when the execution of the test case finished			Date

Table A9. Case Data

8.2.3 Change Data

The information used to document the snapshots that belong to this category was obtained from the document [SIE08]. The data in these snapshots refer to the changes occurred in components during the software development process.

Field name	*Description*	*Values – Data type*
DataSet	Identification of the dataset in the snapshot	String
HistoryDate	Date the snapshot was taken	Date
Program	Program – architectural components. This is the name of the programme of which the project is a part. Program = collection of Projects.	String
Project	from configuration – Name of the project Project has 1 or more teams to serve it	String

<table>
<tr><td>Team</td><td colspan="3">from configuration – Name of the team in charge of the creation of a component

Team has 1 or more projects to serve</td><td>String</td></tr>
<tr><td>System</td><td colspan="3">System name – System = topmost deliverable; often Program name and System name look alike</td><td>String</td></tr>
<tr><td>SubSystem</td><td colspan="3">Subsystem from which it was taken the snapshot - Continuous database the snapshot is taken from. Due to multisite cooperation multiple databases</td><td>String</td></tr>
<tr><td>CmpGroup</td><td colspan="3">from configuration – Name of the group where the component for which the change request was made belongs to.

Cmpgroup is major part of subsystem; subsystem = assembly of 1 or more cmpgroups</td><td>String</td></tr>
<tr><td>Component</td><td colspan="3">from configuration – Name of the component for which the change request was made.

Component is major part of cmpgroup; cmpgroup = assembly of 1 or more components</td><td>String</td></tr>
<tr><td>ChangeIdent</td><td colspan="3">from RIC – Identification of the change request</td><td>String</td></tr>
<tr><td rowspan="7">ChangeState</td><td colspan="3">from RIC</td><td rowspan="7">String</td></tr>
<tr><td>Main State</td><td>Substate</td><td>Meaning</td></tr>
<tr><td>Deferred</td><td>On hold</td><td>Not solved</td></tr>
<tr><td rowspan="4">Open</td><td>Submitted</td><td>Reported</td></tr>
<tr><td>Analysis</td><td>Investigation</td></tr>
<tr><td>Resolution</td><td>Fixing</td></tr>
<tr><td>Evaluation</td><td>Verifying the fix</td></tr>
</table>

	Closed	Closed	Closed after fixing	
	Rejected	Duplicate	Already reported	
		Nonrepro	Can't solve, not reproducible	
		Rejected	Won't solve, live with it	
		By design	Shouldn't solve, intended behavior.	
ChangeStatus	from RIC – State is the main state, open, closed, deferred or rejected. Status is the substate of the main state. To keep them separate is easier when handling this data in queries.			String
ChangeApproved	from RIC – Was the change request approved? Yes/NO			String
Category	from RIC – Classification of the change request: • Budget • Planning • Scope			String
RootCause	from RIC – Cause of the change request. Some standard values are: • External Business Impact • Incomplete Business Impact Analysis • Incomplete Functional Impact Analysis • Incomplete Technical Impact Analysis			String

	• New Business Requirements • Over-estimation of effort • Requirements dropped	
Priority	Ordering to address change requests in the project: 1 = Low 2 = Medium 3 = High 4 = Top	String
Detected	from RIC – Phase of the project when the need of a change was detected. Some of the standard values are: • Preparation system validation • System Validation • Launch • Post Launch • Validation • Detailed Technical Design • Preparation Operational Acceptance • Coding • Business Case • Operational Acceptance • Participant Acceptance • System Validation	String
TargetDate	from RIC – Date when the change request should be solved	Date
EstCostsEUR	from RIC – Estimated cost in euro of solving the change request	Float

EstCostsMD	from RIC – Estimated cost in mandays in additional budget for solving the change request	Float
EstContingencyMD	from RIC – Estimated cost in mandays from contingency budget for solving the change request	?
CreateDate	status transition date from RIC – Date the change request was created	Date
ApprovalDate	status transition date from RIC – Date the change request was approved	Date
LastModified	status transition date from RIC – Last time the change request status was modified	Date
Draft	status transition date from RIC – Date when the draft of the change request was created	Date
Raised	status transition date from RIC – Date when the change request was raised	Date
Assigned	status transition date from RIC – Date the change request was assigned to be analyzed	Date
Completed	status transition date from RIC – Date the change request was completed	Date
Approval	status transition date from RIC – Date the change request started the procedure for approval	Date
Approved	status transition date from RIC – Date the change request was approved	Date
Closed	status transition date from RIC – Date the change request was closed	Date
Cancelled	status transition date from RIC – Date the change request was cancelled	Date

Rejected	status transition date from RIC – Date the change request was rejected	Date

Table A10. Change Data

8.2.4 Defect Data

The information used to document the snapshots that belong to this category was obtained from the document [SIE08]. The data in these snapshots is related with test requirements (usually TDS's), test cases, tests steps, defects and bugs from a QC database.

Field name	*Description*	*Values – Data type*
DataSet	Identification of the dataset in the snapshot. Dataset = set of snapshots from the same source at regular intervals. Project – All teams – Type of data	String
HistoryDate	Date the snapshot was taken	String
Program	Program – architectural components. This is the name of the programme of which the project is a part. Program = collection of Projects	String
Project	from configuration – Name of the project Project has 1 or more teams to serve it	String
Team	from configuration – Name of the team in charge of the resolution of the defect. Team has 1 or more projects to serve	String
ProdSubSys	from configuration – Short indicator for the subsystem/cmpgroup/component. Will be mapped to the proper item in the database	String
System	from configuration – Name of the system System = topmost deliverable; often Program name and System name look alike	String
Version	from QC – Version of the system or subsystem etc.	String

<table>
<tr><td>DefectId</td><td>from QC – Id of the defect</td><td>Integer</td></tr>
<tr><td>DefectType</td><td>from QC – Description of the defect. This data is not always available.

The data contain real defects (PR), changes to the requirements (CR) and impact of normal work (IR). The classification is not always known immediately.</td><td>String</td></tr>
<tr><td>DefectState</td><td>based on status from QC – This is the same as CrStatus but mapped onto a standard set of states.
<table>
<tr><td>Main State</td><td>Substate</td><td>Meaning</td></tr>
<tr><td>Deferred</td><td>On hold</td><td>Not solved</td></tr>
<tr><td rowspan="4">Open</td><td>Submitted</td><td>Reported</td></tr>
<tr><td>Analysis</td><td>Investigation</td></tr>
<tr><td>Resolution</td><td>Fixing</td></tr>
<tr><td>Evaluation</td><td>Verifying the fix</td></tr>
<tr><td>Closed</td><td>Closed</td><td>Closed after fixing</td></tr>
<tr><td rowspan="4">Rejected</td><td>Duplicate</td><td>Already reported</td></tr>
<tr><td>Nonrepro</td><td>Can’t solve, not reproducible</td></tr>
<tr><td>Rejected</td><td>Won’t solve, live with it</td></tr>
<tr><td>By design</td><td>Shouldn’t solve, intended behavior.</td></tr>
</table></td><td>String</td></tr>
</table>

DefectStatus	from QC – State is the main state, open, closed, deferred or rejected. Status is the substate of the main state. To keep them separate is easier when handling this data in queries.	String
DefectEstimate	from QC – Estimated cost to repair the defect (PR) or to implement the change (CR) or the task (IR). In many cases there is a symbol "?", instead of data.	Integer
DefectCost	from QC – Actual cost to repair the defect (PR) or to implement the change (CR) or the task (IR). In many cases there is a symbol "?", instead of data.	Integer
Severity	from QC – S = show stopper/ blocker A = major function affected B = minor function affected C = cosmetic D = all other How much the defect/change affects the performance or behavior of system	String
Priority	from QC – Priority given to the defect for its treatment. [From defects project] Ordering to address things in the project: 1 = Low 2 = Medium 3 = High 4 = Top How soon we want the issue to be solved	String
Injected	from QC – phase of the project when the defect was injected	String

Detected	from QC – phase of the project when the defect was detected	String
DefectStart	status transition date from QC – Creation of the defect	Date
DefectRestart	Not in all files - status transition date from QC – Date when the defect was restarted.	Date
DefectOpen	Not in all files - status transition date from QC – Date when the defect was opened	Date
DefectReopen	Not in all files - status transition date from QC – Date when the defect was reopened	Date
DefectAnalysis	status transition date from QC - State set to in-analysis [sub of analysis]. Date when the analysis started.	Date
DefectResolution	Status transition date from QC – State set to in-resolution [sub of resolution]. Date when the defect entered to resolution	Date
DefectEvaluation	status transition date from QC – state set to in-evaluation [sub of evaluation]. Date when the defect entered to the evaluation process.	Date
DefectFinish	status transition date from QC – state set to closed or to rejected	Date

Table A11. Defect Data

8.2.5 Issue Data

The information to document the snapshots that belong to this category was obtained from the document [SIE08]. The data stored in these snapshots is related with the issues arised during the software development process.

Field name	*Description*	*Values – Data type*
DataSet	Identification of the dataset in the snapshot – Dataset = set of snapshots from the same source at regular intervals. Project – All teams – Type of data	String
HistoryDate	Date the snapshot was taken	Date
Program	Program – architectural components. This is the name of the programme of which the project is a part. Program = collection of Projects	String
Project	from configuration – Name of the project Project has 1 or more teams to serve it	String
Team	from configuration – Name of the team in charge of managing the issue Team has 1 or more projects to serve	String
System	System name – System = topmost deliverable; often Program name and System name look alike.	String
SubSystem	Subsystem from which the snapshot was taken – Continuous database the snapshot is taken from. Due to multisite cooperation multiple databases can be included as source. Subsystem = major part of system; system = assembly of 1 or more subsystems.	String
CmpGroup	from configuration – Name of the component group where the component for which the issue was identified belongs to. Cmpgroup is major part of subsystem; subsystem = assembly of 1 or more cmpgroups.	String
Component	from configuration – Name of the	String

<table>
<tr><td></td><td>component for which the issue was detected.

Component is major part of cmpgroup; cmpgroup = assembly of 1 or more components</td><td></td></tr>
<tr><td>IssueIdent</td><td>from RIC – Identification of the issue</td><td>Integer</td></tr>
<tr><td>IssueState</td><td>from RIC
<table>
<tr><td>Main State</td><td>Substate</td><td>Meaning</td></tr>
<tr><td>Deferred</td><td>On hold</td><td>Not solved</td></tr>
<tr><td rowspan="4">Open</td><td>Submitted</td><td>Reported</td></tr>
<tr><td>Analysis</td><td>Investigation</td></tr>
<tr><td>Resolution</td><td>Fixing</td></tr>
<tr><td>Evaluation</td><td>Verifying the fix</td></tr>
<tr><td>Closed</td><td>Closed</td><td>Closed after fixing</td></tr>
<tr><td rowspan="4">Rejected</td><td>Duplicate</td><td>Already reported</td></tr>
<tr><td>Nonrepro</td><td>Can’t solve, not reproducible</td></tr>
<tr><td>Rejected</td><td>Won’t solve, live with it</td></tr>
<tr><td>By design</td><td>Shouldn’t solve, intended behavior.</td></tr>
</table></td><td>String</td></tr>
<tr><td>IssueStatus</td><td>from RIC – from QC –. Status is the substate of the main state. To keep them separate is easier when handling this data in queries.</td><td>String</td></tr>
</table>

Category	from RIC – Category of the issue – Some default values for this field are: • Change and configuration management • Delivery • Planning • Resource equation • Scope • Vision and Strategy	String
Criticality	from RIC- Criticality of the issue. Default values: • Major • Minor • Moderate • Significant	String
PhaseDetected	from RIC – Phase of the project when the issue was detected	String
CreateDate	status transition date from RIC – Date when the issue was created	Date
LastModified	status transition date from RIC – Last time the issue was modified	Date
Draft	status transition date from RIC – Date the draft of the issue was created	Date
Raised	status transition date from RIC – Date the issue was raised	Date
Assigned	status transition date from RIC – Date the issue was assigned to be solved	Date
Completed	status transition date from RIC – Date the issue was completely solved	Date
Closed	status transition date from RIC – Date the issue was closed	Date

Cancelled	status transition date from RIC – Date the issue was cancelled	Date

Table A12. Issue Data

8.2.6 Project Data - "Effort Data"

The data stored in the snapshots in the category Effort Data is related with the time, effort, budget spent in projects for software development, and it has been retrieved from different databases. The documentation of the snapshots is based on the documents [SIE08], [SIE02-1] and [SIE02-2].

Field name	*Description*	*Values – Data type*
DataSet	Identification of the dataset in the snapshot	String
HistoryDate	Date the snapshot was taken	Date
Unique_Id	from PSN – Unique ID of the task within the MsProject file, maintained over time (i.e. the same snapshots unless task is deleted and replaced)	Integer
Outline_Number	from PSN – The structural ordering of the task in the file, 1comes before 2. 1.1 is the first child, etc.	Float
Milestone	Is the task a Milestone? Yes/No. YES if task is a milestone.	String
Summary	Is the task a summary task? Yes/No. A summary task is a parent and as such, it is the sum of all its child tasks. YES is the task is a parent.	String
Name	from PSN – The description of the task in MsProject.	String
Flag10	Indicates task completion percentage (either 0 / 100). If it is 0 the task is ongoing. Yes/No (100/0) For milestones • set to yes when milestone has been	Integer

	successfully passed For non-summary tasks • set to yes when a task is complete; the task will then be included in the work performed during Earned Value calculations		
Initial_Start	Is the baseline start when first baselines; later baselining may cause baseline start to differ. Not in all snapshots		Date
Initial_Finish	Is the baseline finish when first baselines; later baselining may cause baseline finish to differ. Not in all snapshots		Date
Initial_Work	Is the baseline work when first baselines; later baselining may cause baseline work to differ. Not in all snapshots		Float
Baseline_Start	Start from Baseline 1 in PSN when available	The start, finish and scheduled work are the **current** plan.	Date
Baseline_Finish	Finish from Baseline 1 in PSN when available		Date
Baseline_Work	Work from Baseline 1 in PSN	When the plan is approved, a **baseline** copy is saved of the scheduled start, finish and work (hours).	Float
Start_Date	Scheduled start from PSN		Date
Finish_Date	Scheduled finish from PSN		Date
Scheduled_Work	Scheduled work [Actuals+ETC] from PCteam and PSN	The **actual** start, finish and work reflect progress. When the actual start is set the scheduled start is set to the actual. For ongoing tasks (Flag 10=0) the finish date is set to the snapshot date.	Float
Actual_Start	from PCteam - What is PCteam		Date
Actual_Finish	from PCteam. Only recorded when the task is 100% complete. Included only for		Date

	completeness sake.		
Actual_Work	from PCteam	The actual work is always the effort spent between start and finish.	Float
Text4	The type of activity involved, see explanation. PDSL Activity Type* Project Start System Proposed System Defined Code Complete System Complete System Accepted Project End Project Start Project Implementation Approval SW Components Specified SW Components Available System Validated System Release Approval Project Complete Kick Off Concept Start Product Range Start Design Release Commercial Release Mass Production Release PDSL Activity Type: • Requirements • Design • Coding • Testing		String

