Modeling of Security Measurement (Metrics) in an Information System

A thesis submitted on partial fulfillment of the
requirement for the degree of
Doctor of Philosophy (Ph.D.)

by

Irshad Ahmad Mir

P.G. Department of Computer Sciences
Faculty of Applied Sciences & Technology
University of Kashmir

under the supervision of

Prof. S.M.K. Quadri

in

Computer Science

May, 2013

Declaration

This is to certify that the thesis entitled "**Modeling of Security Measurement (Metrics) in an Information System**", submitted by **Mr. Irshad Ahmad Mir** in the Department of Computer Sciences, *University of Kashmir, Srinagar* for the award of the degree of Doctor of Philosophy in Computer Science, is a record of an original research work carried out by him under my supervision and guidance. The thesis has fulfilled all the requirements as per the regulations of the University and in my opinion has reached the standards required for the submission. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Supervisor and Head

(Prof. S.M.K. Quadri)

Department of Computer Sciences.

University of Kashmir

Srinagar-190 006

Dated: 1st –May –2013

To my Generous Father & Loving Mother
For Their Love & Continuous Support in My Journey
&
To Fouzia for Her Support when I needed that

Acknowledgement

In the name of Allah the most Merciful and Beneficent

First and Foremost praise is to ALLAH, the Almighty, the greatest of all, on whom ultimately we depend for sustenance and guidance. I would like to thank Almighty Allah for giving me opportunity, determination and strength to do my research. His continuous grace and mercy was with me throughout my life and ever more during the tenure of my research.

Now, I would like to thank and express my deep and sincere gratitude to my supervisor **Prof. S. M. K Quadri,** Director, Department of Computer Sciences, University of Kashmir for his continuous support, guidance and encouragement. In addition to being an excellent supervisor, he is a man of principles and has immense knowledge of research. I never forget and thankful for his continues support during the trouble times of my research.

I would also like to express my sincerest gratitude to my ex-supervisor Late **Dr. Mehraj-u-Din Dar**, who showed me the path in the early stages of my research. He was a person with dynamic leadership capabilities and served the community with great dedication. He was a person having a brave heart who fought with Cancer over the period of one year. May Almighty bestow him with Jannah.

I would also like to express my gratitude to the pioneers of my research field. Their work helped me a lot in my research. They include researchers like Rejio Savola, Pratyusa K. Manadhata, W. Jansen , Jaquith Andrew and more.

A Ph.D. candidate also has a life besides studies. Besides my research activities, I also enjoyed my stay in UoK. I thank my fellow colleagues and friends for spending time with me in canteen and for playing counterstrike in lab during leisure hours.

I owe everything to my family who encouraged and helped me at every stage of my personal and academic life and longed to see this achievement come true. I would be nothing without the love, sacrifices and support of my parents. I dedicate this work to my sincere and generous father and my loving mother. Every breath of my life and drop of blood in my body is dedicated for them. I am thankful to a person close to my heart for the tremendous support despite the sufferings of my hectic working hours, who tolerated the tiring times of my research and believed in me when I doubted myself.

I love you all.

Irshad Ahmad Mir

Abstract

Security metrics and measurement is a sub-field of broader information security field. This field is not new but it got very least and sporadic attention as a result of which it is still in its early stages. The measurement and evaluation of security now became a long standing challenge to the research community. Much of the focus remained towards devising and the application of new and updated protection mechanisms. Measurements in general act as a driving force in decision making. As stated by Lord Kelvin “if you cannot measure it then you cannot improve it”. This principle is also applicable to security measurement of information systems. Even if the necessary and required protection mechanisms are in place still the level of security remains unknown, which limits the decision making capabilities to improve the security of a system. With the increasing reliance on these information systems in general and software systems in particular security measurement has become the most pressing requirement in order to promote and develop the security critical systems in the current networked environment. The resultant indicators of security measurement preferably the quantative indicators act as a basis for the decision making to enhance the security of overall system.

The information systems are comprised of various components such as people, hardware, data, network and software. With the fast growing reliance on the software systems, the research reported in this thesis aims to provide a framework using mathematical modeling techniques for evaluation of security of the software systems at the architectural and design phase of the system lifecycle and the derived security metrics on a controlled scale from the proposed framework. The proposed security evaluation framework is independent of the programing language and the platform used in developing the system and also is applicable from small desktop application to large complex distributed software. The validation process of security metrics is the most challenging part of the security metrics field. In this thesis we have conducted the exploratory empirical evaluation on a running system to validate the derived security metrics and the measurement results. To make the task easy we have transformed the proposed security

evaluation into algorithmic form which increased the applicability of the proposed framework without requiring any expert security knowledge.

