The metrics for Complexity to Quality of a system by UML

Shahid Nazir Bhatti and Gerhard Chroust

Faculty of Systems Engineering and Automation, Johannes Kepler University Linz, A - 4040 Linz, Austria
snb@sea.uni-linz.ac.at, gc@sea.uni-linz.ac.at

ABSTRACT

The aspects of quality are that it is something unquantifiable trait- it can be discussed, felt and judged, but can not be weighted or measured. To validate software systems early in the development lifecycle is becoming crucial. Early validation of functional requirements is supported by well known approaches, while the validation of non-functional requirements, such as complexity or reliability, is not. Early assessment of non-functional requirements can be facilitated by automated transformation of software models into (mathematical) notations suitable for validation. These types of validation approaches are usually as -transparent to the developers as possible. The widely acceptance of quality services will only be accepted by users if their quality is of the most acceptable level. UML is rapidly becoming a standard (both in development and in research environments) for software development. The work here in this paper is extension of Quality with UML (QWUML, IDIMT-2004, and SEN-2005), quality of the system measurements with modeling (UML). This paper discusses some important issues regarding system design modeling in association with quality, complexity, and design aspects using UML heuristics.

Keywords: *UML Unified Modeling Language, QWUML Quality with UML, DB Database, KAS Number of key attributes, DRC Depth of Relationship between classes, IRA Inter-relational attributes, IRM and Inter-relational methods.*

1. INTRODUCTION

The increasing demand for software and its proliferation, bringing it into contact with more and more people creates a major demand for high quality software and as a consequence for skilled software engineers, managers and quality specialists and appropriate tools and methods. Quality is gaining more and more importance in the software world, especially in view of quality software's development with international standards [13]. The application of software metrics has proven to be an effective technique for improving the software quality and productivity. The some of aspects about quality, software metrics is discussed in detail in section 2 and section 3. The term 'quality' is used internationally to describe a comprehensive process which ensures and demonstrates the quality of the products and services it produces because quality is a journey which has milestones rather than a destination. It has to be noticed that software organizations invest some how 80% their development resources related to their 'products quality'1 [7].

UML is widely accepted as the standard for representing the various software artifacts generated a software development processes. Visual modeling through UML provides effective traits, on one hand generating a suitable Analysis/ Design graphical view for the system development, beside generating suitable code for different main technologies (like, Oracle, Visual c, Visual Basic, Java etc). The important aspect is while working quality with UML. The non functionality requirements like complexity to quality addressed with graphical representation, i.e. using UML to the system design. We have used schema mapping to effectively display the effect of complexity in the interrelated UML diagrams to relational database report in section 4. In UML we have used here use case diagram and class diagram for modeling the system design aspects to database (Oracle 8i) and vice versa (reverse engineering aspects i.e. Class Diagrams generation from Oracle code). Further illustrating: -the effect of complexity with UML diagrams and data and information redundancy with respect to UML as well as database in Section 4 of this paper.

2. SOFTWARE QUALITY ISSUES

2.1. QUALITY

'The totality of features and characteristics of a product or service to that bear on its ability to meet stated or implied requirements'. Quality is critical for survival and success, the market for software is increasingly a global one and no organization will succeed in that market unless they not produces quality products and services. If any one does not do so then that organization may not even survive [8].

2.2. WHY SYSTEM FAILS

When a system fails, the failure may be the result of any of several reasons as indicated in the figure: 1, and can be accumulated in the following manners:

- *The specification may be wrong or have missing requirements. The specification may not state exactly what the customer wants or needs.*
- *The specifications may contain a requirement that is too complicated to implement, given the prescribed hardware and software scenarios.*

[1] Some detail material about Quality, Software Quality, Quality with ISO/ IEC Standards-Product Quality aspects etc, [ACM-SEN Mar - 2005 Shahid Nazir Bhatti].