	• Management • Support • Problems • Other OR PDSL Milestone: • PS • SP • SD • CC • SC • SA • PE OR OSRP Milestone: • PS (=PS) • PIA (=SP) • SWCS (=SD) • SWCA (=SC) • SV • SRA (=SA) • PC (=PE) OR SPEED Milestone: • KO (=PS) • CS (=SP) • PRS (=SD) • DR (=SC) • CR (=SA) • MPR (~PE) From * For milestones: • **Teammilestone** • **projectmilestone** For non-summary tasks (see Quality Manual) • **REQ** for Requirements/ Specification • **DES** for Architecture / Design • **ARCSUP** for Architecture Support • **IMP** for Implementation / Coding • **ITS** for Integration Test Specification • **ITI** for Integration Test Implementation • **ITE** for Integration Test Execution	

	• **QTS** for Qualification Test Specification • **QTI** for Qualification Test Implementation • **QTE** for Qualification Test Execution • **MGT** for Management/Planning/Tracking/Meeting • **SUP** for CM/QA/Training • **REW** for Problem Solving **Other** for non-project activities	
Text5	Deliverable. If non-empty indicates that the task has a deliverable with it.	String
Text15	This field is only used in some snapshots. This is the name of the programme of which the project is a part. Program = collection of Projects	
Text16	It is the name of the project. Project has 1 or more teams to serve it For all tasks	String
Text17	Is it the name of the team For all tasks Team has 1 or more projects to serve	String
Text25	System For all tasks. System Name, e.g. C-STEP System has subsystems	String
Text26	Subsystem For tasks uniquely related to a single subsystem • Subsystem Name, e.g. SV for Service Layer Otherwise • (same as) System Name Subsystem has cmpgroups	String
Text27	Group	String

	For tasks uniquely related to a single component group • Component Group Name, e.g. SV_HDMAN Otherwise • (same as) Subsystem Name Cmpgroup has components	
Text28	Component For tasks uniquely related to a single component • Component Name Otherwise • (same as) Component Group Name Note: for storage projects related to DVD+RW the Component field is always set to the same value as the Group field because defects are reported only to the level of Component Groups	String
Text29	Release - There is not data available in some files.	String
Resource_Names	from PSN – Names of the people performing the task	String
ETC	from PSN – estimate to complete in mandays	Float
Stage	from PSN – Stage of the project – initiation - execution	String
Phase	from PSN – Same as activity type	String
Skill	from PSN – type of resource needed for the task	String
TaskNumber	from PSN – Identifier	Integer

Predecessors	from PSN – Tasks that this task depends on	Integer
Successors	from PSN – Tasks that depend on this task	Integer
CriticalPath	from PSN – See the literature on critical path and critical chain in a network planning	String
Psn_Start	from PSN	Date
Psn_Finish	from PSN	Date
Psn_Work	from PSN	Float
ParentID	from PSN – Identifier of the task higher in the tree	String
ProjectID	from PSN – Identifier of the account where the cists are booked	String
PIC	from PSN – Identifier within the account to book the costs	String
SubProjName	Not in all files - from PSN for use in Crosslinks – some schedules have sub-schedules that are kept in separate files – this is the name of such a file	String
SubProjTaskId	Not in all files - from PSN for use in Crosslinks - This identified refers to a specific task in the subschedule	Integer

Table A13. Project Data

8.2.7 Requirements Data

The information used for the documentation of these snapshots was obtained from the document [SIE08]. The data stored in these snapshots is related to the requirements for developing a software application.

Field name	*Description*	*Values – Data type*
DataSet	Identification of the dataset in the snapshot. Dataset = set of snapshots from the same source at regular intervals. Project – All teams – Type of data	String

HistoryDate	Date the snapshot was taken	Date
Program	Program – architectural components. This is the name of the programme of which the project is a part. Program = collection of Projects	String
Project	from configuration – Name of the project Project has 1 or more teams to serve it	String
Team	Name of the team in charge of the requirements.	String
ProdSubSys	from configuration – Short indicator for the subsystem/cmpgroup/component. Will be mapped to the proper item in the database	String
System	System name – System = topmost deliverable; often Program name and System name look alike.	String
ReqId	Identification of the requirement	Integer
ReqTraceBack	Not in all files - from QC – Reference to a higher level requirement in another database	Integer
ReqChildren	from QC – Reference to more detailed requirements below this requirement	Integer
ReqParent	from QC – Reference to the higher level requirement above the current one	Integer
ReqOrder	from QC – Sequence number used to determine the order of the requirements when printing / reading	Integer
ReqReview	from QC – Status of review of the requirement	String
Priority	from QC – Priority of the requirement Ordering to address things in the project: 1 = Low	String

<table>
<tr><td></td><td colspan="3">2 = Medium
3 = High
4 = Top</td><td></td></tr>
<tr><td>ReqType</td><td colspan="3">from QC – Type of requirement – Some default values are:
• BTR
• BTRH
• TDS
• TDSH
• DAF
• Folder</td><td>String</td></tr>
<tr><td rowspan="12">ReqState</td><td colspan="3">from QC</td><td rowspan="12">String</td></tr>
<tr><td>Main State</td><td>Substate</td><td>Meaning</td></tr>
<tr><td>Deferred</td><td>On hold</td><td>Not solved</td></tr>
<tr><td rowspan="4">Open</td><td>Submitted</td><td>Reported</td></tr>
<tr><td>Analysis</td><td>Investigation</td></tr>
<tr><td>Resolution</td><td>Fixing</td></tr>
<tr><td>Evaluation</td><td>Verifying the fix</td></tr>
<tr><td>Closed</td><td>Closed</td><td>Closed after fixing</td></tr>
<tr><td rowspan="4">Rejected</td><td>Duplicate</td><td>Already reported</td></tr>
<tr><td>Nonrepro</td><td>Can't solve, not reproducible</td></tr>
<tr><td>Rejected</td><td>Won't solve, live with it</td></tr>
<tr><td>By design</td><td>Shouldn't solve, intended</td></tr>
</table>

	behavior.	
ReqStatus	from QC – State is the main state, open, closed, deferred or rejected. Status is the substate of the main state. To keep them separate is easier when handling this data in queries.	String
LinkedTests	from QC – Identification of the test case with which the requirement will be tested	Integer
ReqStart	status transition date from QC – Creation of the requirement	Date
ReqDesign	status transition date from QC – Date when the requirement started design phase	Date
ReqPreparation	status transition date from QC – Date when the requirement started preparation for being implemented	Date
ReqExecution	status transition date from QC – Date when the test was executed	Date
ReqFailed	status transition date from QC – Date when the test failed complying the requirement	Date
ReqPassed	status transition date from QC – Date when the test passed	Date

Table A14. Requirements Data

8.2.8 Risk data

The information used to document this field was obtained from the document t[SIE08]. The data stored in these snapshots refers to the risks that appear and must be managed during the software development process.

Field name	*Description*	*Values – Data type*
DataSet	Identification of the dataset in the snapshot. Dataset = set of snapshots from the same source at regular intervals. Project – All teams – Type of data	String

<table>
<tr><td>HistoryDate</td><td>Date the snapshot was taken</td><td>Date</td></tr>
<tr><td>Program</td><td>Program – architectural components. This is the name of the programme of which the project is a part.
Program = collection of Projects</td><td>String</td></tr>
<tr><td>Project</td><td>from configuration – Name of the project
Project has 1 or more teams to serve it</td><td>String</td></tr>
<tr><td>Team</td><td>from configuration – Team in charge of the management of the risks
Team has 1 or more projects to serve</td><td>String</td></tr>
<tr><td>System</td><td>from configuration – Name of the system. System = topmost deliverable; often Program name and System name look alike.</td><td>String</td></tr>
<tr><td>SubSystem</td><td>from configuration - Subsystem from which it was taken the snapshot – Continuous database the snapshot is taken from. Due to multisite cooperation multiple databases can be included as source.
Subsystem = major part of system; system = assembly of 1 or more subsystems</td><td>String</td></tr>
<tr><td>CmpGroup</td><td>from configuration – Name of the components group
Cmpgroup is major part of subsystem; subsystem = assembly of 1 or more cmpgroups</td><td>String</td></tr>
<tr><td>Component</td><td>from configuration – Name of the component for which risks are managed. Name of the component. Component is major part of cmpgroup;</td><td>String</td></tr>
<tr><td>RiskId</td><td>from RIC – Identification of the risk</td><td>String</td></tr>
<tr><td>RiskState</td><td>from RIC<table><tr><td>Main State</td><td>Substate</td><td>Meaning</td></tr></table></td><td>String</td></tr>
</table>

	Deferred	On hold	Not solved	
	Open	Submitted	Reported	
		Analysis	Investigation	
		Resolution	Fixing	
		Evaluation	Verifying the fix	
	Closed	Closed	Closed after fixing	
	Rejected	Duplicate	Already reported	
		Nonrepro	Can't solve, not reproducible	
		Rejected	Won't solve, live with it	
		By design	Shouldn't solve, intended behavior.	
RiskStatus	from QC – State is the main state, open, closed, deferred or rejected. Status is the substate of the main state. To keep them separate is easier when handling this data in queries.			String
Category	from RIC – Category of the risk – Some standard values are: • Budget • Delivery			String

<table>
<tr><td></td><td>Performance
Planning
Resource equation</td><td></td></tr>
<tr><td>Probability</td><td>from RIC – Probability that the risk will become true – Default values:
Likely
Unlikely
Possible
Frequent</td><td>String</td></tr>
<tr><td>Impact</td><td>from RIC – Impact on the project in case the risk become true. – Default values:
Significant
Moderate</td><td>String</td></tr>
<tr><td>Exposure</td><td>from RIC – Level of exposure to the risk. Default values:
High
Low
Minor</td><td>String</td></tr>
<tr><td>PhaseDetected</td><td>from RIC – Phase in which the risk was detected.
Some default values are:
Preparation system validation
Coding
Business case
Coding</td><td>String</td></tr>
<tr><td>PhaseImpacted</td><td>from RIC – Phase that would be affected in case the risk become true. Some default values are:
Launch</td><td>String</td></tr>
</table>

	• System validation • Operational acceptance • System validation • Participant acceptance • Operational ready	
CreateDate	status transition date from RIC – Date the risk was created	Date
LastModified	status transition date from RIC – Last time the risk statement was modified	Date
Draft	status transition date from RIC – Date the draft of the risk document was created	Date
Raised	status transition date from RIC – Date the risk was raised	Date
Assigned	status transition date from RIC – Date the risk was assigned to be analyzed.	Date
Completed	status transition date from RIC – Date the risk analysis was completed	Date
Closed	status transition date from RIC – Date the risk was assigned status closed	Date
Cancelled	status transition date from RIC – Date the risk was cancelled	Date
Target Date	Not in all files - From RIC – Date the risk is to be mitigated	Date

Table A15. Risk Data

8.2.9 Test data

The information used to document the snapshots that belong to this category was obtained in the document [SIE08]. These snapshots contain data related to the tests performed in the software development process.

Field name	*Description*	*Values – Data type*
DataSet	Identification of the dataset in the snapshot. Dataset = set of snapshots from the same source at regular intervals. Project – All teams – Type of data	String
HistoryDate	Date the snapshot was taken	Date
Program	Program – architectural components. This is the name of the programme of which the project is a part. Program = collection of Projects	String
Project	from configuration – Name of the project Project has 1 or more teams to serve it	String
Team	from configuration – Name of the team that will perform a test. Team has 1 or more projects to serve	String
ProdSubSys	from configuration – Short indicator for the subsystem/cmpgroup/component. Will be mapped to the proper item in the database	String
System	System name – System = topmost deliverable; often Program name and System name look alike.	String
TestId	from QC – Identification of the test	Integer
TestType	from QC – Type of the test – Default values: • MANUAL • VAPI-XP-TEST • ALT-SCENARIO • LR-SCENARIO	String
TestStatus	from QC – State is the main state, open, closed, deferred or rejected. Status is the substate of the main state. To keep them separate is easier when handling this data in	String

<table>
<tr><td></td><td colspan="3">queries.</td><td></td></tr>
<tr><td rowspan="12">TestState</td><td colspan="3">based on status from QC</td><td rowspan="12">String</td></tr>
<tr><td>Main State</td><td>Substate</td><td>Meaning</td></tr>
<tr><td>Deferred</td><td>On hold</td><td>Not solved</td></tr>
<tr><td rowspan="4">Open</td><td>Submitted</td><td>Reported</td></tr>
<tr><td>Analysis</td><td>Investigation</td></tr>
<tr><td>Resolution</td><td>Fixing</td></tr>
<tr><td>Evaluation</td><td>Verifying the fix</td></tr>
<tr><td>Closed</td><td>Closed</td><td>Closed after fixing</td></tr>
<tr><td rowspan="4">Rejected</td><td>Duplicate</td><td>Already reported</td></tr>
<tr><td>Nonrepro</td><td>Can’t solve, not reproducible</td></tr>
<tr><td>Rejected</td><td>Won’t solve, live with it</td></tr>
<tr><td>By design</td><td>Shouldn’t solve, intended behavior.</td></tr>
<tr><td>TestReview</td><td colspan="3">Status of the revision of the test</td><td>String</td></tr>
<tr><td>TestExec</td><td colspan="3">Has the test been executed</td><td>String</td></tr>
<tr><td>LinkedSteps</td><td colspan="3">Reference to the steps that make up the test case</td><td>Integer</td></tr>
<tr><td>FailedRuns</td><td colspan="3">Number of times the test has been executed and failed</td><td>Integer</td></tr>
<tr><td>PassedRuns</td><td colspan="3">Number of times the test has been executed</td><td>Integer</td></tr>
</table>

	and passed	
TestStart	Date the test was created	Date
TestPreparation	Date the test became prepared	Date
TestExecution	Date the test was executed	Date
TestFailed	Date the test failed	Date
TestPassed	Date the test passed	Date

Table A16. Test Data

Appendix B

9 Appendix B: Quality measurements

In this appendix the attributes that were defined to be assessed are listed in section 1, and then, in section 2 the results of the measurements performed are presented.

9.1 Attributes and metrics

In the following subsections for every company there are tables that enumerate the attributes assessed and the metrics used for the corresponding measurements.

9.1.1 Company 1

Architecture

Table B1 presents the attributes and metrics for the category Architecture.

Name of the field	*Metric to assess*
Product_SubSys, System SubSystem, CmpGroup Component	Completeness

Table B1. Attributes and metrics of Architecture

Defect

Subcategory 1

Table B2 presents the attributes and metrics for the subcategory 1 of the category Defect.

Name of the field	*Metric to assess*
Project, Team, ProdSubSys, System	Completeness
DefectId	Accuracy: Duplicated values
DefectEstimate, DefectCost Injected, Detected, DefectAnalysis DefectResolution, DefectEvaluation DefectFinish	Completeness: when the fields have values like "?", it is considered as null. It is a case of incompleteness where the value exists but is unknown.

DefectState, DefectStatus Severity, Priority	Accuracy: Syntactic errors There are reference domains to be used.
DefectStart, DefectAnalysis DefectResolution, DefectEvaluation DefectFinish	Consistency: Use of some edit rules. DefectStart < DefectAnalysis DefectAnalysis < DefectResolution DefectResolution < DefectEvaluation DefectEvaluation < DefectFinish

Table B2. Attributes and metrics Defect 1

Subcategory 2

Table B3 presents the attributes and metrics for the subcategory 2 of the category Defect.