The motivation of the research is to provide the software development team with a tool to evaluate the level of security of each of the element of the system and the overall system at the early development stages of the system life cycle. In this regard three question "*What is to be measured?*", "*where (in the system life cycle) to measure?*" and "*how to measure?*" have been answered in the thesis.

Since the field of security metrics and measurements is still in the its early stages, the first part of the thesis investigates and analyzes the basic terminologies , taxonomies and major efforts made towards security metrics based on the literature survey.

Answering the second question "*Where (in the system life cycle) to measure security*", the second part of the thesis analyzes the secure software development processes (SSDPs) followed and identifies the key stages of the system's life cycle where the evaluation of security is necessary.

Answering the question 1 and 2, "*What is to be measured* "and "*How to measure*", third part of the thesis presents a security evaluation framework aimed at the software architecture and design phase using mathematical modeling techniques. In the proposed framework, the component based architecture and design (CBAD) using UML 2.0 component modeling techniques has been adopted. Further in part 3 of the thesis present the empirical evaluation of the proposed framework to validate and analyze the applicability and feasibility of the proposed security metrics. Our effort is to get the focus of the software development community to focus on the security evaluation in the software development process in order to take the early decisions regarding the security of the overall system.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

1.1 Introduction

The Journey of man started from the Stone Age to agricultural age and now we are in the today's age of information technology, where from an individual to an enterprise or organizations are heavily dependent upon information and the information processing systems. Information ranging from personnel to commercial have been processed and exchanged by these information systems. With the advent of Internet, the convergence of information & communication technologies and todays very complex nature of business environment resulted in myriad trust and information security concerns. The secure functioning of these information systems is the utmost important and foremost concern. Information security is a field of security which ensures the confidentiality, integrity and availability of information and information processing resources. Many security professionals think that developing a completely secure system is almost an impossible task. According to **[Connolly, 2001]** the completely secure system is one that is disconnected from a network, encased in concrete, and lying at the bottom of the ocean. In this networked environment where there are potential number of hackers and adversaries present, security enforcing mechanisms needs to be incorporated in the information systems to with stand with the both deliberate and accidental malicious intents . Verities of security enforcing mechanisms have been developed and utilized with varying degree of success but the level of achieved security remained almost unclear. Enforcement of security mechanisms alone does not help unless we don't know about how secure a system is? What level of security is desired? Measurement has been a cornerstone of good science for centuries. More than 100 years ago Lord Kelvin observed the importance of measurement in the physical science. He stated that if you can't measure it you can't improve it and if you can express in numbers what you measuring, you know something about it. This fact is also applicable to the security of information system, because without knowing the level of security achieved, it is almost impossible to protect it. We measure to reveal the conditions possibly alert the user, importantly we measure to control processes **[Fowler et al., 2004]**. There are two facets in the field of information security, one is to device new mechanisms to enforce the security protection mechanisms and second is a reliable and systematic approach for measuring and assessing the security level of the system.

Security has been an important quality factor in many types of interactive systems such the complex banking software. The questions, how secure a software product or a network is? How

secure does it need to be? The answers to these questions are possible if we have a reliable system of measurement that can provide us both the quantative and objective indicator of security possessed by a system. Such measurement system certainly help in sound decision making regarding the secure system development and will ultimately provide a baseline to enhance the security of the system in an efficient manner

Measurement is the way by which humans understand with more precision the rational world. We measure to reveal a condition and possibly alert the user. We also measure to quantify the magnitude of phenomena. Probably most importantly, we measure to control processes **[Fowler et al., 2004]**. Paraphrasing Lord Kelvin, when you can measure what you are speaking about and express it in numbers, you know something about it **[William, 1891].** The measurement and metric adoption is almost always a tool to improve and manage developing process [**Knowledge, 2011].**

The rest of the chapter is organized as: Section 1.2 presents a small overview of the security metrics, Section 1.3 presents the terminologies used in the thesis, Section 1.4 explains the motivation behind this work, Section 1.5 presents the goals and objectives of the study, Section 1.6 prsents the contribution made by this study followed by the outline of the thesis in section 1.7.