- *The system design may contain a fault. Perhaps the database and query-language designs make it impossible to authorize users.*
- *The program design may contain a fault. The component descriptions may contain as access control algorithm that does not handle this case correctly.*
- *The program code may be wrong. It may implement the algorithm improperly or incompletely.*

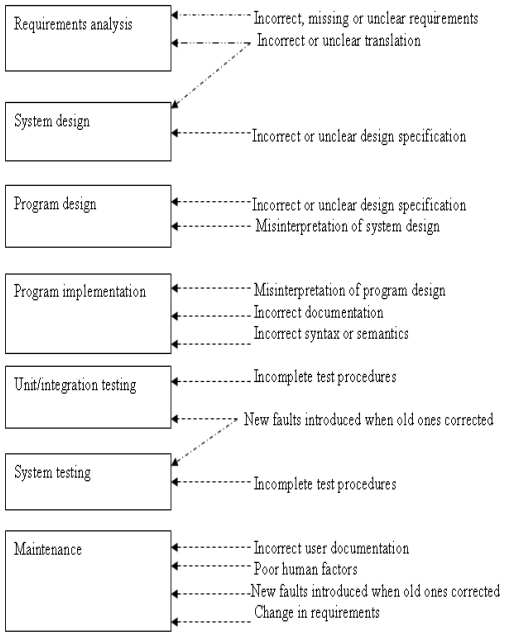

Figure 1: Causes of faults during development [10].

Faults can be inserted into a requirement, design, or code component, or in the documentation, at any point during maintenance. Figure 1 illustrates the likely causes of faults in each development activity. Although we would like to find and correct faults as early as possible, system testing acknowledges that faults may still be present after integration testing.

A 'baseline audit' can be carried out to measure current practice against the requirements of the ISO/IEC 9126 standards, the audit in fact examines the organization's activities under various software quality metrics, software quality metrics framework defines a 'software quality metric' as a quantitative measure of an attribute that describe the quality of a software product or process. Test procedures should be enough to exercise system functions to every body's satisfaction: user, customer, and developer.

In previous work2, detail material available about quality toward software product (ISO/ IEC 9126), term 'Design Quality' and also issues regarding 'Quality with UML i.e. QWUML'. Importantly this work is related to ISO/ IEC 9126 and 25000. With the graphical representation of the characteristics and sub characteristics of these software metrics, also these aspects are more detail illustration to 'product quality'. As the characteristics from ISO/ IEC 25000 Functionality, Usability, Reliability, and Maintainability etc depicted in detail about the quality of the software end product but there is very little information about the issues related to the complexity about the software process or hence effects related to software products. In next section, we have highlighted some important issues regarding software complexity, software complexity toward the issues regarding system development.

2.3. SOFTWARE COMPLEXITY TO SOFTWARE QUALITY

"software engineering is the field of computer science that deals with the building of software systems that are so large or so complex that they are build by a team or teams of engineers" [10]. With ISO-Standards there is very little information available about the term 'complexity' with external metrics characteristics. Although, almost all the existing attributes and sub attributes of these external metrics are influenced either directly or indirectly by this key factor. Some of the classical complexity measures with internal as well as external complexity of modules are shown below in table 1:

Measured by	**Categories**
Halstead: [1977]	Internal-code-based
McCabe: [1976]	Internal-code-based
Shepperd: [1991]	External
Chidamber & Kemerer: [1993]	Method per class-Internal
Henry & Kafura: [1981]	Hybrid-code-based

Table 1: Some Classical Complexity Measures [13].

From table, hybrid category to complexity measure is important factor, as complexity measure of any module and segment is combination of both internal and external complexity. In the past, software was monolithic and procedural in nature e.g. a typical COBOL program of the past was a single entity with subroutines called as required; the logic of program was sequential and predictable.

The examples of the complex systems are control systems in process plant, medical electronics, aircraft controls, machine tool control, nuclear plant and weapon systems. The consequences of failure under these circumstances are often severe and thus attract particular attention. It leads to two more difficulties

- Due to the complexity of software failure modes the possibility of total failure (system) is greater
- Since the use of software makes fault difficult to predict it is difficult to perceive if the integrity of a system is adequate.