Name of the field	***Metric to assess***
Program, Project, Team Prodsubsys, System	Completeness
DefectId	Accuracy: Duplicated values
Defect Type , DefectState DefectStatus, Severity, Priority	Accuracy: Syntactic errors There are reference domains to be used
DefectCost, Injected, Detected DefectAnalysis, DefectResolution DefectEvaluation, DefectFinish	Completeness: when the fields have values like "?", it is considered as null. It is a case of incompleteness where the value exists but is unknown.
DefectStart DefectAnalysis DefectResolution DefectEvaluation DefectFinish	Consistency: Use of some edit rules DefectStart < DefectAnalysis DefectAnalysis < DefectResolution DefectResolution < DefectEvaluation DefectEvaluation < DefectFinish

Table B3. Attributes and metrics Defect 2

Subcategory 3

Table B4 presents the attributes and metrics for the subcategory 3 of the category Defect.

Name of the field	*Metric to assess*
System, SubSystem, Product_subsys Program	Completeness
Problem_number	Accuracy: Duplicated values
Priority, Severity, Problem_type Request_type, Crstatus	Accuracy: Syntactic errors There are reference domains to be used
Defect_type, Scope, Priority Problem_type, Caused_during Discovered_during, Act_total_eff Submitted_time, Analysed_time Resolved_time	Incompleteness
Act_total_eff	Consistency: Outliers such as negative numbers.
Create_time Submitted_time Analysed_time Resolved_time Evaluated_time	Consistency: Use of some edit rules Create_time < Submitted_time Submitted_time < Analysed_time Analysed_time < Resolved_time Resolved_time < Evaluated_time

Table B4. Attributes and metrics Defect 2

Project

Table B5 presents the attributes and metrics for the category Project.

Name of the field	*Metric to assess*
Text4 (type of activity) Text15 (program) Text16 (project) Text17 (team)	Completeness

Text25 (System) Text26 (subsystem) Text27 (component group) Text28 (component)	
Unique_ID	Accuracy: Duplicated values
Flag 10	Accuracy: Syntactic errors There is a reference domain
Baseline_Work, Scheduled_Work Actual_Work	Outliers such as negative numbers
Baseline_Start Baseline_Finish	Consistency: Use edit rules Baseline_Start < Baseline_Finish
Start_Date, Finish_Date	Start_Date < Finish_Date
Actual_Start, Actual_Finish	Actual_Start < Actual_Finish

Table B5. Attributes and metrics Project

Review

Table B6 presents the attributes and metrics for the category Review.

Name of the field	*Metric to assess*
Project, Team, System, ProdSubsys	Completeness
InitiationDate, ClosureDate	Consistency: Use of some semantic rules InitiationDate < ClosureDate
DefectId	Accuracy: Duplicated values
ActivityType, Type, Severity State, Unit	Accuracy: Syntactic errors There are reference domains to be used
NofParticipants, PreparationEffort MeetingEffort, ReworkEffort VerificationEffort, ReviewSize	Consistency: Outliers such as negative numbers

MajorDefects, MinorDefects TotalEffort, PreparationRate RemovalRate, AverageSize DefectCost, LeadTime	

Table B6. Attributes and metrics Review

Size

Table B7 presents the attributes and metrics for the category Size.

Name of the field	*Metric to assess*
Program, System, ProdSubSys	Completeness
Unit	Accuracy: Syntactic errors. There is a reference domain
Total, Blank, Comment, Deleted Equal, Moved, Modified, Added	Consistency: Outliers such as negative numbers.
Source Delta	Accuracy: Use of edit rules Source = Equal + moved + modified + Added Delta = Modified + added

Table B7. Attributes and metrics Size

9.1.2 Company 2

Architecture

Table B8 presents the attributes and metrics for the category Architecture.

Name of the field	*Metric to assess*
Product_SubSys, System, SubSystem CmpGroup, Component	Completeness

Table B8. Attributes and metrics Architecture

Case

Table B9 presents the attributes and metrics for the category Case.

Name of the field	*Metric to assess*
Program, Project, Team, ProdSubSys System, CaseFinish	Completeness
CaseId	Accuracy: Duplicated values
CaseState CaseStatus	Accuracy: Syntactic errors There are reference domains to be used
CaseStart CaseFinish	Consistency: Use of edit rule CaseStart < CaseFinish

Table B9. Attributes and metrics Case

Change

Table B10 presents the attributes and metrics for the category Change.

Name of the field	*Metric to assess*
Program, Project, Team, System Subsystem, CmpGroup, Component	Completeness
ChangeState, ChangeStatus Category, Priority	Accuracy: Syntactic errors There are reference domains to be used
EstCostsEUR, EstCostsMD EstContingencyMD	Consistency: outliers such as negative values.
CreateDate, ApprovalDate LastModified, Draft, Raised, Assigned Completed, Approval, Approved Closed, Cancelled, Rejected	Incompleteness

Table B10. Attributes and metrics Change

Defect

Table B11 presents the attributes and metrics for the category Defect.

Name of the field	*Metric to assess*
Program, Project, Team, ProdSubsys	Completeness

System, DefectState, DefectOpen DefectAnalysis, DefectResolution DefectEvaluation	
DefectId	Accuracy: Duplicated values
DefectType, DefectState, DefectStatus Priority, Severity	Accuracy: Syntactic errors There are reference domains to be used
DefectStart DefectOpen DefectAnalysis DefectResolution DefectEvaluation DefectFinish	Consistency: Use some edit rules DefectStart < DefectOpen DefectOpen < DefectAnalysis DefectAnalysis < DefectResolution DefectResolution < DefectEvaluation DefectEvaluation < DefectFinish

Table B11. Attributes and metrics Change

Issue

Table B12 presents the attributes and metrics for the category Issue.

Name of the field	***Metric to assess***
Program, Project, Team, System SubSystem, CmpGroup, Component Completed	Completeness
IssueState, IssueStatus, Criticality	Accuracy: Syntactic errors There are reference domains to be used
CreateDate, Completed Closed	CreateDate < Completed && CreateDate < Closed

Table B12. Attributes and metrics Change

Project

Table B13 presents the attributes and metrics for the category Project.

Name of the field	*Metric to assess*
Text15 (Program) Text16 (Project) Text17 (Team) Text25 (System) Text26 (Subsystem) Text27 (Component group) Text28 (Component)	Completeness
Flag10	Accuracy: Syntactic errors There is a reference domain to be used
Initial_work, Baseline_Work Scheduled_Work, Actual_Work Psn_Work, ETC	Consistency: Outliers (negative numbers)
Initial_Start Initial_finish	Consistency: Use some edit rules. Initial_Start < Initial_finish.
Baseline_Start Baseline_Finish	Consistency: Use some edit rules. Baseline_Start < Baseline_Finish.
Start_Date Finish_Date	Consistency: Use some edit rules. Start_Date < Finish_Date.
Actual_Start Actual_Finish	Consistency: Use some edit rules. Actual_Start < Actual_Finish.
Psn_Start Psn_Finish	Consistency: Use of edit rule Psn_Start < Psn_Finish

Table B13. Attributes and metrics Project

Requirements

Table B14 presents the attributes and metrics for the category Requirements.

Name of the field	*Metric to assess*
Program, Project, Team, ProdSubSys System, Priority, ReqDesign ReqPreparation, ReqExecution ReqFailed, ReqPassed	Completeness
ReqId	Accuracy: Duplicated values
ReqType	Accuracy: Syntactic errors There is a reference domain to be used
ReqStart ReqDesign ReqPreparation ReqExecution ReqFailed ReqPassed	Consistency: Use some semantic rules ReqStart < ReqDesign ReqDesign < ReqPreparation ReqPreparation < ReqExecution ReqExecution < ReqFailed ReqFailed < ReqPassed

Table B14. Attributes and metrics Project

Risk

Table B15 presents the attributes and metrics for the category Risk.

Name of the field	*Metric to assess*
Program, Project, Team, System SubSystem, CmpGroup, Component Raised, Assigned, Completed TargetDate	Completeness
RiskId	Accuracy: Duplicated values
RiskState RiskStatus	Accuracy: Syntactic errors There are reference domains to be used
CreateDate, LastModified Draft, Raised, Assigned, Completed	Completeness Consistency: Use some edit rules.

Closed, Cancelled	CreateDate < Assigned CreateDate < Completed CreateDate < Closed CreateDate < Cancelled

Table B15. Attributes and metrics Risk

Test

Table B16 presents the attributes and metrics for the category Test.

Name of the field	***Metric to assess***
Program, Project, Team, ProdSubSys System, TestStart, TestPreparation TestExecution, TestFailed, TestPassed	Completeness
TestId	Accuracy: Duplicated values
TestStatus TestState	Accuracy: Syntactic errors There are reference domains to be used
TestType	Accuracy: Syntactic errors Reference domain can be established out of the snapshots – There is not reference domain established since not enough information could be obtained.

Table B16. Attributes and metrics Risk

9.2 Measurements

In the following tables the results of the measurements made for the metrics selected are presented. For every metric the average number of errors and the average simple ratio were calculated and are shown as the final result.

For example in the case of completeness for the field "System", the number of null values was measured in every one of the snapshots that belong to the category Architecture; then the average number of null fields for this field was calculated using the results obtained for all the snapshots.

In the case of the average simple ratio, also the simple ratio for every snapshot was calculated following the method explained in the chapter 3: the number of values with errors is divided by the total number of values, and then the result is subtracted from 1. After the calculation of the simple ratio of the attribute was made for every snapshot, an average simple ratio was calculated using the results of all snapshots in the category evaluated. The resulting average gives a percentage of quality level for the attribute in the dimension evaluated

9.2.1 Company 1

Architecture

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Product_SubSys	0.014084507	1288.7606	0.9999894 = 99.99%
System	0.014084507	1288.7606	0.9999894 = 99.99%
SubSystem	0.014084507	1288.7606	0.9999894 = 99.99%
CmpGroup	0.014084507	1288.7606	0.9999894 = 99.99%
Component	0.014084507	1288.7606	0.9999894 = 99.99%

Table B17. Measurements for Architecture

Defect

Subcategory 1

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Project*	46.166668	527.2222	0.9219657 = 92.20%
Team	0	527.2222	1.0 = 100%
ProdSubSys	0	527.2222	1.0 = 100%
System	0	527.2222	1.0 = 100%
DefectEstimate*	301.5	301.5	0.0 = 0%
DefectCost*	527.2222	527.2222	0.0 = 0%

Injected*	527.2222	527.2222	0.0 = 0%
Detected*	527.2222	527.2222	0.0 = 0%
DefectAnalysis*	297.27777	527.2222	0.44881868 = 44.88%
DefectResolution*	513.19446	527.2222	0.027010806 = 2.7 %
DefectEvaluation*	114.97222	527.2222	0.78496504 = 78.49%
DefectFinish*	114.97222	527.2222	0.78496504 = 78.49%
	Accuracy: Duplicated values		
DefectId	0	527.2222	1.0 = 100%
	Accuracy: Syntactic errors		
DefectState	0	527.2222	1.0 = 100%
DefectStatus	12.5	527.2222	0.979489 = 97.94%
Severity	0	527.2222	1.0 = 100%
Priority	0	527.2222	1.0 = 100%
	Consistency: Edit rules		
DefectStart < DefectAnalysis**	0	225.22223	1.0 = 100%
DefectAnalysis < DefectResolution**	1.75	8.416667	0.8023568 = 80.23%
DefectResolution < DefectEvaluation**	0.5555556	13.694445	0.9630406 = 96.30%
DefectEvaluation < DefectFinish**	0	401.52777	1.0 = 100%

Table B18. Measurements for Defect 1

* This incomplete values are considered because of the presence of the symbol "?" instead of the name of the project.

** These fields are only compared in the case that both values exist; therefore, the calculation of the simple ratio in every file is made with respect to the total number of records where both values exists, and not to the total number of records in the file. The high values of the ratio are only taking into account that dates exist, but in general terms it is not correct to say that these fields contain a high quality level since many of the information is missing.

Subcategory 2

Name of the field	***Completeness***		
	Average number of error values	***Average number of total values***	***Average simple ratio***
Program	0	1718.1711	1.0 = 100%
Project*	205.55856	1718.1711	0.88324195 = 88.32%
Team	0	1718.1711	1.0 = 100%
Prodsubsys	0	1718.1711	1.0 = 100%
System	0	1718.1711	1.0 = 100%
DefectCost*	1718.1711	1718.1711	0.0 = 0%
Injected*	1718.1711	1718.1711	0.0 = 0%
Detected*	1718.1711	1718.1711	0.0 = 0%
DefectAnalysis*	1345.8739	1718.1711	0.2168106 = 21.68%
DefectResolution*	1672.919	1718.1711	0.026404854 = 2.64%
DefectEvaluation*	478.84683	1718.1711	0.7140664 = 71.40%
DefectFinish	478.84683	1718.1711	0.7140664 = 71.40%
	Accuracy: Duplicated values		
DefectId	0	1718.1711	1.0 = 100%
	Accuracy: Syntactic errors		
DefectType	0	1718.1711	1.0 = 100%

DefectState	0	1718.1711	1.0 = 100%
DefectStatus	134.36937	1718.1711	0.92346597 = 92.35%
Severity	368.9189	1718.1711	0.7956158 = 79.56%
Priority	304.17117	1718.1711	0.7656734 = 76.56%
	Consistency: Edit rules		
DefectStart < DefectAnalysis**	0	372.2973	1.0 = 100%
DefectAnalysis < DefectResolution**	4.6936936	17.738739	0.7417263 = 74.17%
DefectResolution < DefectEvaluation**	5.5945945	45.25225	0.8761759 = 87.61%
DefectEvaluation < DefectFinish**	0	1239.3243	1.0 = 100%

Table B19. Measurements for Defect 2

*This incomplete values are considered because of the presence of the symbol "?" instead of the name of the project.

** These fields are only compared in the case that both values exist; therefore, the calculation of the simple ratio in every file is made with respect to the total number of records where both values exists, and not to the total number of records in the file.

Subcategory 3

Some files don't have any content and therefore they were not used for the measurement of errors.

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
System	0	4161.7534	1.0 = 100%
SubSystem	0	4161.7534	1.0 = 100%
Product_subsys	0	4161.7534	1.0 = 100%
Program (This field is not present in all files)	0	2279.8523	1.0 = 100%
Priority	62.503735	4161.7534	0.89522976 = 89.52%
Problem_type	58.99564	4161.7534	0.8953923 = 89.53%
Defect_type	3717.5867	4161.7534	0.0631594 = 6.31%
Scope	4161.7534	4161.7534	0.0 = 0%
Caused_during	1217.0934	4161.7534	0.68420625 = 68.42%
Discovered_during	746.89355	4161.7534	0.79384345 = 79.38%
Act_total_eff	1383.6562	4161.7534	0.5730907 = 57.31%
Submitted_time	71.432755	4161.7534	0.8439392 = 84.39%
Analysed_time	2503.1438	4161.7534	0.31958532 = 31.95%
Resolved_time	1321.0803	4161.7534	0.4746583 = 47.46%
Evaluated_time	1511.9159	4161.7534	0.41079974 = 41.08%
	Accuracy: Duplicated values		
Problem_number	335.85928	4161.7534	0.9581028 = 95.81%
	Accuracy: Syntactic errors		
Priority*	22.284422	4578.1606	0.9451274 = 94.51%

Severity	0	4161.7534	1.0 =100%
Problem_type*	0	4582.0786	1.0 =100%
Request_type	0	4161.7534	1.0 =100%
Crstatus**	44.339973	4161.7534	0.9252044 = 92.25%
	Consistency: outliers (negative values)		
Act_total_eff***	0.36537102	3153.091	0.99994797 = 99.99%
	Consistency: edit rules		
Create_time < Submitted_time	0	4626.095	1.0 = 100%
Submitted_time < Analysed_time****	4.0402083	1982.134	0.9922737 = 99.23%
Analysed_time < Resolved_time****	33.3693	1107.8488	0.9818279= 98.18%
Resolved_time < Evaluated_time****	3.9413128	3253.5027	0.9964815 = 99.65%

Table B20. Measurements for Defect 3

* The calculation of the syntactic errors is made only for the fields that contain a value. There are some incomplete fields; therefore the simple ratio is calculated with respect to the total number of fields that have a value, and not to the total number of fields in the file.