1.2 Overview of Security Metrics

Security metrics is an area of computer security relatively young and received focus very lately and sporadically. Much of what has been written about security metrics is definitional, aimed at providing guidelines for defining a security metric and specifying criteria to strive for **[Jansen, 2010].** The complication behind the immaturity of security metrics is that the current practice of security is still a highly diverse field, and holistic and widely acceptable approaches are still missing **[Savola. 2007]**. It is now an emerging research area rapidly gaining momentum. It is reasonable to infer from the experience to date that security measurement is a tough problem, not to be underestimated **[Bellovin et al., 2006].** In their study **[PfLeeger et al., 2010]** pointed out the nine reasons why security measurement is hard.

Measurement is a way by which we understand the rationale world precision. A metric implies a system of measurement based on quantifiable measures. For information system security, the measures are concerned with the identification of the inherent attributes of the system that are

responsible for the security of the overall system. To generate the security metrics that portray the security state of a system needs a method of measurement by taking into account the internal assessable attributes of the system to obtain the measured values.

Many major efforts to measure or assess security have been attempted. They include the Trusted Computer System Evaluation Criteria (TCSEC) (Department of Defense,1985), Information Technology Security Evaluation Criteria (ITSEC) (Commission of the European Communities, 1991), Systems Security Engineering Capability Maturity Model(SSE-CMM) (International Systems Security Engineering Association, 2008), and Common Criteria (Common Criteria Portal, 2006). Each attempt has obtained only limited success. It is reasonable to infer from the experience to date that security measurement is a tough problem, not to be underestimated **[Bellovin et al., 2006].** Further evidence is that the topic, Enterprise-Level Security Metrics, was included in the most recent Hard Problem Lists prepared by the INFOSEC Research Council (2005), which identifies key research problems from the perspective of its members, the major sponsors of information security research within the U.S. Government.

1.3 Terminology

In this section we put forward the various important definitions and terms used though out the thesis.

1.3.1 General Security terminology

At its simplest, security is the process of protecting against injury or harm. The subject of this thesis is related to the security evaluation of information systems particularly the software systems. Generally the word security refers to the protection of an asset, such as software security, network security. The definition of security varies according to the context of the study. There is not a single definition of the term security even it is defined very roughly also **[Bishop, 2003].** The main reasons behind it are the diversity of the field of study and also computer security is still in the infancy stage of the discipline **[Andrews et al., 2004].** Security in general means protection of assets **[Gollmann, 1999]** such as data, information, hardware, software, networks, and people. Information security refers to the process of employing technical measures *"To protect information and information system from unauthorized access, use, disclosure, disruption,*

modification, or destruction in order to provide confidentiality, integrity, and availability." **[Bishop, 2003].**

1.3.2 Security (definition used in this thesis)

There are different opinions regarding the definition of security. Security is to be related only the protection against the intentional attacks **[Bishop A, 2003],** but according to **[American N. S., 2001]** security includes both the protection against the intentional and accidental malicious intents. In our study we take into account the second case in which it includes both the intentional and accidental risks.

As with the growing importance of information for the organization and individuals, the term information security is commonly used today to represent the security of the information systems which means protecting the information and information systems from unauthorized access, use, disclosure, disruption, modification or destruction. The term information security in real sense is the protection of information regardless of the use of electronic media used, but since currently these electronics means are heavily used by the organizations and the government the terms computer security, information security and information assurance are frequently used interchangeably. In this thesis the following definition of the security:

Security of a system is defined as the protection of the system and its resources against the confidentiality, integrity and availability breach.

1.3.2 Information system

An Information system is a collection of components which stores, process and transmit the information. The various components of an Information system are:

People: (end users and IS specialists)

Hardware: (physical computer equipment and associated devices machines and media)

Software: Programs and procedure (machine readable instructions) that direct the all other components of the system.

Data: Data and knowledgebase involved.

Networks: communication media and network support to fuse various components and to transmit and receive the data and information.

Since our security evaluation aim at the software systems, in this thesis we use the term information system or simply a system to refer to software systems.