[2] Detail work about Quality with UML-QWUML , Functionality Metric working with UML available in [Shahid Nazir Bhatti, IDIMT-2004]

Modern systems are distributed (as it can reside on many computer nodes) and its execution is complex. The size of software is the sum of the sizes of its modules (components). Each component is designed to be of limited manageable size. As a result the size (lines of code) is not factor in the difficulty level is inflated by the fact that these (or any) modules are frequently developed and managed by people and teams not even known to each other [10]. This creates a network of intercommunicating objects. In relationship, dependencies, responsibilities, roles created within domains and randomly introduction of new domains and sharing of objects within different teams. The issue of software complexity can be enlightened as the fact that any object in one working domain (schema) can communicate with any object in another domain creates potential dependencies (relationship) between all objects in the current system.

3. SOFTWARE QUALITY METRICS

Metrics are generally defined as, "quantitative methods" and "they can be used to measure the periodic changes". *The application of software metrics has proven to be an effective technique for estimating, assessing and improving software quality and productivity* [9] i.e. the initiation of a software metrics program will provide assistance to assessing, monitoring and identifying improvement actions for achieving quality goals regarding ISO/IEC 9126 Standard [5]. Software metrics are of interest for several reasons [12]:

- *Quantitative measures can be used as indicators of a software product or development process. These indicators, such as size, product, quality, process quality etc, are of interest to software development mangers, developers, and users.*
- *The software metrics may indicate suggestions for improving the software development process.*

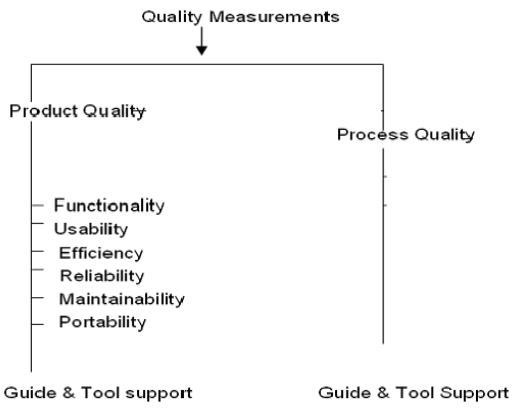

Figure 2: Software Quality Metrics Framework.

Regarding the requirements for software's quality issues, the software quality metrics framework introduces categories of metrics that extends through the phases of software development life cycle which are independent of methodologies. The framework is designed to address the wide range of quality characteristics for the software products and processes i.e. as shown in the figure: 2, the following frame work enables better description of software quality aspects and its importance [9].

3.1. UML HEURISTICS WITH SOFTWARE QUALITY METRICS

UML is used as the first characteristic to be subjected to software quality metrics, we may speak of. 'Quality with UML' (QWUML). The objective is to efficiently design and deploy the software systems that meet customers' requirements; the efficiency which can be measured using QWUML is in terms of cost, quality and lead time.

The dynamic view is depicted with the use cases, list of activities/interactions and the states and there a change by the sequence diagrams, finally the static view is depicted with the class diagram.

If we go with the details of the external metrics from the ISO/IEC 9126 which are listed below, these metrics are indicators that relate to high level size, product and development process quality indicators that are of interest to the software development and maintenance activity. When looking for the graphical support for these software quality metrics by virtue of UML, the requirements are categorized according to the FURPS+ model [9] a useful mnemonic. The requirements are categorized and the functional and non functional support is provided to these software metrics, as by graphical illustration of these software metrics by effective use of UML enhance their worth regarding the software quality, performance, and productivity using the ISO/IEC 9126. In the following comparison in the Table 2 given below we show the relation of the ISO/IEC 9126 external metrics and the UML support for these software metrics in this regard.

ISO/IEC 9126, External metrics	**UML heuristics for software metrics**
Functionality metrics	Functional
Reliability metrics	Reliability
Usability metrics	Usability
Efficiency metrics	Efficiency
Maintainability & Portability metrics	Supportability

Table 2: ISO/IEC 9126 External metrics and UML support for the metrics.