** The domain reference in the documentation contains fewer values than the values that could be introduced in this field. This is the reason why some syntactic errors were found. More information about the values in the reference domain was not found.

*** The calculation of the outliers is made only for the fields that contain a value, since there are many that are incomplete. Therefore, the simple ratio is calculated with respect to the total number of fields that have a value, and not to the total number of fields in the file.

**** These fields are only compared in the case that both values exist; therefore, the calculation of the simple ratio in every file is made with respect to the total number of records where both values exists, and not to the total number of records in the file.

Project

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Text4 (type of activity)	109.775055	1142.9816	0.9098861 = 90.99%
Text15 (program)	0	1085.814	1.0 = 100%
Text16 (project)	102.09064	1142.9816	0.9054161 = 90.54%
Text17 (team)	102.46049	1142.9816	0.9031471 = 90.31%
Text25 (System)	102.15552	1142.9816	0.90529555 = 90.52%
Text26 (subsystem)	102.56021	1142.9816	0.9043652 = 90.43%
Text27 (component group)	102.9644	1142.9816	0.90398824 = 90.39%
Text28 (component)	103.05176	1142.9816	0.90379804 = 90.37%
	Accuracy: duplicated values		
Unique_ID	0	1148.0818	1.0 = 100%
	Accuracy: syntactic errors		
Flag 10	0	1142.9816	1.0 = 100%
	Outliers: negative numbers		
Baseline_Work	0	1142.9816	1.0 = 100%
Scheduled_Work	0	1142.9816	1.0 = 100%
Actual_Work	0	1142.9816	1.0 = 100%
	Consistency: edit rules		
Baseline_Start < Baseline_Finish	0	1142.9816	1.0 = 100%

Start_Date < Finish_Date	0	1142.9816	1.0 = 100%
Actual_Start < Actual_Finish	0.0025759917	857.4413	0.99998677 = 99.99%

Table B21. Measurements for Project

Review

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Project	0	1209.4517	1.0 = 100%
Team	0	1209.4517	1.0 = 100%
System	0	1209.4517	1.0 = 100%
ProdSubsys	0	1209.4517	1.0 = 100%
	Consistency: edit rules		
InitiationDate < ClosureDate	0.61290324	1209.4517	0.999497 = 99.94%
	Accuracy: Syntactic errors		
ActivityType	0	1209.4517	1.0 = 100%
Type	0	1209.4517	1.0 = 100%
Severity	0	1209.4517	1.0 = 100%
State	0.87096775	1209.4517	0.99927557 = 99.93%
Unit	0	1209.4517	1.0 = 100%
	Consistency: outliers (negative numbers)		
NofParticipants	0	1209.4517	1.0 = 100%

PreparationEffort	0	1209.4517	1.0 = 100%
MeetingEffort	0	1209.4517	1.0 = 100%
ReworkEffort	0	1209.4517	1.0 = 100%
VerificationEffort	0	1209.4517	1.0 = 100%
ReviewSize	0	1209.4517	1.0 = 100%
MajorDefects	0	1209.4517	1.0 = 100%
MinorDefects	0	1209.4517	1.0 = 100%
TotalEffort	0	1209.4517	1.0 = 100%
PreparationRate	0	1209.4517	1.0 = 100%
RemovalRate	0	1209.4517	1.0 = 100%
AverageSize	0	1209.4517	1.0 = 100%
DefectCost	0	1209.4517	1.0 = 100%
LeadTime	0	1209.4517	1.0 = 100%

Table B22. Measurements for Review

Size

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Program (this field is not in all files)	0	1516.6239	1.0 = 100%
System	0	8731.976	1.0 = 100%
ProdSubSys	0	8731.976	1.0 = 100%
	Accuracy: Syntactic errors		

Unit (only value found is ncsl)	8731.948	8731.948	0 = 0%
	Consistency: outliers (negative numbers)		
Total	0	8731.976	1.0 = 100%
Blank	0	8731.976	1.0 = 100%
Comment	0	8731.976	1.0 = 100%
Deleted	0	8731.976	1.0 = 100%
Equal	0	8731.976	1.0 = 100%
Moved	0	8731.976	1.0 = 100%
Modified	0	8731.976	1.0 = 100%
Added	0	8731.976	1.0 = 100%
	Consistency: edit rules		
Source = Equal + moved + modified + added	0	8731.976	1.0 = 100%
Delta = Modified + added	0	8731.976	1.0 = 100%

Table B23. Measurements for size

9.2.2 Company 2

Architecture

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Product_SubSys	0	9.5	1.0 = 100%
System	0	9.5	1.0 = 100%
SubSystem	0	9.5	1.0 = 100%
CmpGroup	0	9.5	1.0 = 100%
Component	0	9.5	1.0 = 100%

Table B24. Measurements for Architecture

Case

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Program	0	303.0771	1.0 = 100%
Project	0.1667774	303.0771	0.99667776 = 99.67%
Team	0	303.0771	1.0 = 100%
ProdSubSys	0	303.0771	1.0 = 100%
System	0	303.0771	1.0 = 100%
CaseFinish*	303.02524	303.0771	0 = 0%
	Accuracy: Duplicated values		
CaseId	0	303.0771	1.0 = 100%
	Accuracy: Syntactic errors		

CaseState	0	303.0771	1.0 = 100%
CaseStatus	288.60132	303.0771	0.012111656 = 1.21%
	Consistency: edit rules		
CaseStart < CaseFinish**	0	1	1.0 = 100%

Table B25. Measurements for Case

* In this field the incomplete values are due to the presence of the symbol "?"

** This calculation is only made in the cases where both fields exist. Therefore the simple ratio is calculated with respect to the number of records that are complete, and not to the total number of records.

Change

Name of the field	***Completeness***		
	Average number of error values	***Average number of total values***	***Average simple ratio***
Program	0	162.94151	1.0 = 100%
Project	0	162.94151	1.0 = 100%
Team	0	162.94151	1.0 = 100%
System	0	162.94151	1.0 = 100%
Subsystem	0	162.94151	1.0 = 100%
CmpGroup	0	162.94151	1.0 = 100%
Component	0	162.94151	1.0 = 100%
CreateDate	0	162.94151	1.0 = 100%
ApprovalDate	58.47953	162.94151	0.64679176 = 64.68%
LastModified	0	162.94151	1.0 = 100%
Draft	0	162.94151	1.0 = 100%

Raised	49.526318	162.94151	0.74784744 = 74.78%
Assigned	91.55556	162.94151	0.23304215 = 23.30%
Completed	104.748535	162.94151	0.17057678 = 17.06%
Approval	76.67836	162.94151	0.54850984 = 54.85%
Approved	58.47953	162.94151	0.64679176 = 64.67%
Closed	94.17544	162.94151	0.36166227 = 36.17%
Cancelled	131.61403	162.94151	0.10544653 = 10.54%
Rejected	154.2807	162.94151	0.017934805 = 1.8%
	Accuracy: Syntactic errors		
ChangeState	32.105263	162.94151	0.88975924 = 88.97%
ChangeStatus	83.128654	162.94151	0.39297074 = 39.30
Category	0	162.49123	1.0 = 100%
Priority	0	156.77193	1.0 = 100%
	Consistency: Outliers(negative values)		
EstCostsEUR	0.3888889	39.642857	0.99249196 = 99.24%
EstCostsMD	3.1949685	136.55975	0.9410501 = 94.10%
EstContingencyMD	0.28865978	11.659794	0.9333873 = 93.34 %

Table B27. Measurements for Change

Defect

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Program	0	382.7927	1.0 = 100%

Project	0	382.7927	1.0 = 100%
Team	0	382.7927	1.0 = 100%
ProdSubsys	0	382.7927	1.0 = 100%
System	0	382.7927	1.0 = 100%
DefectState	0.1527861	382.7927	0.9996716 = 99.96
Severity	262.22528	382.7927	0.38770524 = 38.77%
DefectOpen	12.574372	381.3458	0.91524696 = 91.52%
DefectAnalysis	12.261833	382.7927	0.9193228 = 91.93%
DefectResolution	14.177951	382.7927	0.9125623 = 91.25%
DefectEvaluation	14.9173155	382.7927	0.9112846 = 91.12%
DefectFinish	39.838825	382.7927	0.79359597 = 79.36%
	Accuracy: Duplicated values		
DefectId	0	382.7927	1.0 = 100%
	Accuracy: Syntactic errors		
DefectType	0	382.7927	1.0 = 100%
DefectState*	0	382.7927	1.0 = 100%
DefectStatus	18.77112	339.47513	0.8699524 = 87%
Priority	76.709404	378.3667	0.8346582 = 83.46%
Severity*	0	165.211	1.0 = 100%
	Consistency: edit rules		
DefectStart < DefectOpen**	0.43337646	370.44113	0.99420184 = 99.42%
DefectOpen < DefectAnalysis**	0.15976714	370.44113	0.99965465 = 99.96%

DefectAnalysis < DefectResolution**	0.2635379	370.16727	0.99933 = 99.93%
DefectResolution < DefectEvaluation**	2.087846	369.42477	0.9942107 = 99.42
DefectEvaluation < DefectFinish**	4.5448275	358.8652	0.9933143 = 99.33%

Table B28. Measurements for Defect

*The syntactic errors for this value are calculated only for the fields that have a value. Some of them are incomplete or have the character "?". Therefore the calculation of the simple ratio is made with respect to the total number of fields that are complete, and not to the total number of fields that exist.

**The calculation of these edit rules was made only for the cases where both values exists. Therefore the simple ratio is made with respect to the total number of fields that are complete, and not to the total number of fields that exist.

Issue

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Program	0	81.76271	1.0 = 100%
Project	0	81.76271	1.0 = 100%
Team	0	81.76271	1.0 = 100%
System	0	81.76271	1.0 = 100%
SubSystem	0	81.76271	1.0 = 100%
CmpGroup	0	81.76271	1.0 = 100%
Component	0	81.76271	1.0 = 100%
Completed	24.83051	81.76271	0.57453686 = 57.45%
Closed	13.734464	81.76271	0.6968171 = 69.68%

	Accuracy: Syntactic errors		
IssueState	5.4519773	81.76271	0.94493544 = 94.5%
IssueStatus	13.734464	81.76271	0.6968171 = 69.68%
Criticality	0.32768363	80.519775	0.9924305 = 99.24%
	Consistency: Edit rules		
CreateDate < Completed*	0.73939395	61.072727	0.989928 = 99%
CreateDate < Closed*	0	77.68387	1.0 = 100%

Table B29. Measurements for Issue

* The calculation of these edit rules was made only for the cases where both values exists. Therefore the simple ratio is made with respect to the total number of fields that are complete, and not to the total number of fields that exist.

Project

Four of the files could be read for these measurements.

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Text15 *	0	345.33966	1.0 = 100%
Text16*	0	345.33966	1.0 = 100%
Text17*	0	345.33966	1.0 = 100%
Text25*	0	345.33966	1.0 = 100%
Text27*	0	345.33966	1.0 = 100%
Text28	0	345.33966	1.0 = 100%
	Accuracy: Syntactic errors		
Flag10	0	700.92554	1.0 = 100%

	Consistency: Outliers		
Initial_work	6.010718	700.92554	0.9905695 = 99.06%
Baseline_Work	1.1575757	277.56537	0.99606353 = 99.61%
Scheduled_Work	7.103935	700.92554	0.9885842 = 98.86%
Actual_Work	7.7320547	700.92554	0.98781985 = 98.78%
Psn_Work	0.121139474	700.92554	0.9999218 = 99.99%
ETC	4.598646	700.92554	0.9807835 = 98.07%
	Consistency: edit rules		
Initial_Start < Initial_finish	3.0284867	700.92554	0.99374664 = 99.37
Baseline_Start < Baseline_Finish	0.045021646	277.56537	0.99948096 = 99.94%
Start_Date < Finish_Date	0.089127064	700.92554	0.9993087 = 99.93%
Actual_Start < Actual_Finish	0	443.21945	1.0 = 100%
Psn_Start < Psn_Finish	193.51643	700.92554	0.5827201 = 58.27%

Table B30. Measurements for Project

*This field is not present in all files

Requirements

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Program	0	1645.7375	1.0 = 100%

Project	0	1645.7375	1.0 = 100%
Team	0	1645.7375	1.0 = 100%
ProdSubSys	0	1645.7375	1.0 = 100%
System	0	1645.7375	1.0 = 100%
Priority	1383.7932	1645.7375	0.49051768 = 49.05%
ReqDesign	0.021327015	1645.7375	0.9990732 = 99.90%
ReqPreparation	998.3756	1645.7382	0.46825817 = 46.82%
ReqExecution	1229.7299	1645.7382	0.27986038 = 27.99%
ReqFailed	1495.1161	1645.7382	0.091170475 = 9.11%
ReqPassed	1355.4852	1645.7382	0.2315096 = 23.15%
	Accuracy: Duplicated values		
ReqId	0	1645.7375	1.0 = 100%
	Accuracy: Syntactic errors		
ReqState	0	1645.7375	1.0 = 100%
	Consistency: Edit rules		
ReqStart < ReqDesign *	0.6552133	1645.7162	0.99749887 = 99.75%
ReqDesign < ReqPreparation*	11.627858	757.27515	0.97675234 = 97.67%
ReqPreparation < ReqExecution*	81.546715	542.25635	0.9264297 = 92.64%
ReqExecution < ReqFailed*	46.034775	215.64885	0.6213494 = 62.13%
ReqFailed < ReqPassed*	4.847518	151.33777	0.97008663 = 97%

Table B31. Measurements for Requirements

* The calculation of these edit rules was made only for the cases where both values exists. Therefore the simple ratio is made with respect to the total number of fields that are complete, and not to the total number of fields that exist.

