Software Quality Improvement by Documentation – Knowledge Management Model

V. Chomal, Dr. J. Saini

Abstract This paper addresses an essential and significant concern in the development of computer software - its quality. The major proposal of the paper is that computer science faculty, in their design and implementation of core curriculum; do not devote sufficient attention to teaching their students how to develop high-quality software. As in industry, the most common and popular way of assuring the quality of programs is through software testing. In other words, quality is treated as a late addition or as postscript in software development. The paper presents and discusses a software quality improvement by documentation – knowledge management model that can be used to incorporate a wide variety of quality assurance techniques within a curriculum. The specific focus of this paper is to uncover various types of errors done by students during software development, which is been carried out as a part of their master degree programme.

Keywords: Data Flow Diagram (DFD), Error, Quality, Software Development Life Cycle (SDLC), Software Documentation, Software Projects

1. INTRODUCTION

Computing has become prevalent in almost all aspects of human endeavour; software is now critical to the current state of consumer products, transportation, medicine, law, management, and education. The information age has arrived and we as computer science educators are challenged as never before. Software products are larger and more complex, and their development demands immense resources in people, money, and time. Although we have made remarkable progress in the engineering of software, much of it is developed in an ineffective and inefficient manner - the software development landscape is filled with examples of projects with cost and schedule overruns and with severe quality problems. The quality issue can dominate and drive all of the other concerns in the development of software. It is not uncommon in the development of a software 'system to spend 40% to 60% of its development resources in testing. Of course, this is rarely anticipated, with the result that budgets and schedules are wrecked [24].

This paper intends to study and analyze the errors overlooked by students in developing their software projects for completion of their course work. The importance of final year student project reports is evident from the fact that they are on the verge of completing their course and enter the technical field expecting a thorough professional knowledge of software development.

Considering these errors, we propose a Software Quality Improvement by Documentation – Knowledge Management Model which will be helpful to lessen number of errors and improve the quality of software.

During their software project development, students are required to focus on the following aspects of the software:

1) Requirement Analysis
2) System Design
3) Database Design
4) Structural and Object Oriented Modelling Techniques
5) Coding
6) Testing

2. Software Quality

The three decisive issues in the development of software products are cost, schedule and quality. Out of these three fundamental issues, we will focus on quality, so what is quality? Quality is one of those terms that everyone uses; but there is probably not a universal, consistent understanding and agreement about its meaning. Software engineers (and academics) apply the term "quality" to both the product being produced and to the process used to produce it. Although these two types of "quality (product quality and process quality) are dependent on each other, they involve different techniques, measures, and implications. Broadly speaking, product quality is related to how well the product satisfies its customer/user requirements. Related to this (but maybe not specified) are the usability, performance, reliability, and the maintainability of the software. Process quality is concerned with how well the process used to develop the product worked. In this ease we are concerned with elements such as cost estimation and schedule accuracy, productivity, and the effectiveness of various quality control techniques (e.g., inspection rates and yields) [24].

Following the present section, the remaining paper is divided into four sections. First, a review of related literature is presented. This is followed by the methodology detailing the data source and phased procedure followed by us. A section on Findings and Analysis of Errors found during project reviews is presented next. Finally, the paper ends with concluding remarks.

3. Related Literature Review

Dennis et al [4] defines Software Development Life Cycle (SDLC) is a process used during the development of software system starting from planning until the

implementation phase. Data flow diagramming, on the other hand, is used to produce the process model during the analysis phase. SDLC is also used to understand on how an information system can support business needs, designing the system, building the system and delivering the system to users.

According to Lamsweerde [1] Requirement Analysis is the initial stage to start their software development. It is the procedure of shaping user expectations for a new or modified product.

Requirement Engineering can be defined as a subset of Software Engineering that mainly focuses on the use of various tools and techniques for the development and management of software requirements. The processes involved in Requirement Engineering include domain analysis, elicitation, specification, assessment, negotiation, documentation, and evolution.

After collecting requirements from the user as well as the organization for which students are developing their project and detailed analysis of a new system, the proposed system is to be designed. This period is known as system designing. System Design consists of two phases:

1) Beginning / Preliminary or General Design
2) Detailed Design

Database design is a very crucial part of development. It is the first step after requirement gathering and before coding even begins. Taking time to design the database can save a lot of time and frustration during development.

Jain et al [16] defines system design as the creation of the information system which is the solution to the problem. It is concerned with the coordination of the activities, job procedures and equipment utilization in order to achieve system goals. It deals with general design specifications, detailed design specifications, output, input, files and procedures. It also deals with program construction, testing and user acceptance.

Saleh [12] defines software design as the process of refining the software specifications to obtain a basis for describing how the software can be implemented and also suggested following constraints that must be followed while designing software:

1) Consistency: One of the most important criteria for the success of software is consistency of the interface, within the application. Examples of consistency within the same software design include the use of:

   a. Similar font sizes and styles in the screen, dialogs and forms.
   b. Same format to display error messages.
   c. Same icons to mean the same thing such as using the error, warning, and information icons.

