Design Quality Metrics for Real Time Environment Applications.

Ph.D. Dissertation
(Session 2005)

Supervised By
Prof. Dr. Muhammad Shoaib

Submitted By
Mr. Mahmood Ahmed
2005-Ph.D-CS-05

Department of Computer Science & Engineering
University of Engineering and Technology
Lahore-Pakistan
2012
Design Quality Metrics for Real Time Environment Applications.

Ph.D. Dissertation
(Session 2005)

Supervised By

Prof. Dr. Muhammad Shoaib

Submitted By

Mr. Mahmood Ahmed

2005-Ph.D-CS-05

Department of Computer Science & Engineering
University of Engineering and Technology
Lahore-Pakistan
Design Quality Metrics for Real Time Environment Applications.

Mr. Mahmood Ahmed

2005-Ph.D-CS-05

Supervisor

Prof. Dr. Muhammad Shoaib

A dissertation submitted for the degree of
Doctor of Philosophy
Computer Science

Internal Examiner

Prof. Dr. Muhammad Shoaib
Department of Computer Science
and Engineering University of
Engineering & Technology, Lahore

External Examiner

Dr. Sonia Gul
Associate Professor
Department of Computer Science
Lahore College for Women
University

Prof. Dr. M.A. Maud
Chairman
Department of Computer Science
and Engineering University of
Engineering & Technology, Lahore

Dean
Faculty of Electrical Engineering
University of Engineering &
Technology, Lahore

Department of Computer Science & Engineering
University of Engineering and Technology
Lahore-Pakistan
In the name of ALLAH the most beneficent and the most merciful.
Dedicated

This PhD Dissertation is dedicated to all who by no means stopped trusting me and who along with Allah (Subhana hu ta ala) have been my source of guidance and specially to my mother (late), My father (late), My Wife & My Children.
None of the material contained in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or the institution.
Acknowledgment

First I would like to thank my Allah who helped me when ever I asked for help. Then I am greatly thankful to my respected internal supervisor, Professor Dr. Muhammad Shoaib, for their incessant guidance, support and supervision of this research work. University of Engineering and Technology and Higher Education Commission also made it possible to continue my PhD research work through financial assistance. I also greatly acknowledge the support and financial assistance of my parent organization for allowing me to do my research work. I also acknowledge the support of SDMetrics which provided free academic license for full version after confirming that University of Engineering and Technology is a non-profit organization and its students are eligible to receive free academic license.
Abstract

In this research work a design framework model for real time environment applications driven by novel design metrics has been proposed. Objective is to use these design metrics for measuring the design of real time environment applications. Measurement at the design level can save effort and cost. The existing design methodologies for real time systems are lacking with this measurement aspect. In this research work measurement has been introduced in the form of metrics for the design methodology of the real time system. The metrics are validated and the results are satisfactory. The case study has shown that using these metrics at the design level can reduce design errors for the real time applications. The further research can be carried out for development and implementation of real time application using the proposed real time application design framework model and the design metrics.
Table of Contents

Dedicated ........................................................................................................ IV
Declaration ....................................................................................................... V
Acknowledgment ............................................................................................ VI
Abstract ........................................................................................................... VII
Table of Contents ............................................................................................ VIII

List of Figures .................................................................................................. X
List of Tables .................................................................................................... XI

Chapter No. 1 .................................................................................................. 12
Introduction ....................................................................................................... 12
  1.1 Overview .................................................................................................. 12
  1.2 Motivation ................................................................................................. 13
  1.3 Contribution .............................................................................................. 14
  1.4 Organization ............................................................................................. 14

Chapter No. 2 .................................................................................................. 16
Literature Survey ............................................................................................... 16
Summary ............................................................................................................. 41

Chapter No. 3 .................................................................................................. 42
Metrics Driven Framework Model for the Design of Real Time Environment Applications (MDMF) ........................................................................................................ 42
  3.1 Proposed Design Model for Real Time Applications .............................. 42
  3.2 Description of the Proposed Model ......................................................... 44
Step 1: Specification ......................................................................................... 44
Step 2: Object Identification .......................................................................... 45
Step 3: Decomposition .................................................................................... 46
Step 4: Integrated Real Time Application Design ......................................... 47
Step 5: Measurement ....................................................................................... 48
Metrics for Measurement ............................................................................... 49
  1 Soft Deadline Cohesion Factor ................................................................. 49
  2 Hard Deadline Cohesion Factor ................................................................. 49
  3 Overridden Deadline Class Factor ........................................................... 50
  4 Overridden Complexity Factor ................................................................. 51
  5 Message Exchange Factor ....................................................................... 51
  6 Early Decomposition Factor ..................................................................... 51
  7 Deadline based Predictability Factor ...................................................... 52
  8 Life Cycle Support Factor ....................................................................... 52

VIII
List of Figures

Figure 1: Example showing the distorted view of NOC metric [5]. 21
Figure 2: Examples of (a) less cohesive (b) densely cohesive class [5]. ................................................................. 22
Figure 3: Example for POF metric [5] .............................................................. 23
Figure 4: Relation between the control and the process model [7] ................................................................. 24
Figure 5: OMT state diagram [21] .............................................................. 28
Figure 6: Two essential steps common to software development projects [29] ........................................................................................................................................ 31
Figure 7: Implementation steps for large software project [29] .... 32
Figure 8: Iterative interaction between various phases ......................... 33
Figure 9: The waterfall Model [30] .............................................................. 34
Figure 10: Prototyping Model [31] .............................................................. 35
Figure 11: Iterative Developmental Model [31] ........................................... 36
Figure 12: State chart + Solution [35] .......................................................... 36
Figure 13: Timedstate state charts Solution .................................................. 37
Figure 14: Histogram plot for DIT [42] .......................................................... 39
Figure 15: Histogram plot of the NumOp (WMC) [42] ................................. 40
Figure 16: Kiviat Plot for the rule ExpressionNode module [42] .... 40
Figure 17: Design rule violation at package level [42] ............................. 41
Figure 18: MDMF the proposed framework model for real time application .................................................................................................................. 42
Figure 19: Specification Phase ................................................................. 45
Figure 20: Object Identification Phase ........................................................... 46
Figure 21: Statechart with timer ................................................................. 46
Figure 22: Deadline as a transition condition ............................................. 47
Figure 23: Concurrent State Representation using Statechart [41].47
Figure 24: The proposed measurement framework model............................. 48
Figure 25: Hardware software selection based on size............................... 53
Figure 26: Deployment Phase .................................................................... 54
Figure 27: Weapon Delivery System Modules [43] ................................. 57
Figure 28: Concurrent State threads with timing info [43] .................... 60
Figure 29: 3-D Plot of LCSF metric .............................................................. 68
List of Tables

Table 1: CK Metrics Suite [3] ................................................................. 17
Table 2: MOOD Metrics Suite [4] ......................................................... 18
Table 3: Metric values for the weapon delivery design..................... 65
Table 4: Life Cycle Support Factor for different methodologies....... 67
Chapter No. 1

Introduction

1.1 Overview

In the field of science and engineering, measurement is playing a major role in improvement of processes, designs, mathematical models etc. Recently measurement has gained importance in the field of software engineering [1]. Proper measurement makes development faster, improves the quality of design and code, reduce the effort and cost. Measurement also helps to decide most feasible design from different design alternatives.

According to Rob Williams, “a real time application is one that takes into consideration the constraints like: strict timing limit on response of the system, event driven scheduling, low-level programming, software highly coupled to particular hardware, committed dedicated function, the computing system might be within a control loop, variables are normally volatile, Multi-tasking is often implemented, scheduling demand is Run-time, environment is also unpredictable, continuously running system is requirement, and is used as life-critical applications” [9]. Design of the real time systems has always been a challenge [7]. This challenge is due to the difficulty in incorporating timing information of various tasks in the design architecture. Most designers left this deadline management as an extra task for the developer to handle, in the implementation phase [16]. Methodologies are lacking deadlines and support for inheritance of deadlines for design of real time applications. Usability is another property which is gaining interest. Usability is an elusive quantity. The novice users want usability as the most desired property. But the
properties that lead an application to user friendliness for novices are often dissimilar from those preferred by expert users or the software designers [7]. It is very difficult to measure usability, but informal methods can be used such as feedbacks of users from surveys. The design methodologies used for real time application design except S-OODM [27] have not provided any way to measure the design. In the present research work metrics for real time applications design has been proposed.

1.2 Motivation

Design of the real time system has always been a challenging task for the software designers. The main reason of this difficulty is the stringent timing restrictions imposed by various tasks in the real time environment application. The design complexity increases when these timing restrictions are introduced early in the software development life cycle. A mechanism is required through which these restrictions can be addressed early at design level. By doing this, cost effectiveness can be obtained and quality aspects can be handled by quality attributes at the design level. Especially for a real time application, the quality cannot be compromised because this may result in a catastrophe. The existing design methodologies for real time application do not consider the task deadlines or the timing requirements at design level [8-27]. This task is normally left as an implementation challenge for the developer to take care at the programming language level. Especially when object oriented design approach is adopted the deadline handling becomes more difficult. This difficulty is due to characteristics of object oriented approach like inheritance of an object along with its deadline. For non-real time application it may not be necessary but for real time application, dangers of accidents of loss of life are main motives that compel us to take extra care of the quality of the application through measurement. Measurement at design level provides an additional quality control
feature before development. The existing metrics are measuring a structural design based on SA/SD [9] design techniques or for measuring object oriented design metrics.