Risk

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Program	0	140.52792	1.0 = 100%
Project	0	140.52792	1.0 = 100%
Team	0	140.52792	1.0 = 100%
System	0	140.52792	1.0 = 100%
SubSystem	0	140.52792	1.0 = 100%
CmpGroup	0	140.52792	1.0 = 100%
Component	0	140.52792	1.0 = 100%
CreateDate	0	140.52792	1.0 = 100%
LastModified	0	140.52792	1.0 = 100%
Draft	0	140.52792	1.0 = 100%
Raised	46.888325	140.52792	0.6629094 = 66.29%
Assigned	33.588833	140.52792	0.6313997 = 63.13%
Completed	42.335026	140.52792	0.53056604 = 53.06%
TargetDate	43.51639	184.65573	0.64458096 = 64.49%
	Accuracy: Syntactic errors		
RiskState	9.329949	140.52792	0.900995 = 90.1%
RiskStatus	26.395939	140.52792	0.67121965 = 67.12%
	Consistency: Edit rules		

CreateDate < Assigned	19.926554	119.0226	0.8541393 = 85.41%
CreateDate < Completed	0.2881356	109.28814	0.9961148 = 99.61
CreateDate < Closed	0.023121387	129.96532	0.99669695 = 99.67%
CreateDate < Cancelled	0	15.106558	1.0 = 100%

Table B32. Measurements for Risks

Test

Name of the field	*Completeness*		
	Average number of error values	*Average number of total values*	*Average simple ratio*
Program	0	1444.3602	1.0 = 100%
Project	1.8450813	1444.3602	0.9992254 = 99.92%
Team	0	1444.3602	1.0 = 100%
ProdSubSys	0	1444.3602	1.0 = 100%
System	0	1444.3602	1.0 = 100%
TestStart	0	1444.3602	1.0 = 100%
TestPreparation	1444.3602	1444.3602	0 = 0%
TestExecution	1444.3602	1444.3602	0 = 0%
TestFailed	1444.3602	1444.3602	0 = 0%
TestPassed	1444.3602	1444.3602	0 = 0%
	Duplicated values		
TestId	0	1444.3602	1.0 = 100%

	Accuracy: Syntactic errors		
TestState	0	1444.3602	1.0 = 100%

Table B33. Measurements for Test

10 Appendix C: Database Documentation

In this appendix the information concerning the database structure and implementation is documented. In the first section the specifications about the DBMS used are given. Then, in the second section, the database entities and their attributes are described.

10.1 DBMS

The DBMS selected to implement the database is MySQL 5.1 [MYS09]. It was chosen because it is open source and a dump of the database can be created in order to make it portable to be copied and accessed from any pc where the MySQL server is installed.

10.2 Database description

In the following tables every entity of the database is described indicating the type of information it stores, and for every attribute it is explained the data type used and the information related to the data stored there.

10.2.1 Company

This entity represents the companies for which the information of the software development process is stored in the database. A company has many products and many programs.

The fields are described in table C1:

Name	*Data Type*	*Description*
companyId	INT	Primary Key. Id of the entity Company
companyName	VARCHAR(100)	Name of the company

Table C1. Company Entity

10.2.2 Program

This entity represents the programs that are created in the companies for developing software applications. A program is a sequence of projects where one project follows another.

The fields are described in table C2:

Name	*Data Type*	*Description*
programId	INT	Primary key. Id of the entity Program
programName	VARCHAR(100)	Name of the program
companyId	INT	Foreign key. Id of the entity Company that represents the company where the program is implemented

Table C2. Program Entity

10.2.3 Project

This entity represents the projects that compose a program. The goal of a project is to develop one or more products through a set of activities (which are sequences of tasks). Every project is performed by one or more teams.

The fields are described in table C3:

Name	*Data Type*	*Description*
projectId	INT	Primary key. Id of the entity Project.
projectName	VARCHAR(100)	Name of the project
programId	INT	Foreign key. Id of the entity Program that represents the program to which the project belongs.

Table C3. Project Entity

10.2.4 Team

This entity represents the teams that perform a project through the development of the project's activities (sequences of tasks). As every product can be decomposed to the component level, the teams are in charge to define, realize and assemble a collection of a set of components that belong to the product that is being developed in a project. Every team can be part of one or more projects.

The fields are described in table C4:

Name	Data Type	Description
teamId	INT	Primary key. Id of the entity Team
teamName	VARCHAR(100)	Name of the team

Table C4. Team Entity

10.2.5 ProjectTeam

This is an intermediate entity used to break the many-to-many relationship between the entity project and the entity team.

The fields are described in table C5:

Name	Data Type	Description
projectId	INT	Foreign key. Id of the entity Project
TeamId	INT	Foreign key. Id of the entity Team
proTeamId	INT	Primary key. Id of the entity ProjectTeam

Table C5. ProjectTeam Entity

10.2.6 Product

This entity represents the products that are developed at the companies. A product can have many versions, and every one of these versions is called a system.

The fields are described in table C6:

Name	Data Type	Description
productId	INT	Primary key. Id of the entity Product
productName	VARCHAR(100)	Name of the product
companyId	INT	Foreign key. Id of the entity Company that represents the company to which the product belongs

Table C6. Product Entity

10.2.7 System

This entity represents the systems that are versions of a product. A system is composed by many subsystems.

The fields are described in table C7:

Name		Description
systemId	INT	Primary key. Id of the entity System
systemName	VARCHAR(100)	Name of the system
productId	INT	Foreign key. Id of the entity Product that represents the product of which the system is a version

Table C7. System Entity

10.2.8 Subsystem

This entity represents the subsystems that compose a system. Every subsystem is composed by one or more groups of components.

The fields are described in table C8:

Name	Data Type	Description
subsystemId	INT	Primary key. Id of the entity Subsystem
subsystemName	VARCHAR(100)	Name of the subsystem
systemId	INT	Foreign key. Id of the entity System that represents the system to which the subsystem belongs

Table C8. Subsystem Entity

10.2.9 GroupComponent

This entity represents the groups of components that compose a subsystem. A group of components is composed by one or more components.

The fields are described in table C9:

Name	Data Type	Description
groupId	INT	Primary key. Id of the entity GroupComponent
groupName	VARCHAR(100)	Name of the group of components
subsystemId	INT	Foreign key. Id of the entity Subsystem that represents the subsystem to which the group of components belong

Table C9. GroupComponent Entity

10.2.10 Component

This entity represents the components that compose a group of components. The components are developed by teams through activities (series of tasks) that produce workproducts (such as specifications, designs, sources, tests and others) that are part of the process development of the components.

The fields are described in table C10:

Name	Data Type	Description
componentID	INT	Primary key. Id of the entity Component
componentName	VARCHAR(100)	Name of the component
external	INT	Indicates that some part of a system is not created by the program / project / team but delivered by or bought from an external party
groupId	INT	Foreign key. Id of the entity Group that represents the group of components to which the component belongs.

Table C10. Component Entity

10.2.11 SizeData

This entity represents the size information of a component which is measured by a team. For a component the size measurement can be performed one or more times. Every team can perform one or many measurements of size.

The fields are described in table C11:

Name	*Data Type*	*Description*
sizeDataId	INT	Primary key. Id of the entity Size
historyDate	DATETIME	Date when the information about size was stored. It is used to keep historical data.
baseRootPath	VARCHAR(100)	Path to the root directory of the base version
newRootPath	VARCHAR(100)	Path to the root directory of the new version
unit	VARCHAR(100)	Unit of measure to count the code lines
total	INT	Total number of code lines
blank	INT	Number of blank lines in the code file
comment	INT	Number of comment lines in the code file
deleted	INT	Number of lines deleted in the code file
equal	INT	Number of unaltered identical lines
moved	INT	Number of lines moved in the code file
modified	INT	Number of lines modified in the code file

added	INT	Number of lines added to the code in the file
source	INT	Number of lines in the original source Equal + moved+ modified + deleted
delta	INT	It is equal to the number of lines Modified + added
file	VARCHAR(100)	Name of the file with the code
type	VARCHAR(100)	Type of file. Depends on the programming language
matchPath	VARCHAR(100)	Is a file has been moved from one place to another this is the other location
matchFile	VARCHAR(100)	Is a file has been given another name, this is the other name
path	VARCHAR(100)	The relative path below the baserootpath / new rootpath where the file resides
programId	INT	Foreign key. Id of the entity Program that represents the program where the components for which the size of the code is measured are
componentID	INT	Foreign key. Id of the entity Component that represents the component for which the size is measured

Table C11. SizeData Entity

10.2.12 IssueData

This entity represents the issue information of components which is managed by a team. Every team can have many issues. One team can manage issues of many components.

The fields are described in table C12:

Name	*Data Type*	*Description*
issueDataId	INT	Primary key. Id of the entity IssueData
historyDate	DATETIME	Date when the information about the issue was stored. It is used to keep historical data.
issueIdent	VARCHAR(100)	Identification of the issue as it was obtained from the original database of the companies
issueState	VARCHAR(100)	Main state of the issue
issueStatus	VARCHAR(100)	Status is the sub state of the main state
category	VARCHAR(100)	Category of the issue
criticallity	VARCHAR(100)	Criticality of the issue
phaseDetected	VARCHAR(100)	Phase of the project when the issue was detected
createDate	DATETIME	Date when the issue was created
lastModified	DATETIME	Last time the issue was modified
draft	DATETIME	Date the draft of the issue was created
raised	DATETIME	Date the issue was raised
assigned	DATETIME	Date the issue was assigned to be solved
completed	DATETIME	Date the issue was completely solved
closed	DATETIME	Date the issue was closed
cancelled	DATETIME	Date the issue was cancelled

proTeamId	INT	Foreign key. Id of the entity ProjectTeam that represents the team which manages the issue and the Project to which the Team belongs
componentID	INT	Foreign key. Id of the entity Component that represents the component for which the issue was raised

Table C12. IssueData Entity

10.2.13 ChangeData

This entity represents the information about change requests made for a component, which are managed by a team. Every component can have one or many change requests. One team can manage one or many change requests.

The fields are described in table C13:

Name	*Data Type*	*Description*
changeDataId	INT	Primary key. Id of the component ChangeData
historyDate	DATETIME	Date when the information about the change was stored. It is used to keep historical data
changeIdent	VARCHAR(100)	Identification of the change request as it was obtained from the original database of the company
changeState	VARCHAR(100)	Main state of the change
chanteStatus	VARCHAR(100)	Status is the sub state of the main state
changeApproved	VARCHAR(100)	Was the change request approved? Yes/NO
category	VARCHAR(100)	Classification of the change request

rootCause	VARCHAR(100)	Cause of the change request
priority	VARCHAR(100)	Ordering to address change requests in the project
detected	VARCHAR(100)	Phase of the project when the need of a change was detected
targetDate	DATETIME	Date when the change request should be solved
estCostEUR	FLOAT	Estimated cost in euro of solving the change request
estCostMD	FLOAT	Estimated cost in mandays in additional budget for solving the change request
estContingencyMD	FLOAT	Estimated cost in mandays from contingency budget for solving the change request
createDate	DATETIME	Date the change request was created
approvalDate	DATETIME	Date the change request was approved
lastModified	DATETIME	Last time the change request status was modified
draft	DATETIME	Date when the draft of the change request was created
raised	DATETIME	Date when the change request was raised
assigned	DATETIME	Date the change request was assigned to be analyzed
completed	DATETIME	Date the change request was completed
approval	DATETIME	Date the change request started the procedure for approval

approved	DATETIME	Date the change request was approved
closed	DATETIME	Date the change request was closed
cancelled	DATETIME	Date the change request was cancelled
rejected	DATETIME	Date the change request was rejected
proTeamId	INT	Foreign key. Id of the entity ProjectTeam that represents the team which manages the issue and the Project to which the Team belongs
componentID	INT	Foreign key. Id of the entity Component that represents the entity component for which the change was requested.

Table C13. ChangeData Entity

10.2.14 RiskData

This entity represents the information about risks present during the development of a component, which are managed by a team. One component can have one or more risks associated. One team can manage one or more risks.

The fields are described in table C14:

Name	*Data Type*	*Description*
riskDataId	INT	Primary key. Id of the entity RiskData
historyDate	DATETIME	Date when the data about the risk was stored. It is used to keep historical data
riskId	VARCHAR(100)	Identification of the risk as it was obtained from the original database of the company

riskState	VARCHAR(100)	Main state of the risk
riskStatus	VARCHAR(100)	Status is the sub state of the main state
category	VARCHAR(100)	Category of the risk
probability	VARCHAR(100)	Probability that the risk will become true
impact	VARCHAR(100)	Impact on the project in case the risk become true
exposure	VARCHAR(100)	Level of exposure to the risk
phaseDetected	VARCHAR(100)	Phase in which the risk was detected
phaseImpacted	VARCHAR(100)	Phase that would be affected in case the risk become true
createDate	DATETIME	Date the risk was created
lastModified	DATETIME	Last time the risk statement was modified
draft	DATETIME	Date the draft of the risk document was created
raised	DATETIME	Date the risk was raised
assigned	DATETIME	Date the risk was assigned to be analyzed
completed	DATETIME	Date the risk analysis was completed
closed	DATETIME	Date the risk was assigned status closed
cancelled	DATETIME	Date the risk was cancelled
targetDate	DATETIME	Date the risk is to be mitigated
proTeamId	INT	Foreign key. Id of the entity ProjectTeam that represents the

		team which manages the issue and the Project to which the Team belongs
componentID	INT	Foreign key. Id of the entity Component that represents the component for which the risk is being analyzed and managed.

Table C14. RiskData Entity

10.2.15 Task

This entity represents the information of the tasks that are performed by teams in the development of a component. One component is developed through one or more tasks. One team can perform one or more tasks.

The fields are described in table C15:

Name	*Data Type*	*Description*
taskId	INT	Primary key. Id of the entity Task
historyDate	DATETIME	Date when the information about the task was stored
uniqueId	VARCHAR(100)	Unique Id of the task as it was obtained from the original database of the company
outlineNumber	VARCHAR(100)	The structural ordering of the task in the file, 1 comes before 2. 1.1 is the first child, etc.
milestone	VARCHAR(100)	Is the task a Milestone? Yes/No. YES if task is a milestone
summary	VARCHAR(100)	Is the task a summary task? Yes/No. A summary task is a parent and as such, it is the sum of all its child tasks. YES means the task is a parent.
name	VARCHAR(2000)	The description of the task as it was obtained from the original database.

flag10	INT	Indicates task completion percentage (either 0 / 100). If it is 0 the task is ongoing. Yes/No (100/0)
InitialStart	DATETIME	Is the baseline start when first baselines; later baselining may cause baseline start to differ.
InitialFinish	DATETIME	Is the baseline finish when first baselines; later baselining may cause baseline finish to differ
InitialWork	FLOAT	Is the baseline work when first baselines; later baselining may cause baseline work to differ
startDate	DATETIME	Start date of the current plan
finishDate	DATETIME	Finish date of the current plan
scheduledWork	FLOAT	Scheduled work for the current plan
baselineStart	DATETIME	It is a copy of the start date that is made once the plan is approved
baselinefinish	DATETIME	It is a copy of the finish date that is made once the plan is approved
baselineWork	FLOAT	It is a copy of the scheduled work which is made once the plan is approved
actualStart	DATETIME	This date reflects the progress of the task. The actual start date of the task. When it is set, the scheduled start is set to this.
actualFinish	DATETIME	Reflect progress. It is only recorded when the task is 100% complete
actualWork	FLOAT	It is the effort spent between start and finish

activityType	VARCHAR(100)	The type of activity involved
deliverable	VARCHAR(100)	If non-empty indicates that the task has a deliverable with it
release	VARCHAR(100)	Release of the task
resources	VARCHAR(5000)	The names of the persons working on the task
ETC	FLOAT	Estimate to complete in mandays
Stage	VARCHAR(100)	Stage of the project – initiation - execution
Phase	VARCHAR(100)	Same as activity type
Skill	VARCHAR(300)	type of resource needed for the task
TaskNumber	VARCHAR(100)	Identifier
Predecessors	VARCHAR(1000)	Tasks that this task depends on
Successors	VARCHAR(1000)	Tasks that depend on this task
CriticalPath	VARCHAR(1000)	See the literature on critical path and critical chain in a network planning
Psn_Start	DATETIME	from PSN
Psn_Finish	DATETIME	from PSN
Psn_Work	FLOAT	from PSN
ProjectID	VARCHAR(100)	Identifier of the task higher in the tree
PIC	VARCHAR(100)	Identifier within the account to book the costs
SubProjName	VARCHAR(1000)	From PSN for use in Crosslinks – some schedules have sub-schedules that are kept in separate files – this is the name of such a

		file
SubProjTaskId	VARCHAR(1000)	From PSN for use in Crosslinks - This identified refers to a specific task in the subschedule
proTeamId	INT	Foreign key. Id of the entity ProjectTeam that represents the team which manages the issue and the Project to which the Team belongs
componentID	INT	Foreign key. Id of the entity Component that represents the component to which the task belongs

Table C15. Task Entity

10.2.16 RequirementsData

This entity represents the requirements associated to the development of a component, which are managed by a team. A component can be developed based on one or more requirements. A team can manage one or more requirements. A requirement can be tested by one or more tests.