2) Readability, Simplicity, and Clarity: The software design must include simple Graphical User Interface artifacts, such as simple screens with clear, uncrowded messages. For example, the use of appropriate colours, font sizes, and styles that is convenient for the target users of the software. Displayed error, help and warning messages must be clear, concise, and as elementary as possible to assist the user in properly choosing the next action to be performed.
3) User friendliness: The software design must be user friendly, providing helpful, considerate, and non-offending messages. For example, an error message displayed by the system should spell out exactly what the error is and how it can be avoided. Help messages must be concise and contextual by providing only the needed and requested help.

Hecht [8] and [9] and Shah et al [10] state the importance of exception handling in software development has been known for a long time, and considerable research has been performed to develop tools and techniques to aid developers in incorporating exception handling into their programs. Missing or faulty exception handling has caused a number of spectacular system failures and is a major cause of software failures in extensively tested critical systems.

Structural and Object Oriented Modelling Techniques is characterized by the use of graphic documentation separating the design phase into logical phase and physical design. Through the use of graphic documentation, the system is described in unambiguous terms so that it can be understood fully be the students to be programmed correctly. In structured methodology, data flow diagrams are used during analysis phase to capture the requirements of any system. In object-oriented methodology, Unified Modelling Language (UML) specification is used to represent any system requirements.

In computer programming, the term coding refers to writing a set of instructions for software application. It is a set of statements and instructions written in a human-readable programming language. Hecht [8] states the needs for exception handling which is the most vital facet to be included while coding for software product.

Testing is a process to identify the correctness and completeness of the software. The general objective of software testing is to affirm the quality of software

system by systematically exercising the software in carefully controlled circumstances. IEEE defines testing as 'an examination of the behaviour of the program by executing on sample data sets.' Miller [7] and Khan [18] state that the general aim of testing is to affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances. For conducting testing, proper test data and test plan must be prepared to identify the correctness and completeness of the software. IEEE defines test case as 'a set of input values, execution preconditions, expected results and execution post conditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement'.

Selby et al [20] in their work define several error related concepts as:

1) Error related effort: Error isolation effort – How long it takes to understand where the problem is and what must be changed. Error fix effort – How long it takes to implement a correction for the error. Error correction effort – How long it takes to correct an error, which is the sum of error isolation effort and error fix effort.
2) Error type: Wrong – Implementation require a change. The existing code or logic needs to be revised, the functionality is present but it is not working properly. Extra – Implementation requires a deletion. The error is caused by existing logic that should not be present. Missing – Implementation requires and addition. The error is caused by missing logic or function.
3) Error severity (trouble reports): Level 1 – Program is unusable, it requires immediate attention. Level 2 – Program is unusable, but functionality is severely restricted and there is no work – around. Prompt action is required. Level 3 – Program is usable, but has functionality that is not critical, it can be avoided. Level 4 – Problem is minor, example message or documentation error and is easily avoided.
4) Error severity (inspection): Major – Error could lead to a problem reported in the field on a trouble report. Minor – Anything that is less than "major" severity, example, typographical mistakes and misspelling.
5) Error reporter type (trouble reports): User – Error is reported by field user or found by a developer while using the product. Developer – Error is discovered by a developer during field testing or when looking at the source code or searching for error in a released system.
6) Inspection type: Design inspection – These are inspections held during the high level and low level design phase. Engineering inspection – These are code inspection that are held after the completion of unit testing.

Grady [21] suggest that one useful way to evaluate software defects is to transfer process learning from individuals to organizations. It includes not only analyzing software defects but also brainstorming the root causes of those defects and incorporating what we learn into training and process changes so that the defects won't occur again. There are five steps:

1) Extend defect data collection to include root-cause information. Start shifting from reactive responses to defects toward proactive responses.
2) Do failure analysis on representative organization-wide defect data. Failure analysis is the evaluation of defect patterns to learn process or product weaknesses.
3) Do root-cause analyses to help decide what changes must be made. Root-cause analysis is a group reasoning process applied to defect information to develop organizational understanding of the causes of a particular class of defects.
4) Apply what is learned to train people and to change development and maintenance processes.
5) Evolve failure analysis and root-cause analysis to an effective continuous process improvement process.

Peng et al [26] provides guidance on software error analysis. Software error analysis includes error detection, analysis, and resolution. Error detection techniques considered in the study are those used in software development, software quality assurance, and software verification, validation and testing activities. These techniques are those frequently cited in technical literature and software engineering standards or those representing new approaches to support error detection. The study includes statistical process control techniques and relates them to their use as a software quality assurance technique for both product and process improvement.