Most of the design methodologies used for real time application design have not provided any way to measure the design. S-OODM [27] has provided metrics for measuring a web based application design. In the present research work; emphasis is on defining the design framework model and metrics for real time environment application’s design level measurement.

1.3 Contribution

The research has been carried out to define the metrics for the design of real time application. By proposing these metrics, issues related to the deadlines of the tasks have been incorporated at the design level in real time application. The problem of inheritance of deadlines when object oriented design approached is followed, is sorted out. Classical waterfall model has been extended for real time environment applications. A metrics based design framework model for real time applications has been proposed. Measurement has been introduced in this framework model in the form of metrics. The proposed metrics are validated by implementing these metrics with a case study. The results show that these metrics can help to improve the quality of design, reduce effort and cost of development of real time application.

1.4 Organization

The thesis is organized in the following way. The literature survey is given in chapter 2. A framework for the real time application design has been given in the chapter 3. Proposed design metrics are given in chapter 4. Validation of these proposed metrics is given in chapter 5. The results and discussion section is followed by the conclusion.
and then future recommendation are given in chapter 5. At the last references are given.
Chapter No. 2

Literature Survey

According to Norman E. Fenton and Shari Lawrence Pfleeger (1997), Software engineering offers a controlled, systematic, calculated and engineering approach similar to other fields of engineering where as theoretical foundation is the focus of the computer science. “You cannot control what you cannot measure”. This Quote is the famous phrase of DeMarco. Another phrase about measurement by Galileo Galilei (1564-1642), “What is not measureable make it measureable” [1].

Thomas J. McCabe (1976) in this research paper a cyclomatic complexity measurement metrics has been defined. This metric measures the graph theoretic complexity and explains that to manage or control the complexity of a program by using this metric measurement. The limitation of this metric is that the complexity value for a small and large scale programs comes out to be same in some situations [2].

Chidamber and Kemere (1994) proposed a set of six metrics to discover some code and design flaws in object oriented software design [3]. These metrics are listed in the Table 1. As a whole, these six metrics offer senior managers & designers, who might not entirely know the complete design particulars of a software application, with an insight of the reliability of the design. In order to address the structural and architectural consistency of the entire application the managers and designers can utilize it as a tool. Also to find out how much rigorous testing will be needed for
certain areas and which areas are candidates for redesign, these six metrics are very useful. Potential design flaws can be discovered early in the software application development life cycle by using these metrics. There is another advantage of using this metric suite is that the one can find the tradeoffs made between conflicting requirements in the design, such as to facilitate ease of testing and increased reuse. Also from the organizational point of view, a design can be chosen from various possible object oriented design for the same problem for the purpose of reducing cost, maintenance and testing during the entire life cycle of the application development. The main limitation of the CK metrics is that it only for the measurement of a general purpose object oriented designs of software applications and not for a real time environment application design. To measure a real time environment application design the time concept of task deadlines must be considered so metrics for real time environment application can be true measurement entities [3].

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Metric Abbreviation</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WMC</td>
<td>Weighted Method per Class is defined as count of sum of the complexities of all methods in a class.</td>
</tr>
<tr>
<td>2</td>
<td>RFC</td>
<td>Response for a Class is defined as the number of methods in the set of all methods that can be invoked in response to a message sent to an object of a class.</td>
</tr>
<tr>
<td>3</td>
<td>DIT</td>
<td>Depth of Inheritance tree is the maximum length from the node to the root of the tree and measured by the number of ancestral classes.</td>
</tr>
<tr>
<td>4</td>
<td>NOC</td>
<td>Number of Children is defined as number of immediate sub classes subordinated to a class in the class hierarchy.</td>
</tr>
<tr>
<td>5</td>
<td>CBO</td>
<td>Coupling Between Objects is defined as a count of the number of other classes to which it is coupled.</td>
</tr>
<tr>
<td>6</td>
<td>LCOM</td>
<td>Lack of Cohesion in Methods</td>
</tr>
</tbody>
</table>

Table 1: CK Metrics Suite [3].
F.B. Abreu and Walcelio Melo (1996) presented a metrics suit for measuring the Object Oriented Design. The MOOD (Metrics for Object Oriented Design) metrics set includes the following six metrics listed below [4].

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>Metric Abbreviation</th>
<th>Metric Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MHF</td>
<td>Method Hiding Factor</td>
</tr>
<tr>
<td>2</td>
<td>AHF</td>
<td>Attribute Hiding Factor</td>
</tr>
<tr>
<td>3</td>
<td>MIF</td>
<td>Method Inheritance Factor</td>
</tr>
<tr>
<td>4</td>
<td>AIF</td>
<td>Attribute Inheritance Factor</td>
</tr>
<tr>
<td>5</td>
<td>POF</td>
<td>Polymorphism Factor</td>
</tr>
<tr>
<td>6</td>
<td>COF</td>
<td>Coupling Factor</td>
</tr>
</tbody>
</table>

Table 2: MOOD Metrics Suite [4].

Method Hiding Factor [4] is mathematically defined as

\[
MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{M_d(C_i)} (1-V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)}
\]  

(2.1)

Where

\[
V(M_{mi}) = \frac{\sum_{j=1}^{TC} \text{is_visible}(M_{mi}, C_j)}{TC-1}
\]  

(2.2)

The total sum of the invisibilities of whole methods that are defined for a class or classes is the numerator of the method hiding factor (MHF). The percentage of the sum total of all classes from where the method under consideration is not visible is called the invisibility of a method. The sum total of all classes from where the underlying method is not to be visible is defined as the denominator of the method hiding factor [4].

The Attribute Hiding Factor [4] is defined as
The total sum of the present invisibilities of all attributes that are defined for a class or classes are the numerator of the Attribute Hiding factor. The total percentage of the all classes from where the underlying attribute is not visible is called as the invisibility of that attribute. The sum of all the attributes defined for the underlying system is the denominator of the attribute hiding factor [4].

\[
AHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{Ad(C_i)} (1 - V(A_{mi}))}{\sum_{i=1}^{TC} Ad(C_i)} \tag{2.3}
\]

Where

\[
V(A_{mi}) = \frac{\sum_{j=1}^{TC} \text{is_visible}(A_{mi}, C_j)}{TC - 1} \tag{2.4}
\]

According to F.B. Abreu and Walcelio Melo Method Inheritance factor is defined as [4]

The total count/sum of all the methods that are inherited for classes is the numerator of the Method Inheritance Factor (MIF). The denominator of the method inheritance factor is the total count of all the available classes for the underlying system [4].

\[
MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)} \tag{2.5}
\]

Where

\[
M_a(C_i) = M_d(C_i) + M_i(C_i) \tag{2.6}
\]

Attribute Inheritance Factor is defined as

The numerator of the Attribute Inheritance Factor is defined as the total/sum of all the attributes that are inherited in the classes of the underlying system. The denominator of the Attribute Inheritance Factor is defined as the sum/total count of all the classes that are available for the underlying system [4].
\[ AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_d(C_i)} \] 

(2.7)

Where

\[ A_d(C_i) = A_d(C_i) + A_i(C_i) \] 

(2.8)

Polymorphism Factor [4] is defined as

The numerator of the Polymorphism Factor is represented as the actual total count of the all different polymorphic situations that are possible in the underlying scenario.

For the implementation of a named method a message sent to a class \( C_i \) could be dynamically bound or statically bound. There can be as many morphos/shapes as many times a method under consideration is overridden. The maximum number of all the distinct polymorphic situations that are possible is represented as the denominator of the Polymorphism Factor. When all of the new defined methods in a class \( C_i \) are overridden in the total number of the classes that are derived from this class is the case that is under consideration [4].

\[ POF = \frac{\sum_{i=1}^{TC} M_{ol}(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]} \] 

(2.9)

Where

\[ M_d(C_i) = M_n(C_i) + M_o(C_i) \] 

(2.10)

Coupling Factor is defined as

The maximum number of couplings present in a system with number of TC classes is represented as the denominator of the Coupling Factor. The actual total number of the
couplings that are not imputable to inheritance is then represented by the denominator of the Coupling Factor \[4\].

\[
COF = \frac{\sum_{i=1}^{TC} \sum_{j=1}^{TC} is\_client(C_i,C_j)}{TC^2 - TC}
\]  \hspace{1cm} (2.11)

Amandeep Kaur and et al (2010) analyzed the CK and the MOOD metrics and pointed out that there are some flaws in these metrics e.g. whether the method to be inherited should be counted in derived classes or in base class, or in both. The other limitation is that in the case of real programming practice nested call-backs are very deep, whereas in CK metrics only one level is considered for nesting when data is collected to calculate Response for a Class. The distorted view of the underlying system is given by NOC definition because it only counts the sub-classes that are immediate, rather than all the class descendants as shown in the Figure 1 [5].