The fields are described in table C16:

Name	*Data Type*	*Description*
requirementDataId	INT	Primary key. Id of the entity RequirementsData
historyDate	DATETIME	Date when the information about the requirement was stored
reqId	VARCHAR(100)	Id of the requirement as it was obtained from the original database of the company
reqTraceBack	INT	Reference to a higher level requirement in another database
reqChildren	INT	Reference to more detailed

		requirements below this requirement
reqParent	INT	Reference to the higher level requirement above the current one
reqOrder	INT	Sequence number used to determine the order of the requirements when printing / reading
reqReview	VARCHAR(100)	Status of review of the requirement
priority	VARCHAR(100)	Priority of the requirement
reqType	VARCHAR(100)	Type of requirement
reqState	VARCHAR(100)	Main state of the requirement
reqStatus	VARCHAR(100)	Status is the sub state of the main state
reqStart	DATETIME	Creation of the requirement
reqDesign	DATETIME	Date when the requirement started design phase
reqPreparation	DATETIME	Date when the requirement started preparation for being implemented
reqExecution	DATETIME	Date when the test was executed
reqFailed	DATETIME	Date when the test failed complying the requirement
reqPassed	DATETIME	Date when the test passed
LinkedTest	VARCHAR(2000)	Tests that are used to test the requirement
proTeamId	INT	Foreign key. Id of the entity ProjectTeam that represents the team which manages the issue and the Project to which the Team belongs

componentID	INT	Foreign key. Identification of the entity component that represents the information of the component to which the requirement is associated

Table C16. RequirementsData Entity

10.2.17 Review

This entity represents the review information of the work products that are associated to a component. Every review is performed by a team and a team can perform one or more reviews. One work product can be reviewed one or more times.

The fields are described in table C17:

Name	*Data Type*	*Description*
reviewId	INT	Primary key. Id of the entity Review
historyDate	DATETIME	Date when the information of the review was stored. It is used to keep historical data
initiationDate	DATETIME	Date the tasks for review started
kickOffDate	DATETIME	Date the review activity started
loggingMeetingDate	DATETIME	Date for meeting in the process of review
closureDate	DATETIME	Date the review finished
Pool	VARCHAR(100)	From which resource pool the moderatos of the review is coming
workProductTitle	VARCHAR(100)	Title of the document under inspection / review
activityType	VARCHAR(100)	The type of activity involved in each review
nOffParticipants	INT	Number of persons executing the review

entryEffort	FLOAT	Effort spent on entry phase
kickOffEffort	FLOAT	Estimated effort spent on start activities
preparationEffort	FLOAT	Estimated preparation effort spent on this review, reading the documents and preparing a list of mistakes.
meetingEffort	FLOAT	Estimated effort in the review meeting
reworkEffort	FLOAT	Estimated Rework effort
verificationEffort	FLOAT	Estimated effort for the review of the rework made
reviewSize	INT	Number of logical pages or lines of code (LOC) that the review has.
majorDefects	INT	The most important defects that must be solved in the review
minorDefects	INT	The least important defects that must be solved in the review
reviewtype	VARCHAR(100)	Explain the needs of the review
severity	VARCHAR(100)	Level of severity of the review
externalWorkProduct	INT	Indicates whether the product being reviewed is internal or external
state	VARCHAR(100)	Outcome of the review process
unit	VARCHAR(100)	Unit of measurement lines or pages
leadTime	FLOAT	Time it took to review and correct a document
moderator	VARCHAR(100)	Name of the moderator of the

		review
targetDateVerification	DATETIME	Date scheduled for the verification of the rework
targetDateRework	DATETIME	Date scheduled for the rework
totalEffort	FLOAT	Estimated total effort spent on the review
preparationRate	FLOAT	Average effort per page spent on preparation
removalRate	FLOAT	Average Defects removed per page
averageSize	FLOAT	Review Size / Number of participants
defectCost	FLOAT	Total cost of review / major defects solved
defectId	INT	Id of defect reviewed
saneID	INT	Outcome of sanity checks
saneCD	INT	Outcome of sanity checks
saneLT	INT	Outcome of sanity checks
saneNP	INT	Outcome of sanity checks
saneTE	INT	Outcome of sanity checks
sanePE	INT	Outcome of sanity checks
saneTD	INT	Outcome of sanity checks
saneSZ	INT	Outcome of sanity checks
saneDC	INT	Outcome of sanity checks
sane	INT	Outcome of sanity checks
recent	INT	Outcome of sanity checks – whether data element is in the

		expected range
componentID	INT	Foreign key. Id of the entity component that represents the information of the component for which the review is being done
proTeamId	INT	Foreign key. Id of the entity ProjectTeam that represents the team which manages the issue and the Project to which the Team belongs

Table C17. Review Entity

10.2.18 Test

This entity represents the information about the tests that are performed to a component. Every test is executed by a team, and one team can execute one or more test. For every component one or more test can be executed. One or more tests can be used to test a requirement, and one or more tests are associated to a test case.

The fields are described in table C18:

Name	***Data Type***	***Description***
testDataId	INT	Primary key. Id of the entity TestData
historyDate	DATETIME	Date when the information of the test was stored. It is used to keep historical data
testId	VARCHAR(100)	Id of the test as it was obtained from the original database of the company
testType	VARCHAR(100)	Type of the test
testState	VARCHAR(100)	Main state of the test
testStatus	VARCHAR(100)	Status is the sub state of the main state

testReview	VARCHAR(100)	Status of the revision of the test
testExec	VARCHAR(100)	Has the test been executed
linkedSteps	INT	Reference to the steps that make up the test case
failedRuns	INT	Number of times the test has been executed and failed
passedRuns	INT	Number of times the test has been executed and passed
testStart	DATETIME	Date the test was created
testPreparation	DATETIME	Date the test became prepared
testExecution	DATETIME	Date the test was executed
testFailed	DATETIME	Date the test failed
testPassed	DATETIME	Date the test passed
proTeamId	INT	Foreign key. Id of the entity ProjectTeam that represents the team which manages the issue and the Project to which the Team belongs
componentID	INT	Foreign key. Id of the entity component that represents the information of the component for which the test is being done

Table C18. Test Entity

10.2.19 Defect

This entity represents the defect information related to a component, which is managed by a team. Every component can have one or more defects. Every team can manage one or more defects.

The fields are described in table C19:

Name	*Data Type*	*Description*
defectId	INT	Primary key. Id of the entity Defect
historyDate	DATETIME	Date when the information of the defect was stored. It is used to keep historical data
defectType	VARCHAR(100)	Description of the defect
defectState	VARCHAR(100)	Main state of the defect
defectStatus	VARCHAR(100)	Status is a sub state of the main state
defectEstimate	VARCHAR(100)	Estimated cost to repair the defect
defectCost	VARCHAR(100)	Actual cost to repair the defect
severity	VARCHAR(100)	Severity of the defect
priority	VARCHAR(100)	Priority given to the defect for its treatment
injected	VARCHAR(100)	In which phase the defects has been caused
detected	DATETIME	When was the defect detected
defectStart	DATETIME	Creation of the defect
defectAnalysis	DATETIME	Date when the analysis of the defect started
defectResolution	DATETIME	Date when the defect entered to resolution
defectEvaluation	DATETIME	Date when the defect entered to the evaluation process
defectFinish	DATETIME	Date when the state of the defect was set to closed or to rejected
problemNumber	INT	Identification of the defect as it was originally obtained from the

		database of the company
version	VARCHAR(100)	Version of the defect
release	VARCHAR(100)	Release Number
requestType	VARCHAR(100)	Explain the needs of the resolution of the defect
crStatus	VARCHAR(100)	Current state of the defect in the resolution process
actTotalEffort	FLOAT	Estimated total effort spent on solving the defect
createTime	DATETIME	Creation of the record
submittedTime	DATETIME	Date of submission of the defect. State set to submitted
inAnalysisTime	DATETIME	Date when the analysis started
analysedTime	DATETIME	Date when the analysis ended
inResolutionTime	DATETIME	Date when the defect entered to resolution
resolvedTime	DATETIME	Date when the resolution ended
inEvaluationTime	DATETIME	Date when the defect entered to the evaluation process
evaluatedTime	DATETIME	Date when the evaluation ended
modifyTime	DATETIME	Latest change date of the defect's status. When it is closed or closured
modifiableIn	VARCHAR(100)	Name of the subsystem (local database of the responsible party) where the changes will be carried out
discoveredOn	VARCHAR(100)	The project = MTR-A
defectRestart	DATETIME	Date when the defect was

		restarted
defectOpen	DATETIME	Date when the defect was opened
defectReopen	DATETIME	Date when the defect was reopened
proTeamId	INT	Foreign key. Id of the entity ProjectTeam that represents the team which manages the issue and the Project to which the Team belongs
componentId	INT	Foreign key. Id of the entity component that represents the information of the component for which the defect is managed

Table C19. Defect Entity

10.2.20 CaseData

This entity represents the test cases associated to a component. Every test case is managed by a team and one team can manage one or more test cases. Every component can have one or more test cases associated. A test case can be associated to one or more tests.

The fields are described in table C20:

Name	*Data Type*	*Description*
caseDataId	INT	Primary key. Identification of the entity CaseData
historyDate	DATETIME	Date when the information of the use case was stored. This is used to keep historical data
caseId	INT	Id of the use case as it was retrieved from the original database in the company
caseState	VARCHAR(100)	Main state of the test case

caseStatus	VARCHAR(100)	Status is a sub state of the main state
caseStart	DATETIME	Date when the execution of the test case started
caseFinish	DATETIME	Date when the execution of the test case finished
LinkedTests	VARCHAR(2000)	Test that are related to this test case
proTeamId	INT	Foreign key. Id of the entity ProjectTeam that represents the team which manages the issue and the Project to which the Team belongs
componentID	INT	Foreign key. Id of the entity component that represents the information of the component for which the test case is being done

Table C20. CaseData Entity

11 Tests results

11.1.1 Change duration

Figure D1 shows the results obtained for change duration calculated for different dates. It can be observed that the time spent on change management for program with id 8, varies in a range between 50 to 170 days, but in the month of June it reached an elevated value of almost 380 days.

In order to understand the reason of that long duration, it would be necessary to answer new questions. For example, whether the number of changes that were closed in that moment was higher than the number of changes closed in the rest of the months; moreover, whether the number of people on the team in charge to manage changes for program with id 8 was less in that month. New queries could answer these questions.

Other way to obtain a response about the detected behavior would be to consult the people who managed the project, to know whether the team in charge of change management received a training to improve their skills in that area. That would help to understand whether the reason for the high duration was the learning speed of the people, combined with other factors such as the number of the risks or their severity.

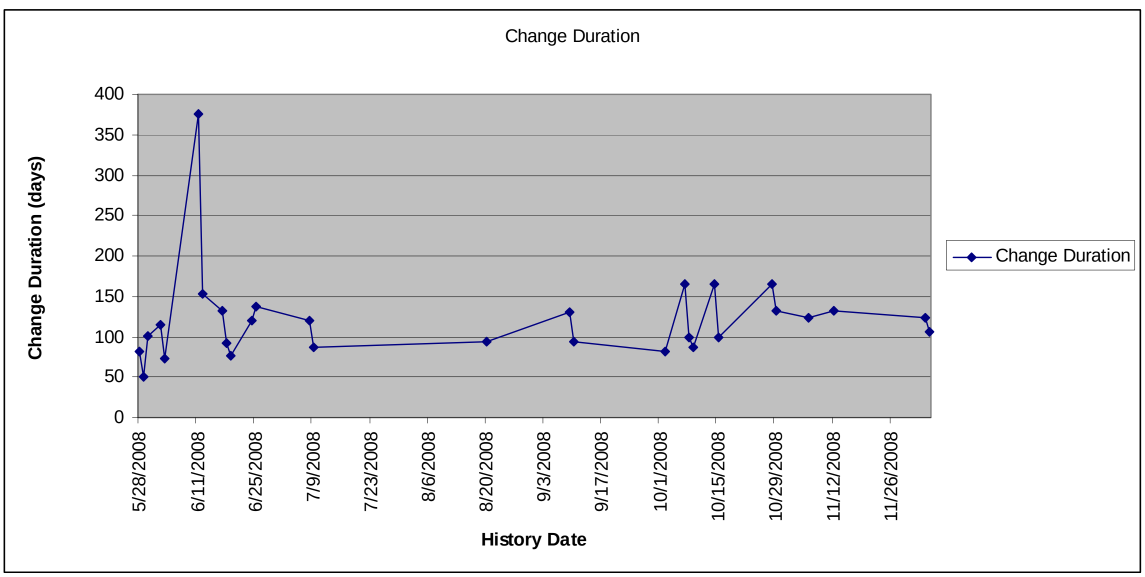

Figure D1. Change Duration

11.1.2 Compliance

Figure D2 shows for program with id 2, the percentage of compliance with requirements for different dates. It can be seen that through time the percentage of compliance varies in a range between 0% and 50%.

Since in different occasions it happened that the percentage of compliance with the requirements was zero, it would be appropriate to create new queries in order to figure out the reason; for example whether the severity of the requirements was too high, and thus in the moment the information was stored none of them was complete. It would also be necessary to check whether the information about the requirements was complete for those dates, and in case it was not, consult with the business experts the reason why the information was not stored.

In other cases it can be seen that the compliance percentage was always maintained in a range from 25 to 35, with a unique case when it was more than 45 percent. In order to understand the reason for this different behavior it would be necessary to consult in the database the severity of the requirements. It would be also helpful to ask the team in charge of requirements management, how the activities for implementation of the requirements were scheduled in order to achieve the discovered almost constant level of compliance.