Hilburn et al [24] place forward that computer science curriculum designers and developers treat software as a central element of their curricula. Software quality should be emphasized in the beginning and throughout the curriculum. Quality reviews (personal reviews, walkthroughs, inspections) should be incorporated at the requirements, design and coding stages of software development. Finally, we do not believe that students can learn to produce high-quality software without the proper guidance, organization, and support. Hence, we believe that it is essential that a software development process that emphasizes quality techniques, methods, and analysis be taught to and used by students throughout their program. Such a process would typically consist of a set of process scripts, forms, guidelines, measures, and tools.

According to Jones [2] an analysis of approximately 250 large software projects between 1995 and 2004 shows an interesting pattern. When comparing large projects that successfully achieved their cost and schedule estimates against those that ran late, were over budget, or were cancelled without completion, six common problems were observed: poor project planning, poor cost estimating, poor measurements, poor milestone tracking, poor change control, and poor quality control. By contrast, successful software projects tended to be better than average in all six of these areas. Perhaps the most interesting aspect of these six problem areas is that all are associated with project management rather than with technical personnel. Two working hypotheses emerged: 1) poor quality control is the largest contributor to cost and schedule overruns, and 2) poor project management is the most likely cause of inadequate quality control.

Dixit et al [5] define data flow diagram as a graphical tool that allows system analysts and users to depict the flow of data in an information system. A method for checking consistency for UML specification, on the other hand, has been done for example in Donald et al [6], Kim et al [13] and Molina et al [15]. Donald et al [6] proposed the following set of rules that must be followed by analysts when drawing data flow diagrams in order to evaluate data flow diagrams for correctness, which we too included in analyzing data flow diagram for the documentation reviewed by us.

1) The context diagram shows the overall business process as just one process and shows the data flows to and from external entities. Data stores are not usually included on the context diagram.
2) At least one input or output data flow for external entity.
3) At least one input data flow and/or at least one output data flow for a process.

4) Output data flows usually have different names than input data flows for a process.
5) Data flows only in one direction.
6) Every data flow connects to at least one process.
7) A process, data flow, data store and external entity must have unique name in data flow diagrams.
8) Every process is wholly and completely described by the processes on its children DFDs.
9) Every data flow, data store and external entity on a higher level DFD is shown on the lower-level DFD that decomposes it.
10) For every data store, data cannot move directly from one data store to another data store as well as data must be moved by a process only.

11) The total number and name of external entities in context diagram are the same as in level 0 DFD.
12) The total number and name of data flows between process and external entity in context diagram is same as level 0 DFD.
13) The total number and name of external entities in level 0 DFD are same as context diagram.
14) The total number and name of data flows between process and external entity in level 0 DFD are the same as in context diagram.

Visconti et al [17] proposed that, one basic goal of software engineering is to produce the best possible working software along with the best possible supporting documentation. And yet, documentation seems to be considered a second class object and not as important as the software itself. However, empirical data shows that low quality or missing documentation is a major cause of errors in software development and maintenance. Low quality or missing documentation is a major cause of errors in software development and maintenance. The present work has also considered documentations of software projects as an input for analysis. In next section, we present the detailed methodology followed by us towards analysis of documentations of software projects.

4. Methodology

The present work considers documentation of software projects prepared by students as a source for data collection. Specifically, documentations of large software projects of only final year students of Masters level course have been considered for the research purpose. The duration of these software projects is six months. The said documentations of software projects were procured from college libraries. These documentations include complete project profile along with the following elements:

1) Requirement analysis
2) Technology used
3) Database design
4) Structural and Object Oriented Modelling Techniques
5) Screen layouts
6) Testing techniques along with test case and data

We analysed and reviewed 505 large software project documentations developed during a period of academic years from 2001-2002 to 2011-2012. During our exploration we considered all of the above described elements. For simplicity and better exhaustive analysis of the documentations, the phased process was followed. As each project is a uniquely different definition from other projects,

it is noteworthy here that this was repeated for each of the 505 project reports under study.

These phases are presented below:

1) Exploration of Project Profile
2) Exploration of Existing System and Proposed System
3) Exploration of Requirement Gathering Techniques
4) Exploration of Requirement Analysis done by Students
5) Exploration of Technology on which Software Project carried out
6) Exploration of Process Model adapted for Software Project Development
7) Exploration of Data Dictionary (including Database Design)
8) Exploration of various Structural and Object Oriented Modelling Techniques
9) Exploration of Screen Layouts
10) Exploration of Generated Reports
11) Exploration of Testing Techniques and Test data

As shown in Figure 1, Diagrammatic Representation of Knowledge Management , the software documentation produced from software developed by students as a part of their Final Year curriculum are been considered as a source of data collection. While studying and analysis these software documentation, we found and categorized errors into eleven broad error categories as shown in Figure 1. "Software Quality Improvement by Documentation – Knowledge Management Model", is proposed as a solution which is implemented in Java Programming Language, will help in highlighting the errors which are shown in Figure 1 and Table 1, done during various software development stages and guiding software professional to remove these errors and improve the quality of software.