![Figure 1: Example showing the distorted view of NOC metric [5].](image)

The LCOM metric defined by CK has the limitation that it is not able to differentiate between the less cohesive class and the more cohesive classes. It can be seen from the Figure 2. The value of LCOM for Figure 2(a) and Figure 2(b) is same that is 8 as calculated in [5].
Similarly flaws in MOOD metrics has also been pointed out in [5] The flaw pointed in MIF is that if all the methods are inherited that the value should come out to be 100% but actually it is evaluated to be 66% in some cases. There is no need for separate metrics for methods and attributes because these are part of the class. When a class is extended the methods and attributes of the classes come as part of the package along with the classes. Hence a class level metric will be more effective and useful. If private methods are counted in a class then it will not give us any useful information because the encapsulation level of the class has no effect with this knowledge of method hiding factor. The value of POF comes out to be greater than 1 in some situations as shown in following calculation w.r.t the Figure 3 [5].

\[
\begin{align*}
Mo(P) &= 1, Mo = (Q)2, Mo(R) = 2 \\
(Mn(P) = 2 * DC(P) = 2) &= 4 \\
(Mn(Q) = 2 * DC(Q) = 0) &= 0 \\
(Mn(R) = 2 * DC(P) = 0) &= 0 \\
\text{Therefore, POF for subsystem"S"=} \\
(1 + 2 + 2)/(4 + 0 + 0) &= 5/4 (and 5/4 > 1)
\end{align*}
\]  

(2.12)
Houari A. Sahraoui et al (2000) in this research work potential flaws detection in the design had been investigated through the use of metrics for the purpose of suggesting correction transformations that may be useful. The title of this paper is actually in a question form about bridging a gap between design quality & automation. The conclusion of the research is that the answer could be Yes or No [6].

Phillip A. Laplante (2004) if the logical correctness of a system is based on the timeliness and correctness of outputs then this system is called as a real-time system. There are normally three kinds of timing restrictions on the tasks of a real time application. These are called soft deadline, hard deadline and firm deadlines. If a soft deadline is missed it will not lead to any problem only a degraded performance, e.g. web, ATM. The user will only be annoyed. In case a hard deadline is not fulfilled it will lead to a catastrophic failure, e.g. accident in a nuclear power plant. In case of a firm real-time application a few missed deadlines will not lead to total failure, but
missing more than a few may lead to complete and catastrophic system failure. There are many design methodologies proposed for real time system design. Some of them are briefly described. SA/SD, Structured Analysis and Structured Design, both of these techniques use Data Flow Diagrams and Module decomposition approaches. Because this methodology is data oriented and not control oriented it is not endowed to handle the real time system requirements. It is very difficult to describe concurrency of simultaneously executing processes in it. Other limitation of SA/SD is that because of the hardware dependence it is not an easy task to translate control flows into the code directly. If the hardware is changed at any state then these changes cannot be reflected, until a huge size is written again. The further consequence is that these changes then must be incorporated right into the first level of the data flow diagram for every process, any level coming in the sequence and at the last these changes have to be reflected into the final code. The real time extension of the SA/SD has also been proposed [7].

Figure 4: Relation between the control and the process model [7].
Rob Williams (2006) according to author there is no clear dividing line between a non-real time system and a real time system. Real-time systems are required to compute and deliver correct results within a specified period of time. Numerous software design methods use the Finite State Machines (FSM) as basic design tool, because FSMs can offer powerful support throughout the design & implementation stages of software development. The lack of expressive power of FSM when complex problems are dealt is often criticized as design tool. As the large systems are generally viewed as hierarchically, the FSM has no facility to handle these large complex systems. It enables people to manage separately all the lower level details. David Harel (1987) introduced the concept of statecharts that can represent the nested FSMs. In statechart model of the state machine each and every state can sequester another FSM that is internal. This state machine model has been used in this research work [9].

H. Gomaa (1984) DARTS Extended the SA/SD method to deal the requirements of the real time systems. In this methodology provided an approach in order to structure the system into concurrent tasks. Also defined interfaces between various tasks. The limitation of this approach is that it is focusing only to the concurrency [10].

Michael A. Jackson and John Cameron (2010) Michael A. Jackson and John Cameron developed Jackson System Development (JSD) which is a linear software development methodology [11]. Main goal was to map progress of the system to be modeled with the progress in the real world. JSD development procedure consists of six steps (step1: Entity/Action, step: 2 Entity Structures, step: 3 Initial Model, step: 4 Function, step: 5 System Timing step and: 6 Implementation) [13]. The problem is that the timing is considered only at the 5th step is JSD method and the concept of
classes is almost not utilized. The other limitation is that the decomposition of a system into tasks or modules in not handled [11] [12] [13].

C. W. Mercer & H. Tokuda (1990) ARTS targets the real time systems and it is an object oriented [15]. This methodology is based on the RTC++ which is real time extension of the C++ [14]. Timing encapsulation and data encapsulation are its key features. This methodology concentrates more on the low level design rather than behavior model regarding the high level design. It allows multithreaded and single threaded objects. Predictability is dealt better in multithreaded objects. It provides solution to the problem of priority inversion. Behavior of the objects in ARTS has no clear understanding that how it is modeled [15].

Hassan Gomaa (1993) COBRA (Concurrent Object-Based Real-Time Analysis) is a mix of concepts of object-oriented Analysis (OOA), Jackson System Development (JSD) [11] [12] [15] and Real Time Structured Analysis. It uses the notation of state diagram and Real Time Structured Analysis. For distributed environments COBRA has an advantage due to its support for decomposition approach. One drawback of COBRA are that it does not considers deadlines. Another drawback is deadlock free system is not guaranteed in event sequencing scenarios. This is because event sequencing scenarios are specific scenarios and not complete representation [16].

European Space Agency (ESA) (2010) HOOD method built with classes and objects as basic building blocks. The aim of HOOD is to construct Detail Design, Architectural Design and coding [17]. For software component development it is a good framework for gaining expertise in development integration of components with
various technologies and languages. It suited for large complex projects. The major benefits of HOOD are reusability, design clarity, maintainability, toolset support, maintainability and extensibility [18]. The limitation of this methodology is that it is a general purpose method and not targeted for dealing the requirements of the real time application design [17] [18].

M. Paludetto & S. Raymond (1993) this methodology extended the HOOD with PNO (Petri Net Objects). HOOD (Hierarchical Object-Oriented Design) is defined by European Space Agency [17]. The advantage of this methodology is that it deals almost the entire life cycle, i.e. it covers the life cycle from requirements to code. But limitations of this methodology are it does not deal with concurrency directly. Deadlines of objects are not tackled in design but are left as implementation challenge for the language [19].

A. Burns and A.J. Wellings (1994) HRT-HOOD (Hard Real Time Hierarchical Object Oriented Design) is adapted from HOOD [17] [18] for real time environments. Abstraction is the main focus of this methodology. Deadlines are better conceptualized due to abstraction. HRT-HOOD separates the high level design activity into sections. The first is physical design which addresses schedulability and timing from functional requirements and other constraints. The second is the logical design that is concerned with satisfying the functional requirements that can be made independently of the constraints enforced by the execution environment. HRT-HOOD supports five kinds of objects including Passive, Active, Protected, Cyclic and Sporadic. This methodology does not clarify if concurrency is supported when objects are assigned to physical processors and if threads are inside an object [20].
James Rumbaugh and et al (2011) OMT, Object Modeling Technique is an object modeling language intended for designing and modeling of software. It was developed roughly near 1991 by Rumbaugh and et al. The focus of this method was to build object-oriented systems and to incorporate object oriented programming concepts in the software development. According to Rumbaugh the main purpose of the modeling technique was to incorporate testing before developing any physical entities, complexity reduction, communication with clients and visualization. Three main types of models have been proposed by the OMT. These models are object models, dynamic models and functional models. The object model is the static representation of the underlying system domain. All the main concepts of classes, associations, aggregation and generalization along with their attributes and related methods or operations are incorporated [21].

The dynamic model as shown in Figure 5 is a state transition model view of the system under consideration. States and state transition from one state to another, the events that are responsible for the transition are the main concepts. Actions occurring are modeled inside states. Aggregation and generalization are the relationships that are predefined [21].

![Figure 5: OMT state diagram](image)

The limitation of the OMT methodology it is a general object oriented methodology and not a real time system design methodology.
Jurgen Ziegler and et. al (1995) OCTOPUS methodology is for handling the embedded real time systems. OCTOPUS extended the OMT [21] for catering interfaces and communication. This methodology uses statecharts for behavior modeling. Concurrency and deadline management are handled. Just like OMT inheritance is also supported by OCTOPUS. The drawback is that it does not support the full life cycle but only design and implementation phases [22].

Bran Selic (1992) ROOM (Real time Object Oriented Modeling) methodology uses two concepts i-e abstraction level and dimension. Based on the nature of the problem the dimension model partitions the system. The system is then portioned into three levels of abstractions i-e system level, concurrency level and detail level. For modeling concepts at highest level, system level abstraction is used, issues of parallelism are dealt at concurrency level and finally the implementation is the focus of the detail level abstraction. The model is refined iteratively at next level of abstraction. This methodology uses the concept of actors which accepts information units (messages) through ports. Actors and their messages are decomposed into groups hidden from higher abstraction levels. The formalization of this behavior of actors is done through statecharts and state machines [23].

P. Dapont, and et. al (1992) In this paper author claims that RTO (Real Time Objects) is suited for hard real time programming. This methodology does not allow concurrency between the objects. Objects are reactive in nature and single threaded and preemption of an action in execution is not allowed. To model the deadlines the theory of timed automata [28] and the concept of timed state statecharts is used. Individual objects have their own code for concurrency synchronization and no
centralized control. Messages are passed between objects asynchronously. Unexpected messages are discarded, deferred or passed to another object. Scheduling is handled through the idea of component (a collection of objects having similar requirements of timing). There is a special object associated with every component called a controller. In order to operate concurrently RTO uses a default standard controller and messages to other objects in the components are dispatched in FIFO order without concerning about the time. External events and physical system are encapsulated in driver objects [24].