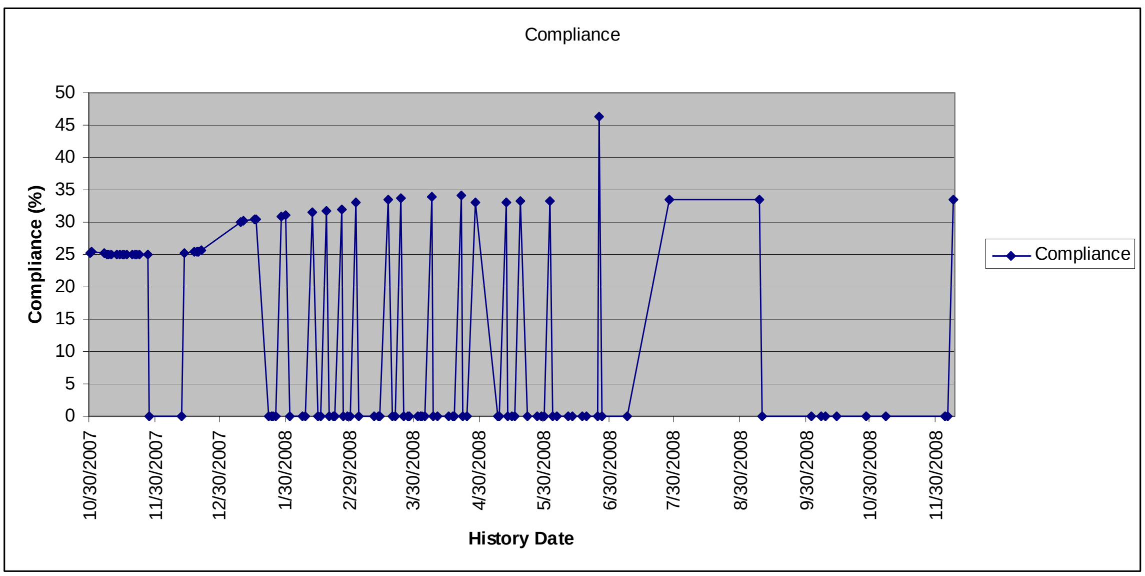

Figure D2. Compliance

11.1.3 Defect Severity

Figure D3 presents the results for the calculation of defect severity for program with id 2, in different dates. It can be seen that the level of severe defects is almost constant, with a value of 25% approximately.

As in the previous case, there are dates when the number of severe defects was zero. It would be useful to consult with the people who stored the data whether the information about defects was not available and thus not stored in those moments. Also it would be useful to make new queries in the database to figure out whether the moments when zero severe defects exists, are moments when maybe not critical components were tested.

Other question that could be answered to understand the reason of the percentage of severe defects, would be to investigate from the data in the database which is the effort the team spent on the tasks for solving defects; this would be useful to know whether the dates when the percentage was zero could be also when the team was more productive.

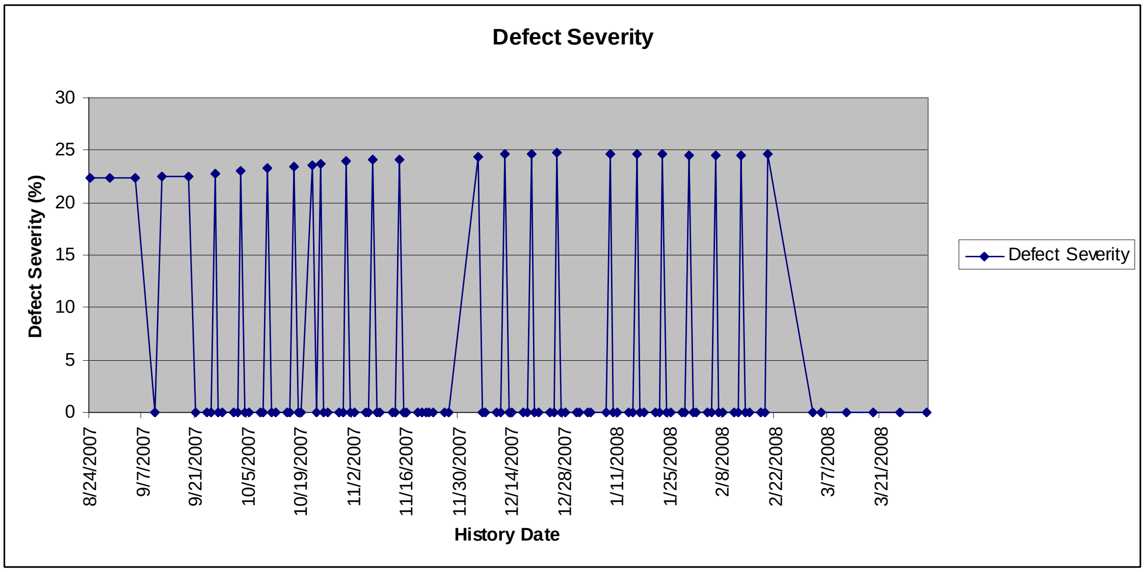

Figure D3. Defect Severity

11.1.4 Effort Distribution

Figure D4 depicts the effort distribution over 3 types of tasks related to program with id 8 through time. It can be seen that the higher effort is always put on tasks oriented to testing, and in second place to tasks oriented to requirements management. In third place, the tasks oriented to design are invested less effort.

The results obtained for the activities related to this program show a behavior that is not logical, since the effort remains constant in time; this would be an unusual case in the context of a project. Therefore the effort distribution metric was calculated for other programs, and it was discovered that the constant value also existed in the related data.

Nevertheless, for other programs such as program with id 30, the pattern changed and more variable values were found for the effort distribution metric. Figure D5 shows the effort distribution for the activities requirements, design and tests associated to program with Id 30.

Given that it was found that the constant pattern is not related to all the programs, it would be appropriate to make new queries that allow checking the values of actual work for every month. It seems that the same values were recorded every time data was stored. The people in charge to collect these data in the business context should be consulted to figure out whether no new data was available and therefore the database was not updated.

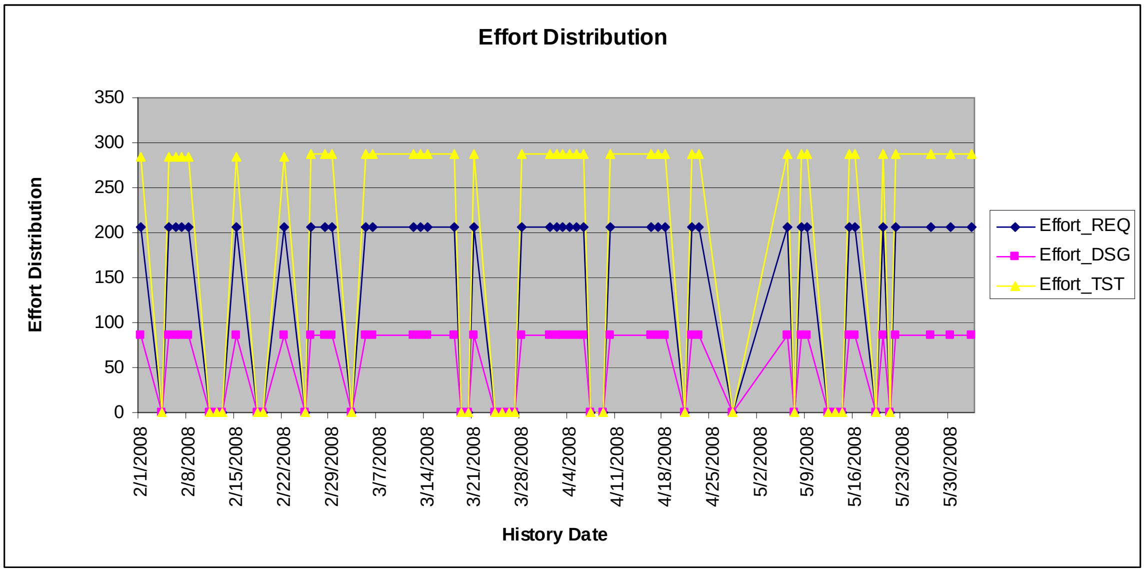

Figure D4. Effort distribution program id 8

Regarding figure D5, it can be observed that the activities that required more effort are the related with design, followed by those related with requirements, and then in third place, those related with testing. Nevertheless at the end of the measurement periods, it can be noticed that the effort spent on testing activities increased in a significant amount, while the other two remained almost constant.

It would be appropriate to investigate whether the planning of the projects related to program with id 30 was aiming to dedicate more time to requirements management and design activities in order to perform them more effectively. The people in charge of management of the projects could inform whether this kind of politic was applied at the beginning of the projects, with the goal to minimize the number of possible errors and therefore the time spent on testing activities.

It would be also appropriate to create new queries to figure out whether at the beginning of the measurements, the components developed didn't have a high complexity and therefore the time spent in testing was not too much. Also whether the complexity of the components increased and therefore the testing activities required more effort. This information could be obtained checking the severity of the tasks involved in the analyzed activities.

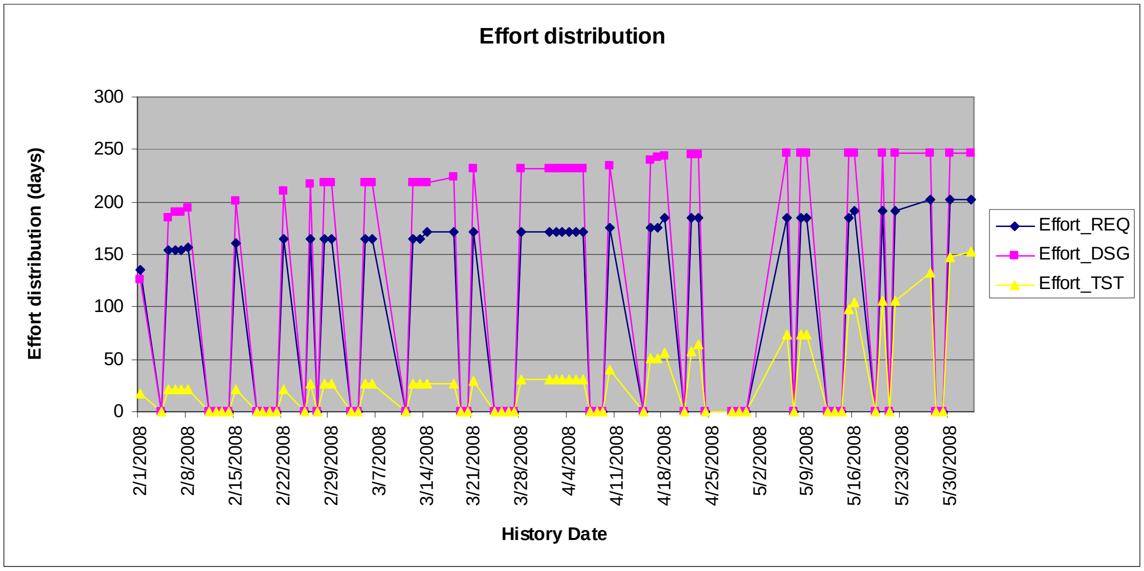

Figure D5. Effort distribution program id 30

11.1.5 Review Coverage

Figure D6 depicts the result of the percentage of review coverage for the project with id 49 through time. It can be seen that there is a high percentage of reviews that were finished every time the snapshots were taken.

It would be appropriate to generate more information about the review tasks for which information was stored in the dates presented in the figure. This would be useful to compare the amount of effort spent on every one of them. Also, to check whether the effort spent varied according to the size of the reviews being performed. A manager in charge of the process management could explain whether the planning of the review activities was made regarding their severity and size, in order to achieve that the percentage of coverage remained high.

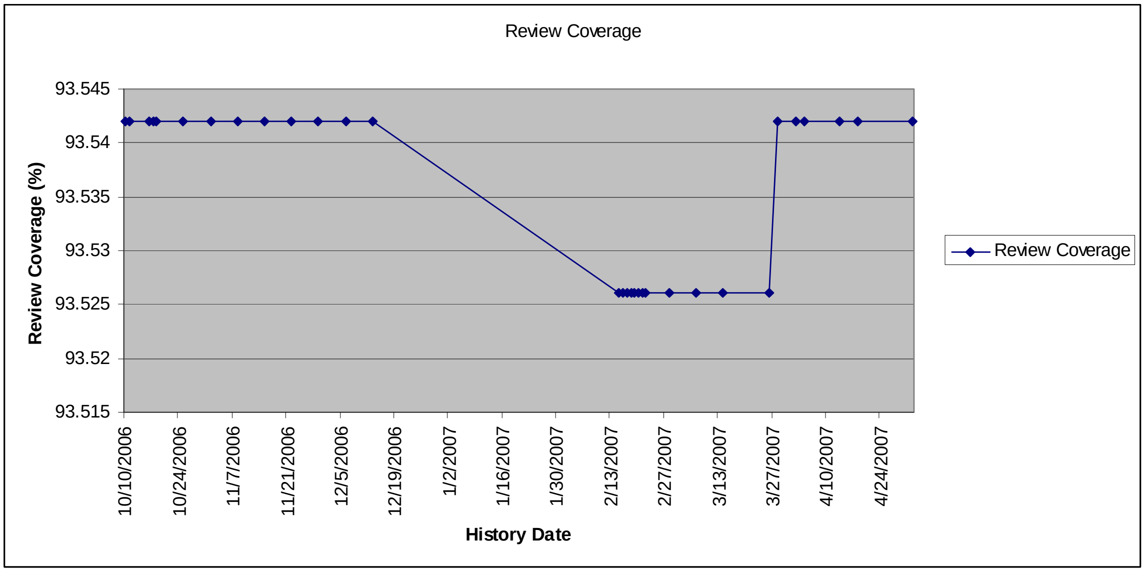

Figure D6. Review Coverage

11.1.6 Actual Cost of the Work Performed

Figure D7 shows the results of calculations for cost of work performed for program with id 8 through time. It can be seen that at the beginning, the work invested on tasks of different types was almost constant, and after the month of November its value increased and acquired a new constant level.

As in the case of effort distribution, the constant cost of work performed seems to have an unusual behavior in projects. The metric was calculated again for program with id 30, and the results are presented in figure D8.

In the case of program with id 8, it would be appropriate to consult with the people in charge to store the information about the actual work, whether the data was available and thus updated every time it was stored. This would explain the very low variability of the results.

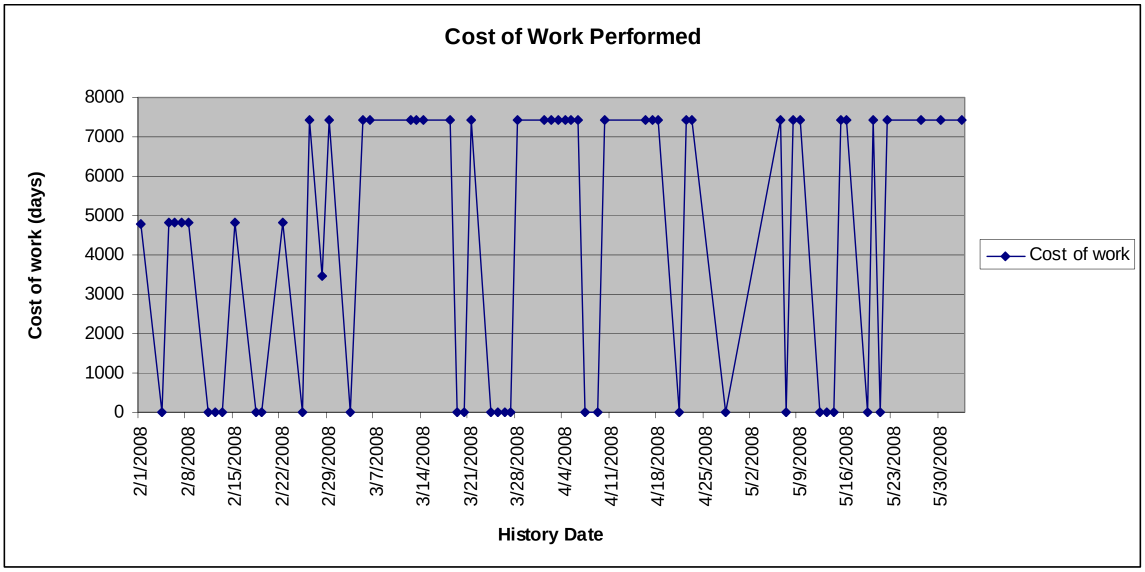

Figure D7. Cost of work performed program id 8

Regarding figure D8, it can be seen that there is variability in the cost of work performed, and in some months such as April, June and October, the difference with other months is considerable. The necessary queries to find out the severity of the activities and the availability of the team in charge of performing them every month, would help to explain why the high differences are present.