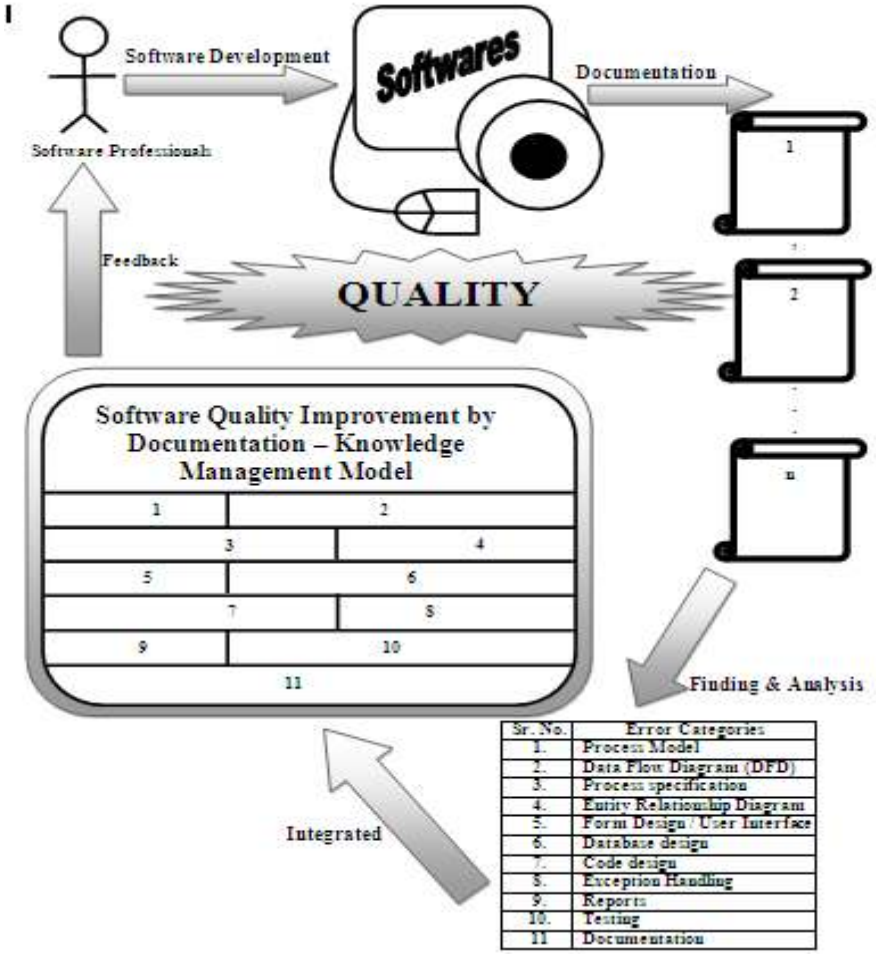


Figure 1 Diagramatic Representation of Knowledge Management

The next section presents the findings obtained through analysis of documentation reports.

5. Findings and Analysis

The main errors and error categories found from the review of project documentations are listed in Table 1. We also present an analysis of the identified points which can be termed as the errors. There were eleven broad categories under which various errors were found. These broad categories are:

1) Process Model
2) Data Flow Diagram

3) Process Specification
4) Entity Relationship Diagram
5) Form Design / User Interface
6) Database Design
7) Code Design
8) Exception Handling
9) Reports
10) Testing
11) Documentation

Table 1. Error Categories and Errors

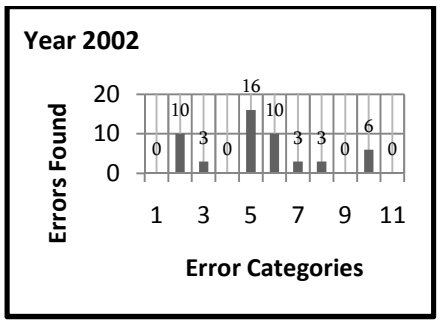
Sr No.	Error Categories and Errors
1.	**Process Model**
1.1	Students state for model but do not explain why the model is appropriate for their system.
2.	**Data Flow Diagram (DFD)**
2.1	DFD are not properly numbered.
2.2	In DFD no labelling is done on data flow.
2.3	Levels are not properly maintained.
2.4	Context diagram does not have the name of the system.
2.5	Data was provided by source or a sink to another source or sink without processing.
2.6	Data movement from source to data store without processing.
2.7	Data movement shown from data store to source without processing.
2.8	Movement of data from data store to data store.
3.	**Process specification**
3.1	In process specification, external entity and data store not shown.
4.	**Entity Relationship Diagram**
4.1	In Entity Relationship Diagram, relationship among entities not mentioned.
4.2	Entity Relationship Diagram confused with Table Relationship Diagram.
5.	**Form Design / User Interface**
5.1	No specification provided regarding which fields are mandatory while filling a form.
5.2	No details regarding filling up of text box and other necessary components (Tools Tips not provided).
5.3	After completing any transaction no information is provided to