1.4 Motivation

The reliance on the information system especially on the software system is increasing day by day. Security of these systems becomes the utmost necessity of the time. Even if the security enforcing mechanisms are to be adopted, the one main question, "How much secure we are" still remains unanswered. It is so rightly stated by Lord Kelvin that "if you can't measure it then you can't improve it. Measurements play a vital role for the application of security mechanisms to these systems. In practice the security evaluation is carried out through reasoning and guess work rather than the direct measurement of the of actual hardware and software components **[Jansen, 2010]**. In particular to the software engineering process various quality attributes have been investigated and studied by the researchers across the globe. The security got very least attention and always treated as an add-on property **[Savola et al., 2009].** If the secure software engineering in practice is carried out that also has been carried out in isolation from the software engineering process **[Meadows, 1994].** Like other quality attributes, security needs to be considered throughout the development phases of the software development. The advantage of security considerations in the early stages of the software life cycle is twofold. On one hand it promotes the more secure systems and on the other hand detection and correction of the security flaws in the early stages of the system life cycle can considerably reduce the cost and efforts required in the further stages of software life cycle. Application of security mechanisms in the software development requires the identification of the most critical components of the system involving the higher risk. Such identification should not be based on the guess work instead; it requires a well-defined well-established measurement process and the metrics that provides the software engineers preferably the quantative indicators of the security posture of the system. The challenge posed by the today's vulnerable networked environment and the potential number of threats posed by both intentional and unintentional malicious intents demands that the system should be evaluated for the security . In his book **[Jansen, 2010]** realized the importance of security evaluation and metrics, the current progress in the field and pointed out the main areas that needs to be taken care by the research community.

Some of the major limitations and motivational aspect of the field of measuring security are:

- Increase in the complexity of software system.

- There is no effective security evaluation framework which developers can use to evaluate the system at the early development stages.
- The current practice of security is still a highly diverse field, and holistic and widely acceptable approaches are still missing **[Savola, 2007].**
- Much of what has been written about security metrics is definitional, aimed at providing guidelines for defining a security metric and specifying criteria to strive for. However, relatively little has been reported on actual metrics that have been proven useful in practice **[Center for Internet Security, 2008], [Berinato, 2005].**

1.5 Research Goals and Objectives

The ultimate goal of this research is to:

Propose a security metric framework and modeling the security metrics for the software systems at the design and architectural level of the system life cycle independent of the size and the environmental factors involved.

Since the field of security metrics is young, in this research various security metrics taxonomies have been investigated. The research aims to evaluate the software system for the security in order to provide the system developers the indicators about the security posture of the system and its components. In answering three question *"Where in software life cycle the security is to be measured"*, *"what is to be measured "* and *" how to measure "* the research objectives are:

1. To analyze the role of security metrics and identify the current practices and processes involved in the secure system development.
2. Identification of the key stages of the software systems lifecycle where the security metrics must be applied to improve the security of the system.
3. Propose a security evaluation framework and derive the metrics using mathematical modeling techniques for the architecture and design stage of the software development. The derived security metrics should act as a tool for the developers.
4. Transformation of the security metric framework into algorithmic form, in order to enhance the applicability of the proposed evaluation framework.

5. Empirical evaluation of the proposed framework on a running system, to validate the feasibility and applicability and relevance of the proposed framework and derived metrics.

1.6 Contributions

With the fast growing reliance on the information systems especially the software systems, the challenge for the security professionals and the research community is to deliver the secure system operations. The advance in the security of any system is only possible if we know how much secure it is and how much secure it needed to be. The answer to these questions can only be possible if we have such mechanisms and scales to measure the security level of a system. The measure contributions of this thesis are:

1. Since the field of security metrics is still young , based upon the survey of existing studies analyzed the basic of security metrics and the taxonomies of the field
2. We surveyed the existing studies on the secure software development process and the tools used in capturing and analyzing the security related issues throughout the software development. Based on the survey we have identified the measure stages of a system lifecycle where security evaluation is to be carried out.
3. Proposed a security metric framework using mathematical modeling techniques to derive the metrics on a controlled scale, for the architecture and design stage of the system lifecycle.
4. We have empirically evaluated the proposed security evaluation framework and the proposed metric on a running system, in order to check the feasibility and the applicability of the proposed framework and derived security metrics.

1.7 Outline

Chapter 2 discusses the preliminaries of the security metrics. Since the field of security metrics is still in its initial stage various terms and taxonomies regarding security metrics have been discussed. The chapter also point out some of the major efforts made towards the security metrics.

Chapter 3 discusses the security in software development in general and security metrics in particular. Further chapter discusses the stages of the software lifecycle where the security

evaluation is necessary and must be carried out. Various tools and techniques in secure software development process have been discussed.

Chapter 4 discusses the factors and attributes of that any security evaluation process should strive for. It also discusses the various software and architectural process that have been adopted in practice along with the relative merits and demerits. Further an extended novel security evaluation framework for the software architecture and design has been proposed and derived the metrics using mathematical modeling techniques in the chapter.