*K.M Sacha* (1992) Transnet is another proposed extension to design methodologies for real time systems. To model the behavior and for verification this methodology uses Petri nets as in HOOD/PNO that as uses Petri nets. This methodology focuses not only on functionality of design but also concerns about deadlines, message passing and concurrency of the objects. Concurrence analysis and support for deadline management is the advantage of this methodology. The drawback of this methodology is that it only supports the specification and preliminary design steps of the life cycle model [25].

*A. Shah* (2003) OODM is an object oriented design methodology proposed specially for the systematic design of web applications. This methodology focuses on analysis and design phases of the web application design. This methodology is suitable for web application only and not for real time systems because it does not considers the task deadlines [26].
Arshad S. and A. Shah (2010) S-OODM extended the OODM [26] by introducing a security model for the design of web applications. This methodology is also suited for web application design but not suitable for real time application design [27].

Winston W. Royce (1987) two essential steps are common to every computer software project regardless of the complexity and its size. The first step is analysis and the second step is coding as shown in the figure below [29].

![Diagram](image)

**Figure 6: Two essential steps common to software development projects [29].**

Although these two steps are very important, but for a successful large software development project some additional steps are also required. For small program it may work but in case of big software projects these two steps will not be sufficient. With only these two steps chances of failure are more as the size of the project is increases. This is a pure waterfall model scheme because water can not flow up the hill at its own. Naturally water falls down the hill under gravity.
In software development the changes has to be incorporated as the project proceeds. The understanding about the software product increased with time. Therefore above model has to be modified with some iteration as shown in the figure below. As each step progresses the details about the system are enhanced. There details must be incorporated through an iterative process and modifications. The iteration is carried out when all the available information is documented, designed or coded. Testing should be included at every step.
Ian Sommerville (2011) the example of a plan driven software development life cycle model is the Waterfall Model. This model has been derived from general system engineering processes. The phases of this plan driven model are requirements analysis and definition, system and software design, implementation and unit testing, integration and system testing, operation and maintenance [30].
Pankaj Jalote (2008) Software process models are described briefly [31]. In waterfall model feasibility analysis is the first step. Requirements analysis and project planning begins follows after a successful feasibility study of the project. After completion of the requirements analysis phase the starts the design phase. When the design is completed the coding is started. After completion of the coding step different coding modules are integrated and tested. System is installed at client’s site when testing is done successfully. The operation and maintenance is the last phase of this model. The major advantage of the waterfall model is the simplicity. A large a complex task is split into a sequence of simple and straight steps/phases, every phase has its own logically different area of concern. There are some limitations of this model. The requirements are finalized early before the design phase. For a new system this model
is not suitable due to ever changing demands. The hardware is also finalized early, so due to long development time it is possible that hardware may become obsolete. The risk of failure is more because user knows at last step what is developed for him. Also formal documentation is required at the end of each phase [31].

In prototyping model requirements kept flexible before the start design and coding phase. Requirements are understood by developing a throwaway prototype. All possible requirements available at that time are utilized to develop this prototype. This prototype is developed by going through all the phases of the software development life cycle model. The client can use this prototype to understand the software application that will be delivered. The requirements are modified again after client’s suggestions and a stable requirement list is documented whose frequency of change is less [31].

![Figure 10: Prototyping Model [31].](image)

The main theme behind interactive development model is that different versions of the software application be developed step by step. Every version of the application incorporates new functionality. This cycle is repeated until a final full software application is implemented [31].
Linda Northrop (2006) the objective of software architecture is to provide the mainly elementary foundation for convince about design decisions and set up important work breakdown structures [33].

J. Wang & H. Chen (1993) the harel’s statechart [8] with timer reduces complexity. Consider the design example of the Coolant spill of nuclear reactor. This simple design is drawn using the Statechart+ solution of the Wang & Chen along with the Timedstate statechart solution [35].

Figure 11: Iterative Developmental Model [31].

Figure 12: State chart + Solution [35].
The Wang & Chen solution uses both timed transitions and timed states that leads to upper time bound ambiguities and the lower bound time ambiguities. To make it simple just use the timed states and timer the solution becomes simple and upper and lower bound time ambiguities are resolved. The same solution is presented as follows. This simple modification can resolve the ambiguities that arise due to inheritance of state chart objects having timing information on transition edges.

![Timed state chart solution](image)

Figure 13: Timed state chart solution

*Wikipedia* (2011) the single or multiprocessor architecture determines the true degree of parallelism. The hardware selection depends on the requirements specification, e.g. in case of concurrent processing requirement a single processor will only give pseudo parallel execution not the true parallel execution as in case of multiprocessor system [34]. This phase is also very important because numerous hardware and software are available in the markets that claim real time capabilities. It is not feasible to design the complex real time applications from scratch. There are dedicated hardware(s) for data acquisition from the real time environment and control hardware. There are also real
time operating systems available. The very first examples of real-time operating systems for a large-scale projects were, the IBM and American Airline’s TPF (Transaction Processing Facility) which was built for the Sabre. At present the top most known, real-time operating systems are Windows CE, OSE, RTLinux, LynxOS, QNX, VxWorks [36].

There is misconception that high level languages are not recommended for stringent timing requirements and low level languages like assembly is recommended. But it is one of the misconceptions pointed out in [37] & [38]. According to [24] the actual point of concern should be that if the language you chose allows access to low level hardware interface without the extra run time support penalty.

It is also necessary to take into account the implementation languages like RTC++ or Ada 9X may do the job. Along with the general purpose languages like, Ada, Modula, & Java there are also special purpose languages for real time systems like Esterel, Lustre, Signal and Statecharts [40]. The implementation may be done in any of the modeling language. The UML (Unified Modeling Language) is a popular choice for modeling of real time system [41]. It will help you measure metrics automatically through available tools [42]. The modeled design in UML can be implemented using real time programming languages as mentioned above or vendor specific software packages. The advantage of using vendor software package is that development time is short and real time operation as offered by most of the vendors can be guaranteed.

There are lots of vendors that have mastered the technology over many years for real time solutions. So it will be time consuming and risky to start the implementation from scratch.
SDMetrics (2011) is a software design metrics tool for the UML diagrams. UML is becoming the favorite software design tool for most of the designers [42]. This tool helps to calculate various metrics automatically and also offers various types of plots like histograms and Kiviat diagrams. Figure below is a histogram plot of the DIT. A high value of DIT means that classes which inherit from many classes are harder to understand. It is also discovered that, classes with high DIT value might not be proper specializations of all of their predecessor classes [3].

Figure 14: Histogram plot for DIT [42].

<table>
<thead>
<tr>
<th>Stat.</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>2</td>
</tr>
<tr>
<td>99th</td>
<td>2</td>
</tr>
<tr>
<td>97.5th</td>
<td>2</td>
</tr>
<tr>
<td>95th</td>
<td>2</td>
</tr>
<tr>
<td>90th</td>
<td>2</td>
</tr>
<tr>
<td>75th</td>
<td>1</td>
</tr>
<tr>
<td>50th</td>
<td>0</td>
</tr>
<tr>
<td>Min</td>
<td>0</td>
</tr>
<tr>
<td>Count</td>
<td>100</td>
</tr>
<tr>
<td>Mean</td>
<td>0.574</td>
</tr>
<tr>
<td>StdDev</td>
<td>0.811</td>
</tr>
</tbody>
</table>

Figure 14 is a histogram plot of the DIT. A high value of DIT means that classes which inherit from many classes are harder to understand. It is also discovered that, classes with high DIT value might not be proper specializations of all of their predecessor classes [3].

Figure 9 is a histogram plot of Number of operations, also known as WMC (Weighted method complexity). It is kept in mind that unity complexity is assigned to each operation.
Figure 15: Histogram plot of the NumOp (WMC) [42].

The graph shown in Figure 16 is a Kiviat diagram, showing the values of all metrics for the module ExpressionNode. Each axis (or ray) of the Kiviat diagram represents one metric, as labeled in the diagram. The range of the metric is the scale of all axes: the lowest value is to be found in the center, the highest value at where the axis ends. The scaling of all the axes is linearly done.

Figure 16: Kiviat Plot for the rule ExpressionNode module [42].

The table on the right side of the Figure 15 shows, different percentiles values against the no. of metrics whose values go beyond the percentile for the selected module.
Metrics with higher values point to inferior quality, a module design must be considered critical for which most of metric values for the module are in the higher percentiles (e.g., 90th, 95th).