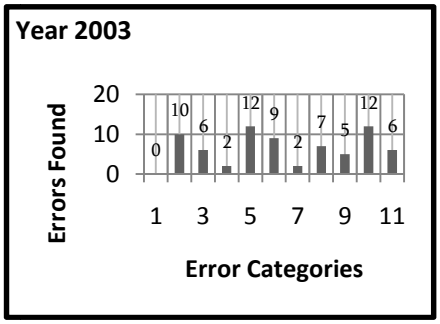
	user.
5.4	Too many details presented on single page.
5.5	Text on label not readable properly.
5.6	Control / Components on the form not arrange properly.
5.7	Uniformity of font's type and size not maintained on forms and buttons.
6.	**Database design**
6.1	Ignoring normalization.
6.2	Poor naming standards.
6.3	One table to hold all domain values.
6.4	Rule of primary key and foreign key not followed.
6.5	Unnecessary assignment of field.
6.6	Proper field not selected for assigning Primary Key.
6.9	Proper description not mentioned for designing field.
6.10	Table field not properly designed (redundancy).
6.11	Data type not properly assigned.
6.12	Data size not properly assigned.
6.13	Specification not provided why table is been designed and its use.
7.	**Code design**
7.1	Code design not followed for proper identification of code.
8.	**Exception Handling**
8.1	Appropriate messages are not displayed in message box (Error handling is not user friendly).
9.	**Reports**
9.1	Reports not properly designed (Date/Time not mentioned, for whom report is been generated)
9.2	Graphs and Charts not having proper naming (legends).
9.3	Complete details not found in report.
9.4	Header and Footer not included in report.
10.	**Testing**
10.1	Test Case and Test Data not included.
11.	**Documentation**
11.1	Documentation does not have uniformity.

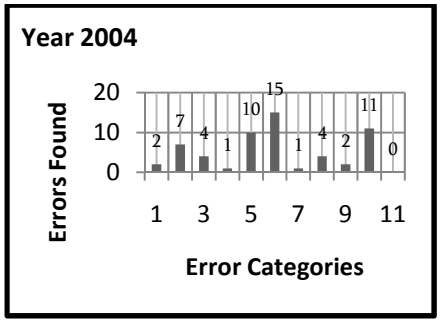
The total number of errors found during our exploration which belongs to the said error categories along with the summation of total number of errors encountered for the academic years from 2001-2002 to 2011-2012 is highlighted in Table 2.

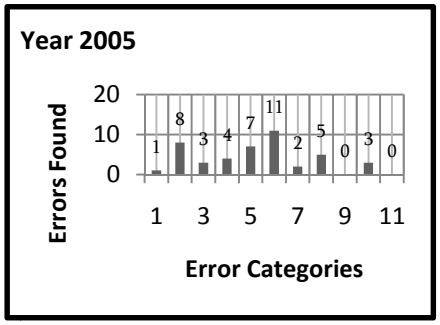
Table 2. Error Category–wise Year-wise Number of Errors

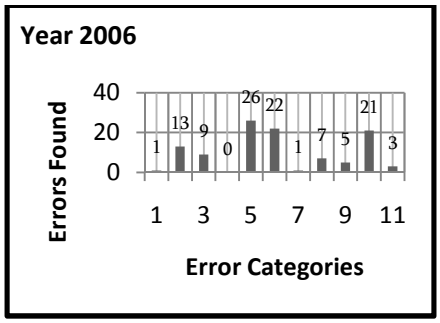
Error Categories	Year											Total
	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	
1	0	0	2	1	1	0	5	7	6	10	15	47
2	10	10	7	8	13	24	13	19	8	19	11	142
3	3	6	4	3	9	3	2	2	5	7	3	47
4	0	2	1	4	0	2	5	5	5	14	15	53
5	16	12	10	7	26	31	15	14	21	62	43	257
6	10	9	15	11	22	47	45	69	65	46	45	384
7	3	2	1	2	1	0	0	2	2	4	5	22
8	3	7	4	5	7	8	5	6	13	11	9	78
9	0	5	2	0	5	8	1	4	4	15	16	60
10	6	12	11	3	21	16	10	7	13	12	13	124
11	0	6	0	0	3	8	5	7	10	7	5	51
Total	51	71	57	44	108	147	106	142	152	207	180	1265

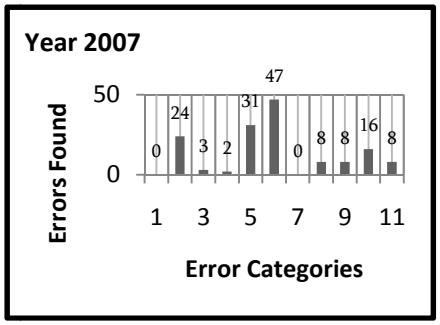

Year 2002
Errors Found
20
10
0
0 10 3 0 16 10 3 3 0 6 0
1 3 5 7 9 11
Error Categories