Chapter 5 presents the empirical evaluation of the proposed framework on a running system, in order to validate the results and to analyze the feasibility, efficiency, and applicability of the proposed framework. It begins with the data collection followed by the application of the derived security metrics of the framework and result analysis.

Chapter 6 summarizes the major findings of this work and the contribution to knowledge made in in this thesis. Further it, presents the future scope of the work for further research.

Some of the material presented in the thesis has been published previously. The complete list of the published articles follows the chapter 6.

CHAPTER 2

Security Metrics Background & Preliminaries

2.1 Introduction

With a shift from standalone application to the complex interconnection of insecure components and networks, the information systems especially the software systems are becoming more and more vulnerable. Verities of protection mechanisms and security approaches are applied but the resulting security level remains unknown. Like other software quality attributes, the level of security a system possess need to be outlined and measured. Such security metrics can be utilized in many ways to benefit an organization, including increasing accountability, improving security effectiveness, and demonstrating compliance **[Alger et al., 2001]** "How much secure a software system is? " "How secure does a system need to be?" "What are the factors responsible for the security of the system?" and "The stages in the software development where the security need to be evaluated", these are questions that are asked to those who work to evaluate the efficiency of security efforts. The answer to all these questions can be only possible if we have such security metrics that evaluate the system for security and provide the evidence of the security level and performance of the System under investigation.

In particular to the software systems several quality attributes, such as reliability, size, complexity etc. have been investigated and evaluated. Very least attention has been remained towards the evaluation of security. Literature surveys showed that the security is an emerging concern in the current times ranging, from an organization to an individual. The area of security metrics is very hot and demanding but at the same time the field is very young **[Jansen, 2010].** The problem behind the immaturity of security metrics is that the current practice of information security is still a highly diverse field and holistic and widely accepted approaches are still missing **[Savola, A 2007]**. The field still aims mainly at the basic definitional aspect and lacks in well-structured literature at hand. According to **[Savola A, 2007]** in order to make an advance in the field of measuring, assessing or assuring security, the current state of the art should be investigated thoroughly. In this chapter we present the preliminaries of the field of security metrics and based on the literature survey analyze the relevant major effort made for measuring the security of information system in general and for the software system in particular.

The rest of the chapter is organized as: Section 2.1 prsents some preliminary concepts of security metrics, theirs properties and objectives. In section 2.3 investigates and presents some major taxonomies in the field of security metrics, Section 2. 4 presents some of the major efforts made

towards the actual measurement in a classified manner, followed by the section 2.5, which presents the conclusion

2.2 Security Metrics Concepts

To understand security metrics, we must first differentiate between the metrics and measurement. Measurements provide single-point-in-time views of specific, discrete factors, while metrics are derived by comparing to a predetermined baseline two or more measurements taken over time **[Jelen, 2000].** Measurements are generated by counting; metrics are generated from analysis **[Alger, et al., 2001]**.In other words, measurements are objective raw data and metrics are either objective or subjective human interpretations of those data. The well-developed security evaluation framework and derived metrics can act as an effective tool for security manager to discern the effectiveness of various components of their security programs, a system, a product or process **[Payne, 2006] .** Such security metrics certainly enables the development team to provide the necessary protection mechanism to ensure the secure system development. As mentioned earlier security metrics have many interpretations. Below are some of short elaborations of term security metrics.

- According to **[SSE-CMM, 2011],** metrics are quantifiable measurements of certain aspects of the system or enterprise. Such measurement is based on some attributes of the system that are responsible for the security of the system. Further, a security metric is a quantative measure of how much these attributes the system possess.
- According to **[Swanson. M et al., 2003],** metrics are tools designed to facilitate decision making and improve performance and accountability through collection, analysis, and reporting of relevant performance-related data such that the end results aim to facilitate in taking the required corrective measures.
- According to **[Payne, 2006]** measurements provide single-point-in-time views of specific, discrete factors, while metrics are derived by comparing to a predetermined baseline two or more measurements taken over time. Measurements are generated by counting; metrics are generated from analysis. In other words, measurements are objective raw data and metrics are either objective or subjective human interpretations of those data.

In their work **[Farooq et al., 2011]**, outlined the importance of measurements and metrics and their relation to the software testing. In the same work **[Farooq et al., 2011]** analyzed the process of software measurement process in general, which is comprised of following key stages.