Working with the requirement categorization and functional and non functional design aspects of the software products, the following results can be achieved while working with the UML in regard with these ISO/IEC 9126 external metrics. Some requirements are called quality attributes [BCK98] of a system. These include usability, reliability, and so forth.

In next section 4, using these UML characteristics to different software metrics, we have considered a simple library scenario to issue books to university employees as shown in figure 3. Further the logical view of this library issue book domain, class diagram as figure 4, showing the different aspects and relationing factors to complexity within classes.

4. UML CHARACTERISTICS TO SYSTEM DESIGN

The application of software metrics has proven to be an effective technique for estimating, assessing and improving software quality and productivity [12]. UML is used as the first characteristic to be subjected to software design quality, due to enrich & standardized graphical modeling support. As early validation and verification of functional & non-functional requirements is becoming crucial in early stages of software development in context to software design quality.

Modeling is a useful working scenario carried out over the years in software development, modeling is abstraction and use case diagram in UML is one such kind of abstraction. Defining use case diagram makes it easier to break up a complex application (big bubble/ use case), into simple, discrete pieces that can be individually studied. For different UML heuristics to system design, we here consider first a simple library use case example to identify some domain requirements as shown in figure 3 below.

Although there are number of other use cases and further aspects in this library use case diagram as shown in figure 3, but here just few use cases are mainly addressed for one domain design to BOOK_ISSUE. In figure 3, the Employee (actor) of any department applies for issuing of a book, the book status is checked and if book is available, it is issued to desired employee with corresponding of another actor librarian. The data used for this research for the complexity indicators from UML is taken from '"HRM system for the hospital Enterprise in ISD, Pak". This is already a running project developed by Shahid Nazir Bhatti. Already the documentations are available for the Requirement analysis, Design and implementation parts of this project. We have used the different ASCII Editors to UML diagrams retrieve the data from UML diagrams and check the validity with respect to data we have from the enterprise's project [15].

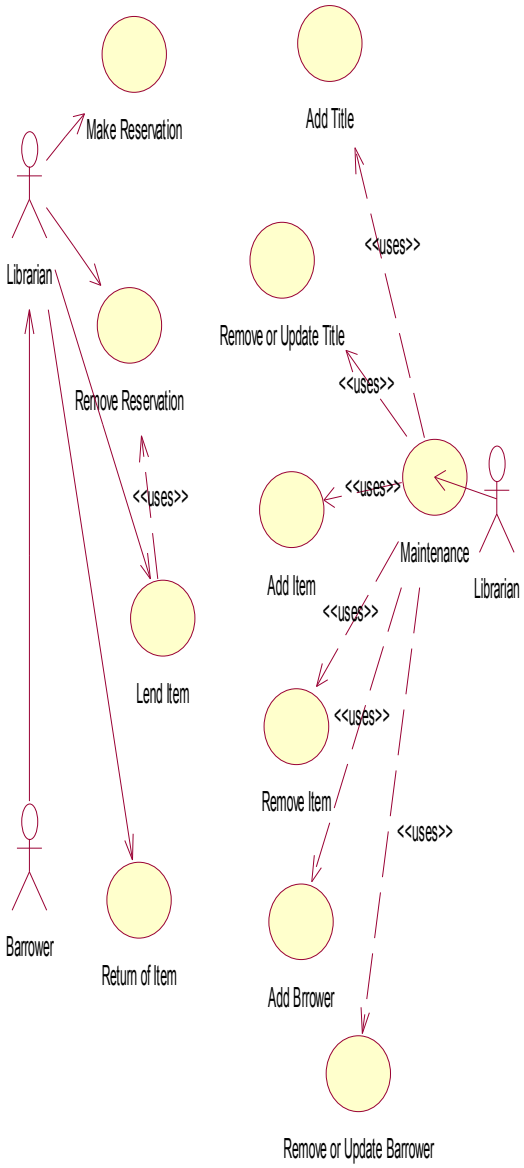

Figure 3: Use case diagram for current Library domain example.