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
<th>Value</th>
<th>Category</th>
<th>Severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDMetrics.com/sdmetrics.ap</td>
<td>SUP2</td>
<td>SDMetrics.com/sdmetrics.style</td>
<td>Style</td>
<td>0-med</td>
<td>Package violates the Stable Dependencies Principle (SDP).</td>
</tr>
<tr>
<td>SDMetrics.com/sdmetrics.ap</td>
<td>SUP2</td>
<td>SDMetrics.com/sdmetrics.style</td>
<td>Style</td>
<td>0-med</td>
<td>Package violates the Stable Dependencies Principle (SDP).</td>
</tr>
<tr>
<td>SDMetrics.com/sdmetrics.ap</td>
<td>SUP2</td>
<td>SDMetrics.com/sdmetrics.style</td>
<td>Style</td>
<td>0-med</td>
<td>Package violates the Stable Dependencies Principle (SDP).</td>
</tr>
<tr>
<td>SDMetrics.com/sdmetrics.ap</td>
<td>SUP2</td>
<td>SDMetrics.com/sdmetrics.style</td>
<td>Style</td>
<td>0-med</td>
<td>Package violates the Stable Dependencies Principle (SDP).</td>
</tr>
<tr>
<td>SDMetrics.com/sdmetrics</td>
<td>CapitalizationSDMetrics</td>
<td>Naming</td>
<td>0-low</td>
<td>Package name has upper case letters.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 17: Design rule violation at package level [42].

Figure 17 is a discovery of the design rule violation by the SDMetrics tool at the package level design. Although the severity of the design rule violation is of medium and low level, still these discovered violations may help the designers to revisit their design and try to correct those errors early and before the implementation.

Summary

Object oriented design metrics provide a way to measure OO designs. The most important metrics in literature known as CK and the MOOD and some limitations of these metrics are surveyed. McCabe’s complexity metric measures the graph theoretic complexity and explains that to manage/control the complexity of a program by using this metric measurement. Also some object oriented design methodologies have been surveyed. Our survey revealed that most of the methodologies are lacking measurement aspect. Like other fields of engineering measurement in software engineering and design is also gaining importance. It is also found out to the best of our knowledge that the design metrics for real time application design have not been yet defined. A tool for automatically measuring the real time application designs in UML has also been surveyed. This tool helps to measure the design for different kinds of metrics if the real time environment application is designed using UML.
Chapter No. 3

Metrics Driven Framework Model for the Design of Real Time Environment Applications (MDMF)

3.1 Proposed Design Model for Real Time Applications

The waterfall model has been extended for real time environment applications. The architecture of the proposed design framework model for the real time applications is given as under.

![Diagram of MDMF model](image-url)

Figure 18: MDMF the proposed framework model for real time application.
The description of the steps involved for the design of real time applications is given as under.

1. **Real Time System Specification**
   1.1. Collect specification with deadlines of all tasks according to IEEE standard SRS document.
   1.2. **WHILE** there are objects in SRS
       1.2.1. Identify all the sensor objects.
       1.2.2. The control objects.
       1.2.3. And the actuator objects
   1.3. **END WHILE**

2. **The RTS Design**
   2.1. **Static Model**
       2.1.1. Using Object Oriented technique build class and component level.
       2.1.2. Use inheritance by inheriting all feature of a class along with the task deadlines.
       2.1.3. Overriding of deadlines is also allowed.
   2.2. **Behavior Model**
       2.2.1. For behavior model use Harel’s statechart.
       2.2.2. Separate the concurrent objects with dotted lines.
       2.2.3. Deadlines classification
           2.2.3.1. If object is not in inheritance hierarchy put deadlines on transition edges as timing condition. Allow upper and lower bound time interval values.
           2.2.3.2. If object is in the inheritance hierarchy put deadlines as timers inside the statechart. Upper and lower bound time interval values are not allowed.
           2.2.3.3. Repeat step 2.2.3.1 until all are classified.

3. **Measurement by Design Metrics**
   3.1. **While** there are modules to measure
       3.1.1. Measure with defined Novel Real Time System Design Metrics
       3.1.2. Measure modules cohesion with SDCF & HDCF.
       3.1.3. Measure complexity with ODCF & OCF.
       3.1.4. Measure the message traffic with MET
       3.1.5. Measure the early decomposition with EDF
       3.1.6. Measure the predictability with DBPF
       3.1.7. Measure the Life Cycle support factor by LCSF metric.
   3.2. **END While**
   3.3. If measured values are satisfactory then
   3.4. Go to Implementation else
   3.5. Go to design step and review the real time system design.

4. **Real Time System Design Implementation.**
   4.1. **Hardware Software Selection**
       4.1.1. If available real time software guarantees the fulfillment of task deadlines get buy it to save time.
       4.1.2. Else write customized software for the real time system application.
       4.1.3. End if

5. **Integration and System Testing**
   5.1. While there are modules.
       5.1.1. Integrate all modules
       5.1.2. Test all modules for task deadlines.
   5.2. **END While**

6. **Operation and maintenance.**
   6.1. Deploy the software at client site.
   6.2. Check each input and output for timing constraints.
   6.3. Modify/change if required.
3.2 Description of the Proposed Model

The model proposed is described in detail in following steps. From real time application specification step to deployment step, details of the input, processing and output delivered by each step is discussed in the following.

Step1: Specification

**Input:** Requirements gathered for the underlying real time application is the input to this step.

**Processing:** This step starts with gathering all possible information about the real time application from the client. The operational environments of application and its functionality, constraints and tolerance levels are collected. Also the functional and performance requirements for mechanical and electrical hardware, sensors and actuators and components related to control and operation of the real time environment application are collected. From this information the system level specification are extracted first. After that the module level specifications are extracted from system level specifications. The functional and performance requirements are the next to be extracted from the module level specifications. Then these functional and performance requirements are broken down into three categories of requirements. These three categories are soft deadline functional requirements, hard deadline functional requirements and non real time deadline functional requirements. At the last all these requirement are written as a formal document which is called SRS or the system requirement document.
Figure 19: Specification Phase

**Output:** SRS (system requirement specification) Document.

**Step 2: Object Identification**

**Input:** SRS (System requirement Specification) from step1.

**Processing:** Real time object identification and classification.

System requirement specifications are broken-down into groups of appropriate objects. From the specifications the objects relating to sensors, actuators and control are identified. The control objects are placed in-between the sensor objects and the actuator objects. Real-Time Objects (RTOs) and are then classified as sensor class objects, control class objects and actuator class objects. This classification of real time objects into different categories is formally documented.
Output: A formal document showing all the identified real time objects classified in three categories.

Step 3: Decomposition

Input: Document of Real time objects as identified in step2.

Processing: Decomposition based on inheritance.

The objects that are to be inherited and the objects that are not to be inherited are identified. If an object is to be inherited, the deadline will be included inside the statechart as timer condition as shown in figure below.

Figure 21: Statechart with timer
The statechart with timer will be represented as rounded rectangle with process mentioned inside statechart along with timing condition as a timer. If an object is not to be inherited than its deadline will be indicated as a condition on transition.

![Figure 22: Deadline as a transition condition](image2)

Then decompose the Concurrent states statechart is decomposed and divided by a dotted line [43].

![Figure 23: Concurrent State Representation using Statechart [41].](image3)

**Output:** Formal design document with objects decomposed based on inheritance.

**Step 4: Integrated Real Time Application Design**

**Input:** All formal Documents from step1 to step3.

**Processing:** The formal document delivered in previous step is an input for this step. Subsystem components are integrated in a hierarchy.

**Output:** Integrated real time application design document.
Step 5: Measurement

**Input:** The formal design document from step 4.

**Processing:** The proposed design metrics framework model is used to measure the real time application design. Analysis is carried out to decide about components those required revision. These components are measured with the help of proposed measurement framework model as shown in the figure below.

![Figure 24: The proposed measurement framework model.](image)

Calculate the values of the following metrics for the real time application design. Measurement before implementation is the main purpose to define these design metrics. Measurement at design level gives an additional quality control, reduction in effort and cost of development.
Metrics for Measurement

The following proposed design metrics for real time environment application can be used for measurement.

1 Soft Deadline Cohesion Factor

It is defined as the ratio of the classes having soft deadlines to the total no. of classes having soft, hard, overridden deadlines.

\[
SDCF = \sum_i^n \sum_j^m \frac{c_{sij}}{c_{sij}+c_{hij}+c_{oij}} \tag{3.1}
\]

Where

\( n \) = Total no. modules constraint by timing restriction

\( m \) = Total no. classes per module constraint by timing restriction

\( c_{sij} \) = ith Class in the jth module with soft deadlines.

\( c_{hij} \) = ith Class in the jth module with hard deadlines.

\( c_{oij} \) = ith Class in the jth module with overridden deadlines.

This factor tells about how cohesive the modules in terms of soft deadlines are. Higher the factor mean the system modules may be given less concentration because of error tolerance level is more in that module.

2 Hard Deadline Cohesion Factor

It is defined as the ratio of the classes having hard deadlines to the total no. of classes having soft, hard, overridden deadlines.

\[
HDCF = \sum_i^n \sum_j^m \frac{c_{hij}}{c_{sij}+c_{hij}+c_{oij}} \tag{3.2}
\]
Where

\[ n = \text{Total no. modules constraint by timing restriction} \]
\[ m = \text{Total no. classes per module constraint by timing restriction} \]
\[ C_{sij} = \text{ith Class in the jth module with soft deadlines.} \]
\[ C_{hij} = \text{ith Class in the jth module with hard deadlines.} \]
\[ C_{oij} = \text{ith Class in the jth module with overridden deadlines.} \]

This factor also tells about the cohesive nature of modules in relation to hard deadlines. Higher the factor means the system modules may be given more concentration because of error tolerance level for these modules are zero.