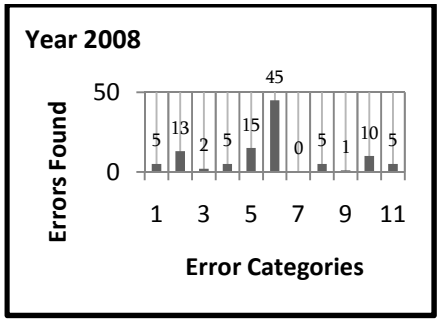

Year 2003
Errors Found
20
10
0
0 10 6 2 12 9 2 7 5 12 6
1 3 5 7 9 11
Error Categories

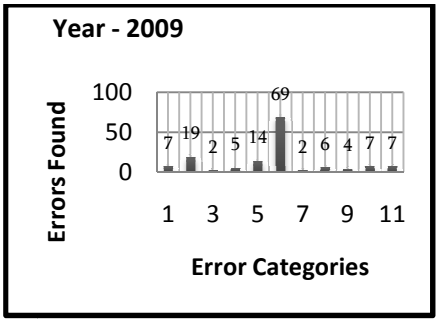

Year 2004
Errors Found
20
10
0
2 7 4 1 10 15 1 4 2 11 0
1 3 5 7 9 11
Error Categories

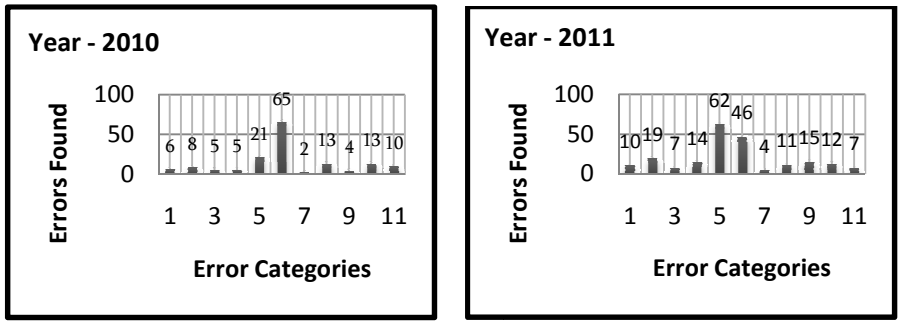

Year 2005
Errors Found
20
10
0
1 8 3 4 7 11 2 5 0 3 0
1 3 5 7 9 11
Error Categories


Year 2006
Errors Found
40
20
0
1 13 9 0 26 22 1 7 5 21 3
1 3 5 7 9 11
Error Categories


Year 2007
Errors Found
50
0
0 24 3 2 31 47 0 8 8 16 8
1 3 5 7 9 11
Error Categories


Year 2008
Errors Found
50
0
5 13 2 5 15 45 0 5 1 10 5
1 3 5 7 9 11
Error Categories


Year - 2009
Errors Found
100
50
0
7 19 2 5 14 69 2 6 4 7 7
1 3 5 7 9 11
Error Categories



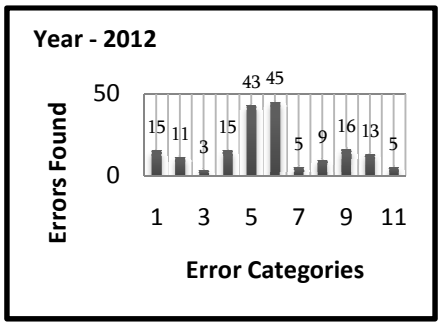


Figure 2. Error Categories versus Number of Errors Found in Respective Category (a) Year – 2002 (b) Year – 2003 (c) Year – 2004 (d) Year – 2005 (e) Year – 2006 (f) Year – 2007 (g) Year – 2008 (h) Year – 2009 (i) Year – 2010 (j) Year – 2011 (k) Year – 2012

We can see from Figure 2 that the sources of errors vary significantly for the eleven academic years from 2001-2002 to 2011-2012. No charts are alike. In Table 3, we present the top two error categories for the academic years from 2001-2002 to 2011-12 where maximum errors were found.

Table 3. Year–wise Most Frequent & Second Most Frequent Error Category

Year	Most Frequent Error Category	Error Category No. from Table: 1
2002	Form Design / User Interface	5
2003	Form Design / User Interface Testing	5 10
2004	Database Design	6
2005	Database Design	6
2006	Form Design / User Interface	5
2007	Database Design	6
2008	Database Design	6
2009	Database Design	6
2010	Database Design	6
2011	Form Design / User Interface	5
2012	Database Design	6

Year	Second Most Frequent Error Category	Error Category No. from Table: 1
2002	Data Flow Diagram Database Design	2 6
2003	Data Flow Diagram	2
2004	Form Design / User Interface	5
2005	Data Flow Diagram	2
2006	Database Design	6
2007	Form Design / User Interface	5
2008	Form Design / User Interface	5
2009	Data Flow Diagram	2
2010	Form Design / User Interface	5
2011	Database Design	6
2012	Form Design / User Interface	5

From Table 3, as the error category number 6 of Table 1 is having maximum occurrence, we conclude that 'Database Design' is the error category where highest number of errors have been encountered during last eleven years.