The complexity indicators with respect to use case diagram (figure 3) consist of the following factors:

- Candidates' classes
- Major functionality modules
- Dependency relationship
- Object association

From the figure 3 the numerical values of these complexity indicators are shown in table 3 in detail. We have used the Rational Rose for UML diagrams and ASCII Editors and XML code for these facts.

Complexity Attributes	**Recognized by**	**Measurements**
Candidate Classes	Object Class	02 (figure 3)
Major functionality modules (use cases)	Object use case	11 (figure 3)
dependency between objects	stereotype	Uses (figure 3)
dependency between use cases	stereotype	Uses (figure 3)
(object role/ use case, initiator)	Supplier	02 (Lib. Barrow.)

Table 3: Internal code depiction from the use case diagram.

$C = (x)^{a} + b(x) + d$ (where $a \neq d$)[3]

(Here 'C' is for complexity, 'a' are the number of object classes, 'b' number of object use cases, and 'd' are actors).

Generating the "Logical View" i.e. Class diagram from this library use case diagram (figure 3) using Rational Rose for one of the domain ISSUE BOOK. The resulting Class diagram based on the library use case diagram is shown in figure 4. In different classes within class diagram beside other information are also description for set of objects that share the same specifications of features, constraints and semantics [4].

In Static structures of models, called also state models are expressed in Class a diagram, class diagram visualizes classes (and interfaces), their internal structure, and their relationships to other classes. The class diagram is one of the static models of the use case diagram and defines the further static aspects to the design of the system. The resulting Class diagram from the library use case for current BOOK_ISSUE segment, featuring the aspects of schema generation (key attributes), attributes, methods and relational integrity as shown in figure 4 below.

In figure 4, as can be seen from figure the stereo types of these different classes are 'Relational Table', as besides successfully generating the logical view from use case diagram. With predefined operations and the integrity constraints toward relational database (schemas), the schemas are generated (Rational Rose, UML to Oracle8 database).

[3] Work related to Software quality metrics attributes with UML diagrams use case diagram, sequence, deployment & activity diagrams etc is already accepted for publication in CFP-2006, Prague Czech.

In figure 4, a class diagram is shown with attributes and methods to simple library working, the names of classes in this class diagram are BOOK, BOOKCATEGORY and BOOK_ISSUE, EMPLOYEE, and DEPT as shown in figure 4. In this library working the class BOOK is the one diagram, used by the other classes for their relational functionality. In class BOOK, the attribute BOOKNO is a Key attribute used as Primary key and used as foreign key to the classes BOOKCATEGORY and BOOK_ISSUE. Further in class BOOKCATEGORY there are two key attributes, the BOOKCATNO as primary key and BOOKNO as foreign key. In BOOK_ISSUE there are again three key attributes but they are all foreign key attributes from classes BOOK, BOOKCATEGORY and EMPLOYE. Further with two classes EMPLOYEE and DEPT there are two key attributes EMPNO and DEPTNO respectively as can be shown in figure 4.