3 Overridden Deadline Class Factor

It is defined as the ratio of the classes having overridden deadlines to the total no. of classes having soft, hard, overridden deadlines.

\[ \text{ODCF} = \sum_i^n \sum_j^m \frac{C_{oij}}{C_{sij} + C_{hij} + C_{oij}} \]  

(3.3)

Where

\[ n = \text{Total no. modules constraint by timing restriction} \]
\[ m = \text{Total no. classes per module constraint by timing restriction} \]
\[ C_{sij} = \text{ith Class in the jth module with soft deadlines.} \]
\[ C_{hij} = \text{ith Class in the jth module with hard deadlines.} \]
\[ C_{oij} = \text{ith Class in the jth module with overridden deadlines.} \]

This factor is the most important because it tells about deadline related ambiguities that lies in those modules having high ODCF value. These ambiguities are due to the
inheritance of deadlines. Most of the concentration must be given to those modules having high ODCF.

4 Overridden Complexity Factor

The overriding factor is defined as the ratio of overridden classes to the total no. of classes having hard deadlines.

\[
OCF = \frac{\sum_{c_0=1}^{n_0} c_0}{\sum_{c_h=1}^{n_h} c_h} = \frac{ODCF}{HDCF}
\]

This factor tells the overall trend the module towards soft or the hard real time approach. A value less than 1 means timing constraints have to be met all cost.

5 Message Exchange Factor

No. of exchanged messages considered per second between project partitions.

\[
MEF = \frac{\sum_{i=1}^{n_e} m_i}{T}
\]

Higher the MEF more critically that module must be analyzed. It means that if the message traffic between the different modules is very high then the deadlines may not be guaranteed.

6 Early Decomposition Factor.

The Early Decomposition Factor is defined as the ratio of no. of project partitions to the project stage no. times the message exchange factor.

\[
EDF = \left(\frac{\text{No. of partitions of Project}}{\text{Project Stage No}}\right) \times (\text{Message Exchange Factor})
\]

Mathematically it is represented as

\[
EDF = \frac{N_p}{S_n} \times \frac{\sum_{i=1}^{n_e} m_i}{T}
\]
If early portioning into sections for a large system is done then it could lead to a poor design if at a later stage it is found out that message traffic between different sections of the system will consume enormous amount of resources.

7 Deadline based Predictability Factor

The Deadline based Predictability Factor is defined as ratio of the run time resource requirement for a task to the run time resource requirement for all the tasks plus total no. of multithreaded objects.

Mathematically it is represented as

\[
\text{DPF} = \frac{\sum_{b=1}^{n_{\text{tasks}}} \text{RTS}_{\text{task}}}{\sum_{b=1}^{n_{\text{tasks}}} \text{RTS}_{\text{all tasks}}} + \sum_{i=1}^{n} \text{Obj}_{\text{mt}}
\]  

Ideally the first factor should be than 1 and practically be close to zero. The second factor should be more than 1, because multithreading increases the predictability [15].

8 Life Cycle Support Factor

LCSF is the ratio of no. of phases having support for deadlines to the total no. of phases on the life cycle plus one.

\[
\text{LCSF} = \frac{\text{No.of phases having deadline support}}{(\text{Total No.phases in the life cycle}) + 1}
\]  

Every methodology has support for Software life cycle in some phases. If this factor is equal to 1 this means that the methodology supports the entire life cycle beyond the code release and into the code maintenance.

Output: Integrated real time application design measurement analysis.
Step 6: Hardware Software Selection

**Input:** Detail design document from step 5.

**Processing:** Size is calculated to decide about the selection of software and hardware. For example low level languages are required for small size applications. For large scale real time applications dedicated hardware and software packages are required e.g. PLC and ladder logic.

![Diagram](image)

**Output:** Hardware software selection document.

Step 7: Implementation

**Input:** All deliverables or documents from previous step.

**Processing:** The deliverables/documents produced during the requirements phase, design phase and implementation is done using hardware and software selected in step 6. The extensive amount of testing on the early system versions during this step is performed. This will remove the anomalies in the test cases.
Output: Real time application software and related user manuals or training documents.

Step 8: Deployment

Input: All deliverables from step 7.

Processing: Deployment plan is discussed with the supporting staff. Proper modifications are carried out if so required. Installation and onsite testing is the next step. Training is also an important activity before handing over the application.
**Output:** Real time application acceptance document.

**Step 9: Maintenance**

**Input:** All formal documents from step 1 to 8.

**Processing:** Incorporate all the changes required by client. Deadlines of the tasks must be guaranteed while incorporating the required changes.

**Output:** Formal modification documents with version no.
4.1 Introduction

The real time application design framework model driven by the software metrics is proposed in this research work. To validate this metric driven framework measurement and analysis has been carried out on the real time application as a case study. A weapon delivering application design has been selected for the purpose of measuring the proposed design metrics. Bruce Powel Douglass [43] used this case study for explaining real time UML (Unified Modeling Language). This is a real time application design of a weapon firing and control system. The purpose of this application is to hit a target as accurately as possible by using the state of the art technology and GPS (Global Positioning System) communication. Modern software and hardware tools help to achieve successful target hitting with reasonable accuracy tolerance. The global positioning system is used to automatically acquire target’s global position coordinates. The targeting system decides whether the object to be targeted is a threat or not. If the targeting system points the object as a potential threat, the fire and control mechanism takes action based on global position coordinates acquired from position module. The case is useful as task deadlines are considered in it [43].

Weapon delivery system modules is given below
4.2 Problem Statement

Weapon delivery system is a real time target tracking, sensing and missile firing control system. There are different modules in the design of real time missile firing and control application. These modules in this real time environment application design are listed below.

- Track Module.
- Position Module.
- Velocity Module.
- MultiSensorTrack Module.
- MissileTransaction Module.
- FireControl Module.
- TargetingSystem Module.
- FlightPath Module.
- Missile Module.

- **Track Module**: tracks the target by using the Global Positioning System coordinates evaluated by the Position Module and the velocity information from the Velocity Module.

- **Position Module**: position module measures the latitude, longitude, time of measurement and date of measurement according to Global Positioning System.

- **Velocity Module**: Then based on longitude and latitude values, the target velocity is sensed in case it is a moving target. The velocity for a stationary target is zero. Zero velocity increases the probability of hitting a target successfully.

- **MultiSensorTrack Module**: MultiSensorTrack module tracks the target. If target is a threat then the position module calculates the latitude and longitude coordinates of the target according to global positioning system. This information is given to TargetingSystem and the flight path module.

- **MissileTransaction Module**: MissileTransaction module controls the delivery of missile and also updates its missile inventory if missile is fired towards the target.

- **FireControl Module**: this module decides whether to fire the missile or not based on the information provided by the TargetingSystem module and the MissileTransaction Module.

- **TargetingSystem Module**: Controller of the targeting system decides whether the target is a threat or not. If the target is potential threat then it activates the fire control module.
• **FlightPath Module:** Based on the Global Positioning System coordinates the actual flight path of the missile is evaluated and provided to the MultiSensorTrack module. The projected flight path is a modification in the actual flight path according to the changing atmospheric conditions like temperature, pressure, wind speed etc.

• **Missile Module:** Based on the information provided by the FlightPath module and the MissileTransaction module the missile module delivers the missile to MultiSensorTrack module which guides the missile towards the target.

### 4.3 Measurement Phase

In this phase the measurement is carried out on the design of the above mentioned real time application case study. The proposed metrics SDCF (Soft Deadline Cohesion Factor), HDCF (Hard Deadline Cohesion Factor), OCF (Overridden Complexity Factor), MEF (Message Exchange Factor), EDF (Early Decomposition Factor) and DBPF (Deadline based Predictability Factor) are measured for the design of the Weapon Delivery System.

### 4.4 Behavior Model Evaluation

Behavior model shows how the real time objects react dynamically when some data is given as inputs to them. In other words the behavior model is actually a description of objects in action.

The behavior model of the weapon delivery design system is given below
4.4.1 Soft Deadline Cohesion Factor Metric Evaluation

The number of classes having soft deadlines in the above model is identified which is one. The total numbers of classes having soft and hard deadlines is counted and are put in the SDCF Metrics and the SDCF value is as under.

\[
SDCF = \frac{1}{1 + 1 + 1 + 1 + 1 + 1 + 1 + 1}
\]

\[
SDCF = \frac{1}{8}
\]

\[
SDCF = 0.125
\]
The value for this factor is towards lower side meaning that the this module has 12.5% soft deadline class objects and the module design can be considered as tilting towards a soft real time class of applications. This indicates that the predictability of the module is towards higher side.

4.4.2 Hard Deadline Cohesion Factor Metric Evaluation

The number of classes having hard deadlines in the above model is identified which are three. The total numbers of classes having soft and hard deadlines is counted and are put in the HDCF.

\[
HDCF = \frac{1 + 1 + 1}{1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1}
\]

\[
HDCF = \frac{3}{8}
\]

\[
HDCF = 0.375
\]

The value for this factor is 37.5%, which is on lower side (less than 50%) which means that this module has 37.5% hard deadline class objects and the module design can be considered as less tilting towards a hard real time class of applications. Lower the HDCF value indicates higher predictability of meeting task deadlines where as higher HDFC value indicates that the predictability is hard to guarantee.