There are also several cases in which the value obtained is zero. The people in charge to store data about actual work spent on activities should be consulted in this case, with the aim to know whether the data was not properly stored for those dates.

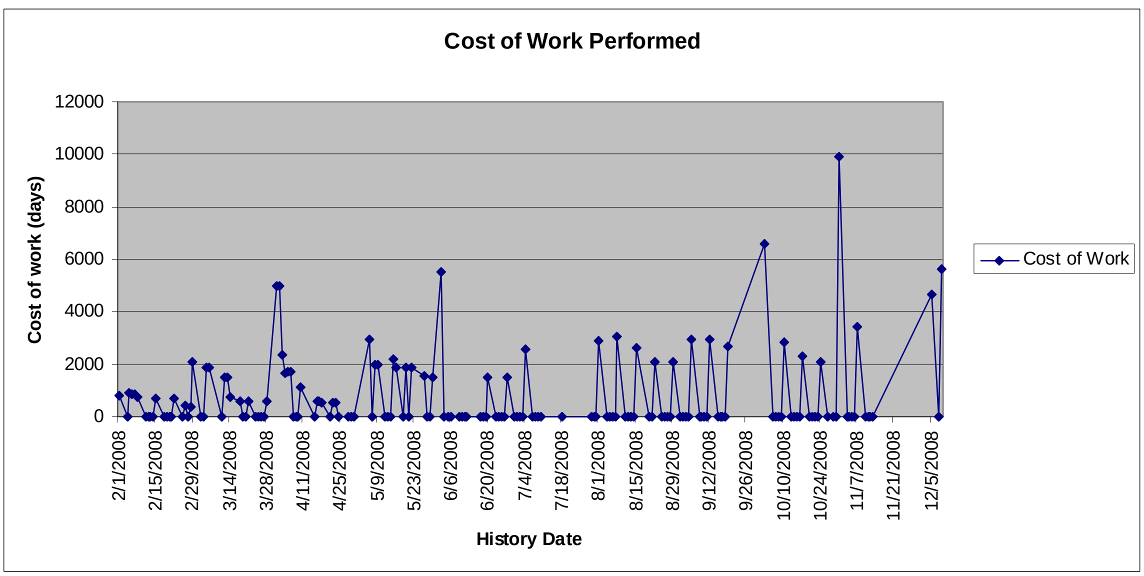

Figure D8. Cost of work performed program id 30

Appendix E

12 Appendix E: Work Methodology Process Model

The figure E1 presents one of the three main deliverables of this project, a process model that summarizes the work methodology that was proposed in chapter 3. This work methodology was followed during the development of the project in order to create the quality database. It is also intended to serve as a model that can be followed in the future using new information from companies provided by the consultant. In the figure the activities to be done, along with data and documents needed for these activities or produced by them are represented.

The main steps to be followed are:

1. Analyze the structure of the data in the snapshots.
2. Define quality dimensions to be measured during data cleaning.
3. Define metrics associated to the dimensions.
4. Define the attributes of the snapshots which will be assessed with the metrics during the cleaning.
5. Define information that is required from the business context to apply the metrics. This is because in some cases not all the metrics can be applied without having real world information that can be used to assess dimensions such as accuracy.
6. Define the definitive metrics to be applied based on the knowledge of the business context.
7. Perform measurements. In this activity algorithms can be developed, but also some existing techniques or tools could be found that can be used according to the dimensions that are being measured. In this project, algorithms were developed.
8. Define a standard quality level to be used as reference to establish whether the quality level obtained during measurements is good enough.
9. Analyze the results obtained during the measurements by comparing them with the standard quality level defined. Establish which data should be improved.
10. Obtain information necessary to perform improvements from the business context. In case needed data is not available no improvements are possible to be done. In this project the possible improvements were performed, and in cases where the information was not available, it is explained why improvements were not made.

11. Perform possible improvements. Once this is finished the data cleaning phase has finalized, and the quality of the data has been improved.
12. Create Data Analysis model to define classes that will represent objects of the real world which information will be stored in the database.
13. Define whether new quality dimensions must be added to the information to be stored in the database. In case new dimensions are defined, a Quality Analysis model must be created complementing the Data Analysis model to indicate which are the dimensions and the data for which the dimensions will be added. In this project no additional dimensions were necessary, so there is not a Quality Analysis model.
14. Create the Entity Relationship model which represents the design of the database. This model must be based on the Data Analysis model and the Quality Analysis model (in case this exists). This is to define all the necessary entities to represent the objects of the real world and the entities or attributes that are necessary to add new quality dimensions. The model must be readable, correct and normalized in order to give it interpretability.
15. Define the DBMS that will be used to create the database.
16. Create the database and store there data from the snapshots. This is the second of the three main deliverables of the project.
17. Create documentation of the database taking into account the interpretability characteristics explained in chapter 3. This is the third of the three main deliverable of this project, and can be found in appendix C.
18. Define tests to be done in order to prove that the information of the database can be used.
19. Perform the test and document the results.

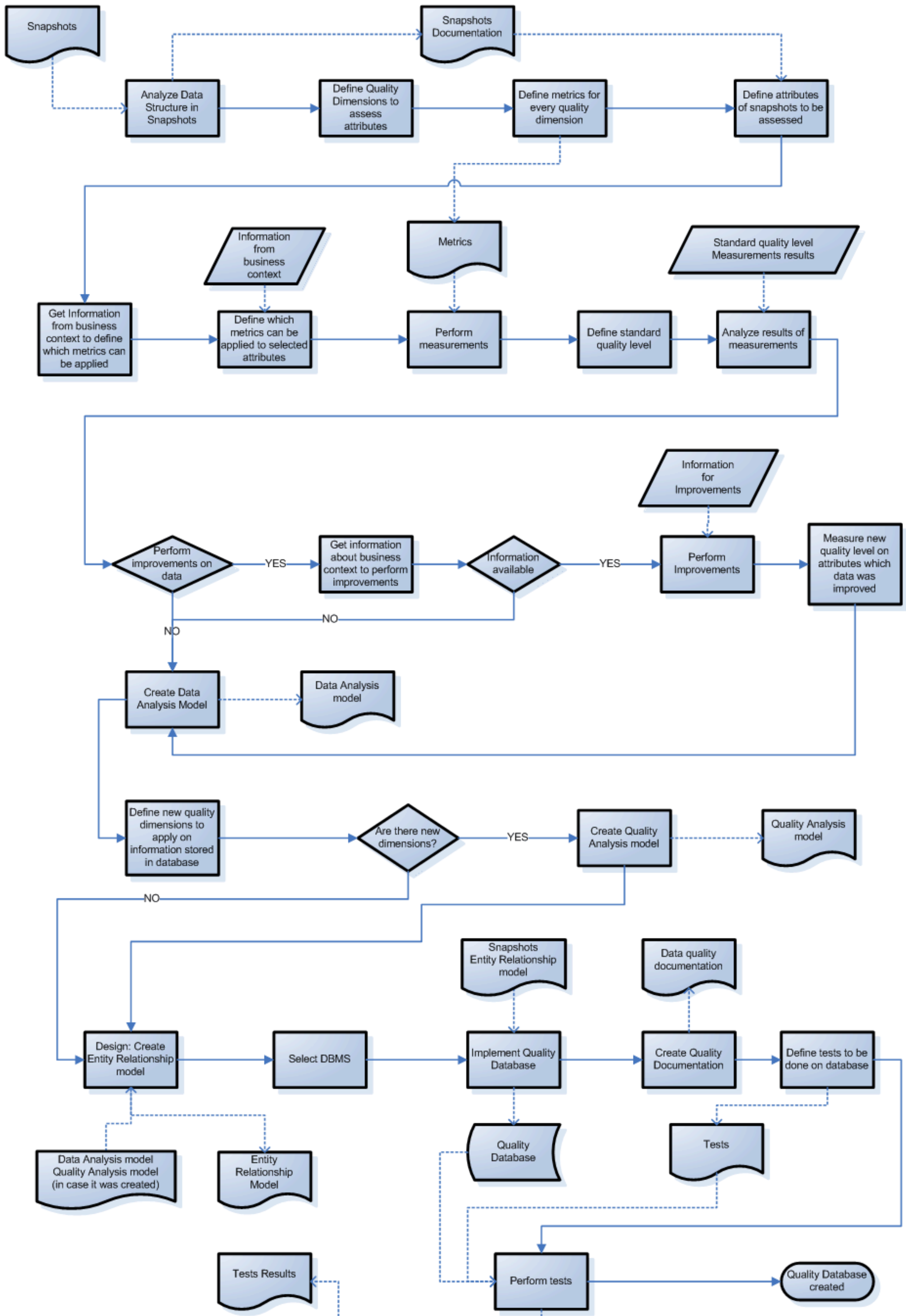

Figure E1. Work Methodology Process Model

The quality database, as one of the deliverables of the project, has already been created and data, which quality was measured and improved, was stored there. Therefore the process model can be used as reference to work with new data provided in the future, and only the steps that are necessary must be followed. For example, if new data is proportioned, only the measurement and improvement of the quality must be done, to then store the data in the database. If new quality dimensions are necessary, the current Data Analysis model must be complemented with a new Quality Analysis model where these dimensions are represented. Afterwards, the design of the data model can be improved adding the entities or attributes that are considered necessary to add new quality dimensions to the information stored there; finally, also the consequent modifications to the database can be done.

References

[MYS09] http://dev.mysql.com/doc/refman/5.1/en/index.html

[SIE08] Siemons P., "Measurement System Design", Pages 14, 16, 17, 21. Euroclear 2008.

[URE08] Ureña E., "Process Mining applied to the Change Control Board Process - Discovering Real Processes in Software Development Process". Page 27. Series Master Theses Operations Management and Logistics. TUE Department Technology Management. Eindhoven University of Technology. Eindhoven, The Netherlands. 2008.

[IBE08] Ibern A. and Morató R. "Statistical Analysis of defect data in software development". Pages 9, 12. Eindhoven University of Technology, Faculty of Technology Management, Division of Information Systems. Eindhoven, The Netherlands. 2008.

[SIE02-1] Siemons P., "Measurement database". July 2002. Page 8.

[SIE02-2] Siemons P., "Measurement Plan Philips". December 2002. Page 13.

[BAT06] Batini C., Scannapieca M. "Methodologies for Data Quality Measurement and Improvement", chapters 2, 3, 4, 5, 7. ISBN 978-3-540-33172-8. Book Series, Data-Centric Systems and Applications. Publisher Springer Berlin Heidelberg. DOI 10.1007/3-540-33173-5. Copyright 2006.

[SIE03] Siemons P. "Design description for measurement database". 2003.

[SIE04] Siemons P. "Software Measurement Guidebook". Software Engineering Methodologies for Embedded Systems. 2004.

[WAN98] Wang R. "A product perspective on Total Data Quality Management". Communications of the ACM. February 1998/Vol. 41, No. 2.

[WAN93] Wang R.Y., Kon H.B., Madnick, S.E. "Data quality requirements analysis and modeling". Sloan School. of Management, MIT, Cambridge, MA. Proceedings Ninth International Conference on Data Engineering, 1993. 19-23 Apr 1993. On pages: 670-677.

[PIP02] Pipino L., Lee Y., Wang R. "Data Quality Assessment". 2002, ACM. Communications of the ACM. April 2002/ Vol 45, No. 4ve.

[SCA02] Scannapieco M., Pernici B., Pierce E. "IP-UML: Towards a methodology for quality improvement based on the IP-MAP framework". Proceedings of the Seventh International Conference on Information Quality. (ICIQ-02).

[Fel69] Fellegi I., Sunter A. "A Theory for Record Linkage". Journal of the American Statistical Association, Vol. 64, No. 328 (Dec., 1969), pp. 1183- 1210. Published by: American Statistical Association

[WAN95] Wang, R.Y., Storey, V.C. Firth, C.P. "A framework for analysis of data quality research", IEEE Transactions on Knowledge and Data Engineering. On page(s): 623-640, Volume: 7, Issue: 4, Aug 1995

[WAR96] Ward R., Wand Y. "Anchoring data quality dimensions in ontological foundations" Communications of the ACM archive. Volume 39, Issue 11 (November 1996) Pages: 86 - 95. ACM New York, NY, USA

[STR97] Strong D., Lee Y., Wang R. "Data quality in context". Communications of the ACM archive. Volume 40, Issue 5 (May 1997). Pages: 103 – 110. ACM New York, NY, USA

[BER07] Berti-´Equille L., "Measuring and Modelling Data Quality for Quality-Awareness in Data Mining", Studies in Computational Intelligence (SCI) 43, 101–126 (2007). IRISA, Campus Universitaire de Beaulieu, Rennes, France - Springer-Verlag Berlin Heidelberg 2007

[BER07-1] Berti L., "Data Quality awareness: a case study for cost-optimal association rule mining". Knowledge and Information Systems (2007) 11(2) : 191 – 215. Springer – Verlag London Limited 2006.

[BAL99] Ballou D., Kumar G. "Enhancing data quality in Data Warehouse environments". Communications of the ACM. January 1999/ Vol. 42, No. 1.

[GAR98] Gardner S., "Building the Data Warehouse". Communications of the ACM. September 1998/ Vol. 41, No. 9. pages 52 – 61.

[VAS02] Vassiliadis P., Simitsis A., Skiadopouls S. "Conceptual Modeling for ETL Processes". National Technical University of Athens, Dep. Of Electrical and Computer Eng. November 8, 2002, McLean, Virginia, USA. ACM Comunnications.

[ANN06] Annoni E., Ravat F., Teste O., and Zurfluh G. "Towards Multidimensional Requirement Design". IRIT-SIG Institute. University of Paul Sabatier. DaWaK 2006, LNCS 4081, pp. 75–84, 2006. Springer-Verlag Berlin Heidelberg

[MAZ08] Muñoz L., Mazón J., Pardillo J., Trujullo J. "Modeling ETL Processes of Data Warehouses with UML Activity Diagrams". OTM 2008 Workshops, LNCS 5333, pp. 44–53. Springer-Verlag, Berlin Heidelberg 2008.

[JAR03] Jarke M., Lenzerini M., Vassiliou Y., Vassiliadis P. "Fundamentals of Data Warehouses" 2nd Revised and Extended Edition. Springer-Verlag, 2003. 214 pages, ISBN: 3-540-42089-4

[WIN06] Winkler W. "Data Quality: Automatic Edit/Imputation and Record Linkage". European Conference on Quality in Survey Statistics. Proceedings of Q2006.

[BAL98] Ballou D., Wang R., Pazer H., Tayi G.K, "Modeling Information Manufacturing Systems to Determine Information Product Quality", Management Science, 44(4), 1998.

[STE06] Steven P., Pooley R. "Using UML: Software Engineering with Objects and Components". Second Edition. Addison Wesley. Object Technology Series. Series Editors. 2006

[SIE09] Siemons P., "Collecting metrics". Metrific Management Consult. 2009.

[SIE07] Siemons P', "Measurement and analysis Plan". 2007

.