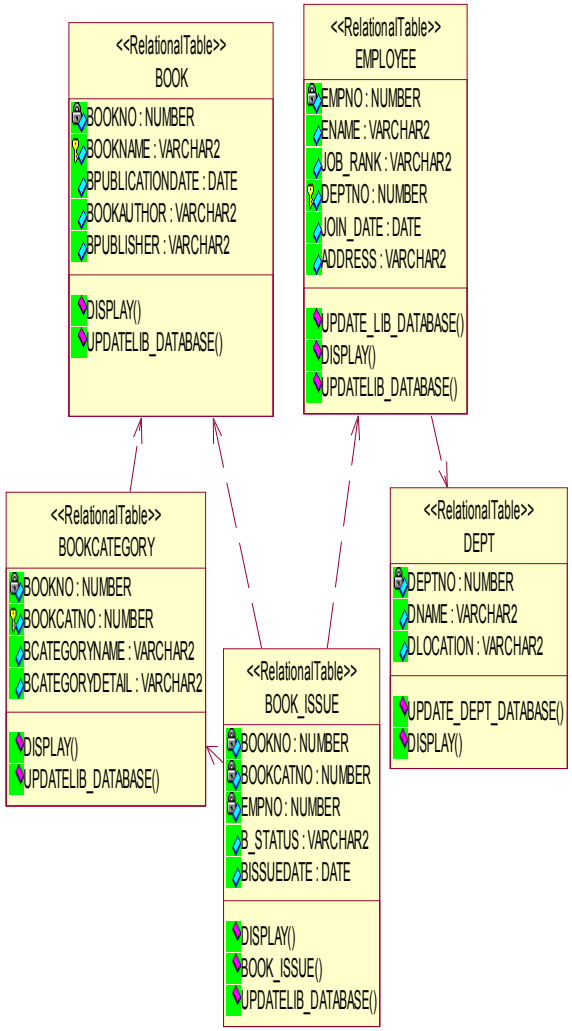

Figure 4: Class diagram to relational tables.

With the process of class mapping of the current class diagram to relational schemas (tables), with the process of UML Class diagram with Rational Rose to database server.

The successful relational schemas are generated for related database tables (oracle in this case). In figure 4, the 5 classes in class diagram i.e. BOOK, BOOKCATEGORY, BOOK_ISSUE, DEPT, and EMPLOYEE results here five independent (but relational) database tables are generated in oracle database server. Further, the relational classes and detail attributes to complexity of the class diagram are shown in relational-db report in next section. The hierarchal structure of these relational tables (e.g. Book Issue) can be viewed from Oracle (Describe Book Issue).

The physical code (to oracle database) illustration of these five classes in class diagram is shown in next section in figure 5 below. Further, figure 5 show the physical names of classes, attributes, dependencies (if any) of these classes.

4.1. RELATIONAL SCHEMA REPORT FROM THE CLASS DIAGRAM

The relational database report (in oracle) from class diagram in previous section is shown below, in figure 5.

Oracle 8 Database Model Report
Sorted by Name Sonntag, 28. Mai 2006
Schema Logical Name: BOOK_ISSUE
Physical Name: BOOK_ISSUE
==========
Class Name: BOOK_ISSUE StereoType:
<<RelationalTable>>
Physical Name: BOOK_ISSUE
Documentation: Details about BOOKS. Columns
Name: BOOKNO
Physical Name: BOOKNO NUMBER(,05)
NullsAllowed : True Name: BOOKCATNO
Physical Name: BOOKCATNO NUMBER(,05)
NullsAllowed : True
Name: EMPNO Physical Name: EMPNO NUMBER(,05)
NullsAllowed : True *Methods*
Name: DISPLAY Physical Name: DISPLAY()
Name: UPDATELIB_DATABASE Physical Name:
UPDATELIB_DATABASE()
Dependencies BOOK <<RelationalTable>>
BOOKCATEGORY <<RelationalTable>> Schema Logical
Name: BOOKCATEGORY
==========
C*lass Name: BOOKCATEGORY StereoType:*
<<RelationalTable>>
Physical Name: BOOKCATEGORY
Documentation: BOOK Categorization. *Columns*
Name: BOOKNO Physical Name: BOOKNO
NUMBER(,05)
Name: BOOKCATNO Physical Name: BOOKCATNO
NUMBER(,05)
NullsAllowed : False (Primary Key)
Methods Name: DISPLAY Physical Name:
DISPLAY()
Name: UPDATELIB_DATABASE Physical Name:
UPDATELIB_DATABASE()
Dependencies BOOK <<RelationalTable>>

Schema Logical Name: BOOK Physical Name: BOOK4

Figure 5. Relational Tables report from Class Diagram.