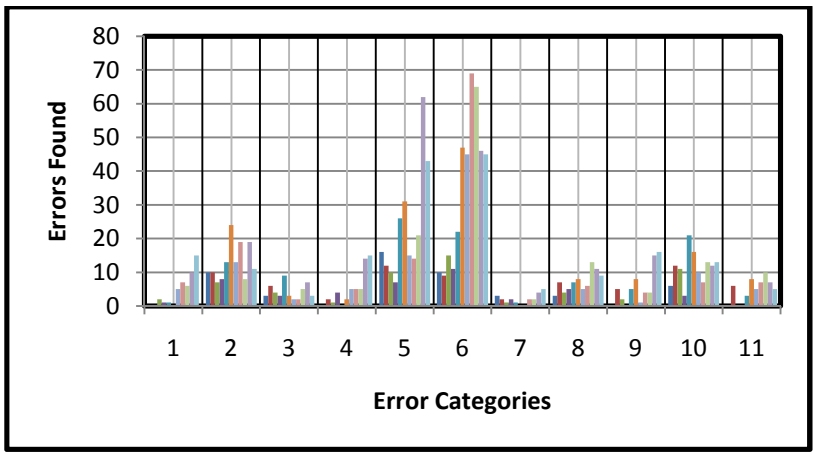


Figure 3. Error Categories and Number of Errors for Year 2002 To Year 2012

Figure 3 show Error Categories and Number of Errors from Year 2002 To Year – 2012. Error category number 6 of Table 1 'Database Design' shows the tallest spike followed by error category number 5 of Table 1 'Form Design / User Interface'

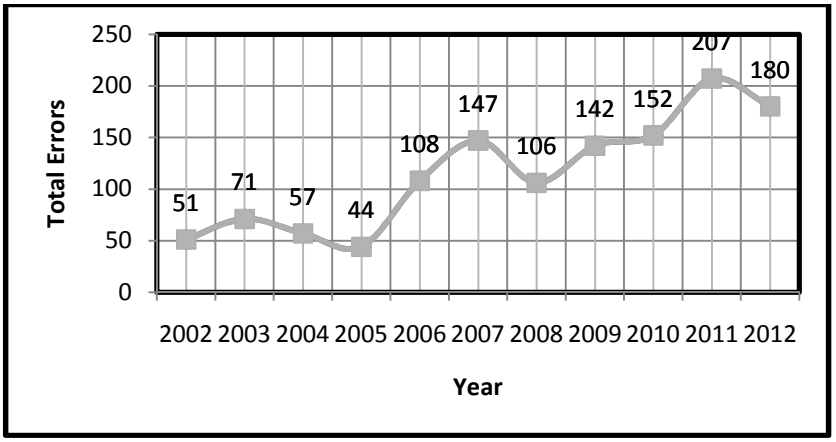


Figure 4. Total Number of Errors found from academic year - 2002 to year - 2012

Figure 4 show that total numbers of errors are having an upward trend from 2002 to 2012.

6. CONCLUSION

Documentation of large software projects prepared by final year students of Master level course have considered as a basis of data collection. The duration of these software projects was six months. Errors were analysed from these software projects documentation. We found many errors and classified them into error categories viz... Process Model, Data Flow Diagram, Process specification, Entity Relationship Diagram, Form Design / User Interface, Database design, Code design, Exception Handling, Reports, Testing and Documentation. We analysed number of errors in each category and carried out this procedure for the academic years 2001-2002 to 2011-2012. Based on our study for these academic years, we conclude that sources of errors vary significantly over a period of time. Among the extremely significant error categories derived from our finding and analysis, we conclude that 'Database Design' is the error category where highest number of errors have been encountered during last four years. Form Design / User Interface is the second most notorious error category been identified from finding and analysis. Our final conclusion is that the overall trend of total number of errors for the period of study shows an upward trend. Hence, if due concern is not given to the most frequently occurring error category identified by us then a still higher number of errors are predicated for projects to develop.

Even though, we all agree upon the statement that software quality is crucial in software development but, we as educator often do not make it a major concern in the curriculum we deliver. We recommend that software quality should not

be an afterthought - it should be addressed in the front-end of the life-cycle. We believe that software development oriented curriculum should focus on quality and that the Software Quality Improvement by Documentation – Knowledge Management Model provides the conceptual framework for such a focus.

7. References

1. Axel van Lamsweerde., "*Requirement Engineering in the Year 00: A Research Perspective* ", proceeding of the 22nd International conference on Software Engineering (2000).

2. Capers Jones., "*Software Project Management Practices: Failure versus Success*" Software Productivity Research LLC October (2004).

3. Cheng Bo., and Meng Xiang-Wu Chen Jun-Liang., "*An adaptive user requirements elicitation framework*", 31st Annual International Computer Software and Applications Conference IEEE (COMPSAC 2007)

4. Dennis A., Wixom B.H. and Roth, R.M., "*Systems Analysis and Design*", 3rd ed. Hoboken: John Wiley & Sons, Inc (2006).