4.4.3 Overridden Deadline Class Factor Metric Evaluation

There are no class objects which have overridden soft deadlines with the hard deadlines. Therefore number of classes having overridden deadlines in the above model is identified as zero. The total numbers of classes having soft, hard and
overridden deadlines is counted and are put in the ODCF Metrics and value obtained is as under.

\[
ODCF = \frac{0}{1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1}
\]

\[
ODCF = 0/8
\]

The ODFC factor for this module is zero. This indicates that module’s complexity is zero.

**4.4.4 Overridden Complexity Factor Metric Evaluation**

The values of the HDCF & ODCF are put in the OCF metrics and the value obtained is as under.

\[
OCF = 0/0.375
\]

\[
OCF = 0
\]

Zero value indicates that there is no overridden complexity factor for this module.

**4.4.5 Message Exchange Factor Metric Evaluation**

The total number of messages exchanged between the different project partitions is counted which is 23 and are put in the MEF metric. MEF value obtained is as under.

\[
MEF = \frac{23}{1000}
\]

\[
MEF = 0.023 \text{ msg/ms}
\]
The no. of messages exchanged per millisecond is not on the higher side for this module. The system resources are sufficient to ensure the predictability of all the tasks to be completed in specified time limit.

4.4.6 Deadline Based Predictability Factor Metric Evaluation

By using the concurrent state thread with timing diagram shown in Figure 28, the number of multithreaded objects is counted which is 2. The runtime resource requirements for all the tasks are counted and are put in the DBPF metric and value is obtained as under.

\[ DBPF = \frac{1}{9} + 2 \]

\[ DBPF = 2.11 \]

The value 2.11 indicates that the predictability of the guaranteeing a task deadlines is normal. But this value should be higher. To increase the value of this metric the number of multithreaded objects should be increased. The run time resource required for the task should always be less than the total number of recources required for all the tasks. This condition is satisfied in this case but the number of multithreaded objects is on lower side.

Summary of the measured metric values for the weapon delivery system design is given in Table 3. The soft deadline class cohesion factor value is less. The requirement is that the cohesion should be large. If the modules are cohesive than those modules can be given to different programmers. It will lead to parallel
development. Ideally a high cohesion and less coupling is the requirement for any software design [1].

4.5 Component Model Evaluation

For evaluating the component model consider the Figure 27. This model has been evaluated for how early the decomposition has been performed for the weapon delivery real time application design.

4.5.1 Early Decomposition Factor Metric Evaluation

The project stage number is identified which is 2. The total number of project partitions in weapon delivery is counted and are put in EDF metrics and the EDF value is as under.

\[
EDF = \frac{11}{2} \times \frac{23}{1000}
\]

\[
EDF = 5.5 \times 0.023
\]

\[
EDF = 0.1265
\]

The value of this metric is normal and it is not too early to partition the project. It will not consume too much system resources and the decision to partition at this stage is normal. A lower value means that project is partitioned late in the software development life cycle model. The value 12.65% is normal so the partitioning is neither late nor early.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Result/Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDCF</td>
<td>0.125</td>
<td>Only 12.5% modules are cohesive with reference to soft deadlines.</td>
</tr>
<tr>
<td>HDCF</td>
<td>0.375</td>
<td>37.5% modules are cohesive with reference to hard deadlines.</td>
</tr>
<tr>
<td>ODCF</td>
<td>0</td>
<td>No Complexity</td>
</tr>
<tr>
<td>OCF</td>
<td>0</td>
<td>No Complexity</td>
</tr>
<tr>
<td>MEF</td>
<td>0.023 msg/ms</td>
<td>Message Traffic is acceptable</td>
</tr>
<tr>
<td>EDF</td>
<td>0.1265</td>
<td>Acceptable</td>
</tr>
<tr>
<td>DBPF</td>
<td>2.11</td>
<td>Higher the better</td>
</tr>
</tbody>
</table>

Table 3: Metric values for the weapon delivery design.

4.6 Case Study of Design Methodologies Comparison

4.6.1 Introduction

This case study has been selected to carry out measurement of the Life Cycle Support (LCSF) Metric. Real time application design methodologies have been surveyed in chapter number two. From that survey a comparison among the real time application design methodologies has been extracted. Methodologies are evaluated for deadline Support in different phases of the Software Development Life Cycle and LCSF metric is measured for each methodology.

4.6.2 Problem Statement

An ideal real time application design methodology is one that takes into account the task deadlines in all the phases of the Software Development Life Cycle Model. Most of the real time application design methodologies surveyed do not support the task deadlines in different phases of the SDLC. Some support one phase, some support two, three or four. There is a need to evaluate these methodologies for this support.
For this purpose the metric SDLC has been proposed. Measure the SDLC metric from the comparison study of the real time application design methodologies.

### 4.6.3 Evaluating the LCSF Metric

For every methodology the support for deadlines is indentified in each phase of the software development life cycle model. In this calculation each ‘N’ is given 0 (zero) value. It means that the methodology has no deadline support in that phase. Each ‘Y’ is given value 1 (one) which indicates that the methodology has deadline support in that phase. The numbers are put in LCSF metric and the LCSF value for each methodology is evaluated as under.

\[
LCSF_{JSD} = \frac{N + Y + N + N + N + N}{6 + 1} = \frac{0 + 1 + 0 + 0 + 0 + 0}{7} = 0.142857
\]

\[
LCSF_{ARTS} = \frac{Y + Y + Y + N + N + N}{6 + 1} = \frac{1 + 1 + 1 + 0 + 0 + 0}{7} = 0.428571
\]

\[
LCSF_{COBRA} = \frac{N + N + Y + N + N + N}{6 + 1} = \frac{0 + 0 + 1 + 0 + 0 + 0}{7} = 0.142857
\]

\[
LCSF_{HOOD/PNO} = \frac{N + N + Y + N + N + N}{6 + 1} = \frac{0 + 0 + 1 + 0 + 0 + 0}{7} = 0.142857
\]

\[
LCSF_{HRT-HOOD} = \frac{Y + Y + Y + Y + N + N}{6 + 1} = \frac{1 + 1 + 1 + 1 + 0 + 0}{7} = 0.571429
\]

\[
LCSF_{OCTOPUS} = \frac{N + Y + Y + N + N + N}{6 + 1} = \frac{0 + 1 + 1 + 0 + 0 + 0}{7} = 0.285714
\]
\[ LCSF_{OMT} = \frac{N + N + Y + N + N + N}{6 + 1} = \frac{0 + 0 + 1 + 0 + 0 + 0}{7} = 0.142857 \]

\[ LCSF_{ROOM} = \frac{Y + Y + Y + N + N + N}{6 + 1} = \frac{1 + 1 + 1 + 0 + 0 + 0}{7} = 0.428571 \]

\[ LCSF_{RTO} = \frac{N + N + Y + N + N + N}{6 + 1} = \frac{0 + 0 + 1 + 0 + 0 + 0}{7} = 0.142857 \]

\[ LCSF_{TRANSNET} = \frac{N + N + Y + N + N + N}{6 + 1} = \frac{0 + 0 + 1 + 0 + 0 + 0}{7} = 0.142857 \]

\[ LCSF_{MDFM} = \frac{Y + Y + Y + Y + N + Y}{6 + 1} = \frac{1 + 1 + 1 + 1 + 0 + 1}{7} = 0.7143 \]

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Specification</th>
<th>Design</th>
<th>Implementation</th>
<th>Testing &amp; verification</th>
<th>Deployment</th>
<th>Maintenance</th>
<th>Phases having Support</th>
<th>LCSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSD</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>1</td>
<td>0.1429</td>
</tr>
<tr>
<td>ATRIS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>3</td>
<td>0.4286</td>
</tr>
<tr>
<td>COBRA</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>1</td>
<td>0.1429</td>
</tr>
<tr>
<td>HOOD/PNO</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>1</td>
<td>0.1429</td>
</tr>
<tr>
<td>HRT-HOOD</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>4</td>
<td>0.5714</td>
</tr>
<tr>
<td>OCTOPUS</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>2</td>
<td>0.2857</td>
</tr>
<tr>
<td>OMTs</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>1</td>
<td>0.1429</td>
</tr>
<tr>
<td>ROOM</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>3</td>
<td>0.4286</td>
</tr>
<tr>
<td>RTO</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>1</td>
<td>0.1429</td>
</tr>
<tr>
<td>Transnet</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>1</td>
<td>0.1429</td>
</tr>
<tr>
<td>MDFM</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>5</td>
<td>0.7143</td>
</tr>
</tbody>
</table>

Table 4: Life Cycle Support Factor for different methodologies.
A 3-D plot of the LCSF metric is shown in figure above. The fact has been established after analyzing these real time application methodologies that not a single methodology has full life cycle support for the deadlines. Most of the methodologies support deadlines only at the implementation phase or the programming language.