It Show the main object in this library working scenario i.e. BOOK (as object table). It shows the simple attributes, methods of these classes beside the relational dependencies between classes i.e. BOOK, BOOKCATEGORY, BOOK_ISSUE, DEPT and EMPLOYEE. The key attributes to relational dependencies to identify the records uniquely are shown with the keyword 'primary key'. In the following report from class diagrams to relational tables as shown in figure 5, there are number of attributes which are the domain candidates to the complexity of the class diagrams. As complexity of the class diagrams in following domain in figure 4 and 5 depends on the relational depth with in the existing classes i.e. relationship, how these classes are related to each other. Importantly in case of different class diagrams with respect to relational database, the complexity depends on many factors, out of them some factors are type of relationship (key-attributes) among classes i.e. are the relationship is bidirectional or unidirectional. Then the multiplex relationship among classes in class diagrams e.g. case of generalization, association and composition etc.

In the figure 5 the complexity of some of the factors to system design are candidate to such complexity (mechanical code with respect to Rational Rose). The software metric here is combination of the following characteristics, the number of key attributes (NKAS), Depth of Relationship between classes (DRC), Inter-relational attributes (IRA) and Inter-relational methods (IRM) as shown in table 4.

Metric Measurements:

The metric proposed on these findings (figure 4 & figure5) is shown in table 4. Here the characteristics such as, the key attributes (KAS) here are the combination of the number primary key and foreign key attributes used here. With the higher or lower value of (KAS) key attributes, it can be thus one of indication of the relationship dependencies (as some relationship do not need key attributes among classes) between the multiple relational classes.

Higher the value of the factor KAS, thus show the complex nature of relationship (DRC) between the different classes and hence relational tables in the database as shown in table 4.

The depth of relational dependencies (DRC) here between classes is a total factor of IRA and IRM. Where IRA is inter-relational attributes and IRM is inter-relational methods.

As the relationship among classes just not of relational dependencies (KAS), as it can be that of association, generalization, or composition etc. That's why, for this purpose, the DRC is also combination of these two factors here i.e. IRA and IRM, although DRC value as domain of i varies from 0, 1, 2... n as shown in table 4.

4 Figure 5 show some aspects of Oracle report (Relational database report from figure 4) for the class diagram as shown in the figure 4.

DRC= $\sum_{i=0}^{n}$ (IRAi + IRMi)	∑ i= 0,1,2,3... n	DRC= 29 (IRA= 19, IRM=10)
NKAS ≤ DRC	N= (number of)	KAS= 10
Comp.C≈ $\sum_{j=0}^{m}$ (DRCj + NKASj)	∑ j=0, 1, 2, 3...m	Comp. C= 29(fig. 5)[5]

Table 4: Metric Characteristics for software complexity with Relational database.

The DRC value consist of two candidates values IRA and IRM. The IRA can be key attributes as well as also non key attributes depending upon the system design requirements. Thus Relational hierarchy, i.e. depth of relationship between classes (DRC) is total sum of the two key factors here, 1st is shared methods between classes i.e. inter-relational methods IRM and then 2ndly inter-relational attributes in multiple classes. While NKAS ≤ DRC, depicts that although every occurrence of KAS show the relationing between classes but depth in relational classes is not only dependent to attributes too, as methods are also used for this purpose.

As number of key attributes showing here the existence of the relationship between different classes, beside KAS factor is also indication of depth in hierarchy to interrelated classes'. Hence total complexity in this relational database design (class diagrams) is combination of number of key attributes within classes (NKAS) as well as relational hierarchal tree (depth) with in classes i.e. DRC as shown in table 4. Thus complexity of class diagram here (table 4) is

$$\text{"Comp. C} \approx \sum_{j=0}^{m} (\text{DRCj} + \text{NKASj})\text{"}$$

, where value of this complexity domain varies from 0 to m i.e. j= 0, 1, 2 … m. Higher the value of factor j in the Comp. C, higher will be the complexity of the class diagrams in that relational database domain, the detail values to these aspects are described in example next.