5. Dixit, J. B., and Kumar R.,. "*Structured System Analysis and Design*" Paperback edition. New Delhi, India: Laxmi Publisher (2008).

6. Donald S., and Le Vie Jr., "*Understanding Data Flow Diagram*", Proceedings of the 47th annual conference on Society for Technical Communication. Texas: Integrated Concepts, Inc (2000).

7. E. F. Miller., "*Introduction to Software Testing Technology*", Tutorial: Software Testing & Validation Techniques, Second Edition, IEEE Catalog No. EHO 180-0, pp. 4-16.

8. Herbert Hecht. , "*A Systems Engineering Approach to Exception Handling*", Proceeding ICONS '08 Proceedings of the Third International Conference on Systems Pages 190-195 IEEE Computer Society Washington, DC USA (2008).

9. Herbert Hecht. , "*Establishing Requirements for Exception Handling*"

10. Hina Shah. , and Carsten Gorg. , and Mary Jean Harrold. , "*Why Do Developers Neglect Exception Handling?*", Proceeding WEH '08 of the 4th International workshop on Exception Handling Pages 62 – 68, New York, USA, (2008).

11. Jeffrey A. H., and George J.F., and Valacic J.S., "*Modern Systems Analysis and Design*" 3rd ed. US (2002)

12. Kassem A. Saleh, "*Software Engineering*" India Edition, Published by J. Ross (2009)

13. Kim D.H., and Chong K., "A Method of Checking Errors and Consistency in the Process of Object- Oriented Analysis", Proceedings of the 1996 Third Asia-Pacific Software Engineering Conference. Korea: IEEE Computer Society. Pp. 208-216 (1996).

14. Lester O. Lobo., and James D. Arthur. , "*Effective Requirements Generation:*

15. Synchronizing Early Verification & Validation, Methods and Method Selection Criteria", CORR abs/cs/0503004: (2005).

16. Lucas F.J., and Molina, F., and Toval, A., A "*Systematic Review of UML Model Consistency Management. Information and Software Technology*", 51(12), pp. 1 – 15 (2009).

17. Madhulika Jain, and Vineeta Pillai., and Shashi Singh., and Satish Jain., "*System Analysis, Design and Management Information System made Simple*" – BPB Publications.

18. Marcello Visconti., and Curtis Cook., "*Software System Documentation Process Maturity Model*", Proceeding CSC '93 of the 1993 ACM conference on Computer Science Pages 352 – 357 New York, USA, (1993).

19. Mohd. Ehmer Khan. , "*Different Forms of Software Testing Techniques for Finding Errors*", International Journal of Computer Science Issues, ISSN: 16940784, Volume: 7, Issue: 3, Pages 11-16, (2010).

20. Muhammad Waseem., and S. A. K. Ghayyur, "*An interrogative review of requirement engineering*", International Journal of Reviews in Computing (IJRIC), ISSN: 2076-3328 (2009).

21. Richard W. Selby., and Victor R. Basili., "*Analyzing Error – Prone Systems*", IEEE Transactions on Software Engineering Volume 17 Issue 2, Pages 141-152, (1991)

22. Robert B. Grady. , "*Software Failure Analysis for High-Return Process Improvement Decisions*", Article 2, Hewlett-Packard Journal (1996).

23. Rosziati Ibrahim., and Noraini Ibrahim., "*A Tool for Checking Conformance of UML Specification*", Proceedings of the 2009 World Academic of Science and Technology (WASET), Volume 51, pp. 262-266 (2009).

24. Rosziati Ibrahim., and Siow Yen., "*Formalization Of The Data Flow Diagram Rules For Consistency Check*", International Journal of Software Engineering & Applications , Publisher: Academy & Industry Research Collaboration Center (AIRCC), (2010).
25. Thomas B. Hilburn., and Massood Towhidnejad., "*Software Quality: A Curriculum Postscript?",* Proceedings of the thirty – first SIGCSE technical symposium on Computer Science Education, pages 167 – 171, New York, USA (2000).
26. Victor R. Basili., and Barry T. Perricone., "*Software Errors and Complexity: An Empirical Investigation*", Magazine: Communications of the ACM, Volume: 27, Issue: 1, pages 42 – 52, New York, USA (1984).
27. Wendy W. Peng. , and Dolores R. Wallace., "*Software Error Analysis*", NIST Special Publication 500 – 209, (1993).

AUTHORS' PROFILE:

Vikas Sitaram Chomal

vikschomal@yahoo.co.in
Assistant Professor
Narmada College of Computer Application,
Bharuch, Gujarat, India – 392 011

Dr. Jatinderkumar R. Saini

saini_expert@yahoo.com
Director (I/C) & Associate Professor
Narmada College of Computer Application,
Bharuch, Gujarat, India – 392 011.