From the above figure it is clear that MDFM has the maximum support for deadline in different phases of the software development life cycle model. ARTS, HRT-HOOD, OCTOPUS & ROOM also have good support for the deadlines. Most of the real time application methodologies handle the issues related to task deadlines in the implementation phase. For example JSD, COBRA, HOOD/PNO, OMT, RTO and Transnet have deadline support only in one phase of the SDLC.
4.7 Discussion

The case studies were selected for evaluation of the proposed design metrics for the real time environment application. Measurement of the design metrics SDCF (Soft Deadline Cohesion Factor), HDCF (Hard Deadline Cohesion Factor), OCF (Overridden Complexity Factor), MEF (Message Exchange Factor), EDF (Early Decomposition Factor) and DBPF (Deadline based Predictability Factor) has been carried out for the design of the Weapon Delivery System case study. The comparison of real time application design methodologies conducted in literature survey chapter has been selected to carryout measurement of the Life Cycle Support (LCSF) Metric. First SDCF metric was evaluated and it was 0.125. This value indicates that it has 12.5% soft deadline class objects and the module design can be considered as tilting towards a soft real time class of applications. This means that the predictability of the module is high. The task deadlines can be guaranteed with the help available recourses. The HDCF value was evaluated next and that was 0.375. This values is also on lower side (less than 50%) which means that this module has 37.5% hard deadline class objects and the module design can be considered as tilting towards a soft real time class of applications. Higher the HDFC factor means that the predictability is hard to guarantee. The ODFC metric for this module is zero. It indicates that there are no classes in the hierarchy that have overridden soft deadlines with hard deadlines. OCF value is zero indicates that there is no overridden complexity factor for this module. MEF value of 0.023 msg/ms clearly indicates that number of messages exchanged per millisecond is high. Ensuring this much number of messages to be exchanged between objects is easy to guarantee with available system resources. DBPF is evaluated as 2.11. For module of this size the number of multithreaded objects is normal. To increase the predictability there should be as
much multithreaded objects as possible. Value of EDF metric is 12.65% which is normal and indicating that partitioning the project into modules is neither late nor early. It will not consume too much system resources and the decision to partition at this stage is right.

LCSF metric measured for different real time system design methodologies. The proposed methodology MDFM has the maximum support for deadline in different phases of the software development life cycle model. ARTS, HRT-HOOD, OCTOPUS & ROOM also have good support for the deadlines. Most of the real time application methodologies handle the issues related to task deadlines in the implementation phase only. JSD, COBRA, HOOD/PNO, OMT, RTO and Transnet have deadline support only in one phase of the SDLC.
Chapter No. 5

Conclusion & Future Recommendations

5.1 Conclusion

Measurement is the key to a successful software development project like the other fields of the engineering sciences. For controlling and managing a software development project, measurement aspect must be introduced at design stage so that design flaws could be identified before the implementation. This is important because a good quality design will lead to good quality software. The real time environment application software development is specially a difficult task due to a very strict timing requirements for the various real time tasks. The normal practice is that this task of handling deadlines is left as an implementation challenge for the developer at the target programming language level. The deadlines are not considered in the design stage of the real time environment application development life cycle. The survey of methodologies revealed the fact that there is not a single methodology that considered the deadlines in all the phases of the software development life cycle. The measurement aspect is also lacking in existing real time application design methodologies. So keeping in view these limitations of methodologies, a metrics driven framework model (MDMF) for real time environment applications is proposed. Measurement is introduced by proposing design metrics to evaluate the design of a real time application for identifying the areas where a more thorough concentration is required. Those areas or the modules are studied again and the design can be reviewed again to reduce complexity and improve cohesion, reduce message traffic between
different project partitions. Two Case studies are used to measure and validate the proposed design metrics. Design metrics SDCF, HDCF, ODCF, OCF, MEF, EDF and DBPF are measured for the design of the Weapon Delivery System case study.

The proposed design metrics are measured and results are discussed to elaborate how these metrics can improve the quality of the design by introducing the measurement early in the design phase before implementation. The advantage measurement at design level is reduction in effort, cost and time of development by identifying and rectifying errors early.

Measurement of the Life Cycle Support Factor (LCSF) is carried out on a comparison study of real time application design methodologies conducted in literature survey. LCSF metric measured for different real time system design methodologies. The proposed methodology MDFM has the maximum support for deadline in different phases of the software development life cycle model. ARTS, HRT-HOOD, OCTOPUS & ROOM also have good support for the deadlines. Most of the real time application design methodologies handle the issues related to task deadlines in the implementation phase only. JSD, COBRA, HOOD/PNO, OMT, RTO and Transnet have deadline support only in one phase of the SDLC. But ideally a methodology is required to have deadline support in the full life cycle model. The methodologies HRT-HOOD and MDMF are close to ideal.

The proposed design metrics are measured and results are discussed to elaborate how these metrics can improve the quality of the design by introducing the measurement early in the design phase before implementation. The advantage of these metrics is
reduction in effort, cost and time of development by identifying and rectifying errors early.

5.2 Future Recommendations

- Measurement is a good tool to control the progress and quality of any real time environment application. Unfortunately the measurement aspect has not been given the due importance by the software industry. Measurement should be introduced in a real time application life cycle as early as possible. It is recommended that it is introduced right from the requirement gathering step.

- Complexity introduced due to inheritance of deadlines along with all other attributes and methods of a class needs more research work. Because simple automata become very complex when time or deadline is introduce into the state machine model.

- Hardware is tightly coupled to the software of the real time environment application. Some time it is not the software that can guarantee real time task deadlines, but the hardware that has to respond in the specific time limit. If no hardware is available to guarantee the required minimum deadlines then hardware development for real time applications is the another area of research that must be considered in more detail. In this research work hardware aspect has been touched slightly. It is recommended that hardware issues relating to real time environment application be treated as a separate research project.
• Concurrency is the core characteristic of the real time application. When concurrency is considered at the design level then inheritance issues increase the complexity many folds. Because there are lots of concurrent tasks in the real time application, so the complexity introduced due to concurrency and inheritance of concurrent task is also a separate area of research.

• The further research can be carried out for development and implementation of a real time application using the proposed framework model and the metrics. That means the task deadlines have to be incorporated early into the real time application development life cycle. Then after the measurement by our defined real time application design metrics, decide whether to proceed to implementation phase or go back to design phase.
References


[37] R. Kurki-Suonio, “Real time: further misconceptions (or half-truths)”, Computer, vol. 27, issue 6, 1994, page(s) 71-76.


Index

A

AHF, 18
AIF, 18
ARTS, 26, 68, 70, 72, 76
ATM, 23
Attribute Hiding Factor, 18
Attribute Inheritance Factor, 18, 19

C

CBO, 17
CK, 17, 21, 41, 75
COBRA, 26, 67, 68, 70, 72
COF, 18
Coupling Between Objects, 17
Coupling Factor, 18, 20
cyclomatic complexity, 16

D

DARTS, 25
DBPF, 43, 59, 63, 65, 69, 72
deadline, 12, 13, 23, 29, 30, 44, 46, 47, 50, 61, 63, 65, 66, 68, 69, 70, 72, 73
Deadline based Predictability Factor, 52, 59, 69
Depth of Inheritance tree, 17
DIT, 17, 39

E

Early Decomposition Factor, 51, 59, 64, 69

F

Finite State Machines, 25
firm real-time application, 23

FSM, 25

H

hard deadline, 23, 44, 69
Hard Deadline Cohesion Factor, 49, 59, 61, 69
HDCF, 43, 49, 59, 61, 62, 65, 69, 72
HOOD, 26, 27, 30, 67, 68, 70, 72, 76
HRT-HOOD, 27, 72

I

Iterative Developmental Model, 36

J

Jackson System Development, 25, 26
JSD, 25, 26, 67, 68, 70, 72

L

Lack of Cohesion in Methods, 17
LCOM, 17, 21
LCSF, 43, 52, 65, 66, 67, 68, 69, 70, 72
Life Cycle Support Factor, 52, 67, 72

M

McCabe’s complexity, 41
MDMF, 42, 71, 72
measurement, VII, 12, 13, 14, 16, 17, 41, 48, 49, 52, 56, 58, 59, 65, 69, 71, 72, 73, 74
MEF, 51, 59, 62, 65, 69, 72
Message Exchange Factor, 51, 59, 62, 69
Method Hiding Factor, 18
Method Inheritance Factor, 18, 19
MHF, 18
MIF, 18, 19, 22
MOOD, 18, 21, 22, 41, 75

N
NOC, 17, 21
Number of Children, 17

O
OCTOPUS, 29, 67, 68, 70, 72, 76
ODCF, 43, 50, 62, 65, 72
OMT, 28, 29, 68, 70, 72, 76
Overridden Complexity Factor, 51, 59, 62, 69
Overridden Deadline Class Factor, 50, 61

P
POF, 18, 22, 23
Polymorphism Factor, 18, 20
prototyping, 35

R
real time application, VII, 12, 13, 14, 23, 27, 31, 41, 42, 44, 47, 48, 52, 56, 59, 64, 65, 68, 69, 70, 71, 72, 73, 74
Response for a Class, 17, 21
RFC, 17
ROOM, 29, 67, 68, 70, 72, 76
RTO, 29, 67, 68, 70, 72

S
SA/SD, 14, 24, 25
SDCF, 43, 49, 59, 60, 65, 69, 72
soft deadline, 23
Soft Deadline Cohesion Factor, 49, 59, 60, 69
software engineering, 12, 41
S-OODM, 13, 14, 31, 77
statechart, 25, 36, 43, 46, 47
<table>
<thead>
<tr>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transnet, 30, 67, 68, 70, 72, 76</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML, 38, 39, 41, 56, 78</td>
</tr>
<tr>
<td>Unified Modeling Language, 38, 56</td>
</tr>
<tr>
<td>Usability, 12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>waterfall model, 14, 31, 34, 42</td>
</tr>
<tr>
<td>Weighted Method per Class, 17</td>
</tr>
<tr>
<td>WMC, 17, 39, 40</td>
</tr>
</tbody>
</table>