Metric Results:
Looking in class diagram (in figure 4), the class diagram and metric characteristics in table 4 beside computational results of COCOMO model about database and product complexity too. The best suitable results to the complexity of the design with given measurements [table 4] are as follows:

Results: = ((DRC: Depth of relationship between classes), (KAS: Key attributes), Comp. C: Complexity of Classes))
{

20-40, 20-35, 40-50 /* GOOD*/
// {i.e. close range to this ratio}

5-15, 4-12, 20-25 /*ACCEPTABLE*/
// {intermediate range values here}

0-2, 0, 5-18, /* POOR*/
// {relational values hardly used}

50-80, 40, 60 /*POOR*/
// {to large values for relational values}

}

The suitable combinations to the designing of the system in this case can be e.g. {22, 15, 25} and poor results in this can be {1, 2, 5} or {43, 5, 55} or may be combination of both. As by having the lowest value of the initial two factors (i.e. DRC, Comp. C) show the less or no interaction between relational classes and vice versa highest value of these two factors depicts the ambiguous relational dependencies. By Boehm's computational values with COCOMO model if database (relational tables) size is nearer or greater then 100 show instability [13]. Thus the medium range or combinations of medium results between these three factors show more suitable strategy. Further work is required toward the Efforts and Difficulty level to the domain of relational systems with UML heuristics i.e. how can the complexities of UML heuristics affect the metric's characteristics of the relational system and security issues to relational systems.

5. CONCLUSION

The increasing demand of graphical systems design and to model driven architectures and modest improvement in quality to systems designs. There is further need of technologies to object integration beside growing complex structure to current systems. This work about software Quality with UML (QWUML) helps to establish efficient software quality metrics on the basis of UML diagrams. Thus importantly the need of quality with the aspects to software system toward issues like information multiplicity, mapping and system efficient reusing. Quality is the key determinant of success regarding software development; one can no longer rely on functionality & productivity of the system with out having quality with international standards.

In this work, efficient use of UML diagrams with relational system design (system) helps in fabricating optimal system which equally qualify for the constraints toward integrity & system reuse (code conversion). Thus data and service integrity to system is further boosted by the use of consistency checks (key-integrity & mapping issues) with UML and system

[5] Work related to Software quality attributes with UML diagrams use case diagram, sequence, deployment & activity diagrams etc is already accepted for publication in CFP-2006, Prague Czech.

requirement specifications. The working domain this way i.e. system design complexity with UML leads to the quality issues relating to increasing the efficiency, integrity (relational aspects) and system Understandability. Further work required in multiple topics in relation to system design issues like effort & difficulty, efficient reliability of system-design and domain constraints with UML diagrams.

REFERENCES

[1] C. J. Date, An introduction to Database system, Addison-Wesley pub, Inc, 1975.
[2] C. J. Date, An introduction to Database system (p. series), Vol-2, Addison-Wesley pub, Inc, 1985.
[3] Craig Larman, Applying UML and Patterns, Prentice Hall, Inc, 2000.
[4] Dan Pilone, Neil Pitman, UML 2.0 In a Nutshell, O'Reilly Media, Inc. 2005.
[5] ISO/IEC TR 9126-2: Software engineering – Product quality – Part 2: External metrics, 19-12-2000.
[6] Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified Software development process, Addison Wesley, Inc.
[7] Joc Sander, Eugene Currean, Software Quality, Addison Wesley, 1994.
[8] John A Mcdermid, Software Engineer's Reference Book, Butterworth-Heinemann, 1991.
[9] K. H. Möller, D. J. Paulish, Software Metrics, Chapman & Hall Computing, 1993.
[10] Leszek A, Maciaszek, Bruce Lee Liong, Practical Software Engineering, Pearson Education Ltd, UK 2005.
[11] Lorenz, Mark and Kidd, Jeff, Object Oriented Software Metrics, Prentice Hall Publishing, 1994.
[12] Shahid Nazir Bhatti, Why Quality? ISO 9126 Software Quality Metrics (Functionality) Support by UML Suite, SEN-2005, UK.
[13] Shari Lawrence Pfleeger, Software Engineering Theory & Practice, Prentice Hall, Inc, 2001.
[14] Tom Gilb, Dorothy Graham, Software Inspection, Addison Wesley, 1993.
[15] D.F.D'Souza, A.C.Wills, Objects, Components, and Frame works with UML, Addison Wesley, 1998.