- *Planning:* Defining the procedure and scope of the measurement process.
- *Implementation:* The actual application of measurement process and procedure defined at the planning stage. The output of this stage should be in the form reports of performance related data.
- *Improving:* Based on the process and progress evaluation (through the reports generated in the implementation phase) the necessary decision is to be taken in order to make an improvement to the system.

The output scale of the software measurement and metrics is possibly hierarchal in nature, which is comprised of various levels. Each level scale in the hierarchy possesses all the properties of lower level scale in the hierarchy. In the same work **[Farooq et al., 2011]** identified the five measurement scale based on the literature survey.

2.2.1 Characteristics of Security Metrics

According to **[Jelen. G, 2000]**, security metrics should be SMART, i.e. Specific, Measureable, Attainable, Repeatable, and Time dependent. Security metrics should be able to identify and measure the degree of security attributes like confidentiality, integrity and availability of a system. The method of measurement employed should be reproducible, that is, capable of attaining the same results when performed independently by different evaluators **[Jansen, 2010].** The ultimate goal of security metrics should be to mitigate the security risk and act as a tool for decision making especially in *assessment* or *prediction*, for the development team and other stockholders. When the target is to predict the security level of a system then mathematical model and algorithms are applied to the collection of measured data (e.g. regression analysis) to predict the security of the system, process, or product. Our security metric framework is based on the *prediction* method where mathematical modeling techniques has been adopted and finally the security evaluation process is transformed into an algorithmic form. It is important to clearly know the entity that is the target of measurement because otherwise the actual metrics might not be meaningful **[Savola, 2008].** Federal Information Processing Standards **[FIPS, 2004]** provides a mechanism for the investigation of *confidentiality*, *integrity* and *availability* separately. As the

security requirements for each of the system or organization vary according to the needs, so the security evaluation should be based on the well-defined security attributes such as *confidentiality, integrity, availability, privacy, nonrepudiation* etc.

2.2.2 Security Metrics: Properties

Security metrics properties can be investigated based on the following classification **[Savola, A 2007]**

- **Quantitative vs. Qualitative metrics**: The end result of the security metrics may be either quantitative or qualitative in nature. The quantitative results are preferred over the qualitative one because of the discrete measurable nature of the results. At the same time the generating the quantitative metrics are more challenging than the quantitative one.
- **Objectivity vs. Subjectivity of Security Metrics:** As with the case of quantative vs. quantitative the resultant security metrics should be either objective or subjective in nature. Objective security metrics is the preferred one and portrays the security posture of a system or process in certain discrete levels such as *low*, *medium,* and *high* on a scale. The subjective metrics normally takes into consideration the human behavioral aspects in the security.
- **Direct vs. Indirect metrics:** Direct metrics are those that measure an atomic attribute of the system in a sense that the measured attribute responsible for the security does not depend on the other attributes , whereas indirect metrics involve multiple attributes that are interdependent.
- **Static vs. Dynamic metrics:** Result of the dynamic metrics will be effected by the time elapsed whereas static metrics do not take the time into the account.
- **Absolute vs. Relative metrics:** An absolute metrics is atomic in nature in a sense that it does not depend on the output of any other metric, whereas relative one does.

2.2.3 Security Metrics Objectives:

The main objective of the security metrics is to gauge into the system for the level of security it possess, such that the most critical elements of the system with respect to the security can be identified. According to **[Savola, 2009]** security correctness, security effectiveness and security efficiency are main three objectives of the security measurement, defined as:

- ***Security Correctness:*** ensures that security enforcing mechanisms have been implemented correctly in the system under investigation and system meets its security requirements.
- ***Security effectiveness:*** ensures that stated security requirements are met in the system under investigation and the system does not behave in any way other than what it is intended.
- ***Security efficiency:*** ensures that the adequate security quality has been achieved in the system under investigation.

2.3 Security Metrics taxonomies

Since the area of security metrics is still in its early stages with varying definitions and terminologies. Various security metrics taxonomies exists in the literature that aim at the categorization of security metrics at higher level of abstraction. Based on the literature survey in this section we look at some of the most common among them.

In **[Swanson, 2001], [Swanson et al., 2003]**, NIST presented a security metrics taxonomy, which categorized the security metrics into three modes i.e. *Management, Technical* and O*perational.* It further presents 17 sub categories of metrics each with examples. The focus of this taxonomy is from the organizational and stockholders perspective, rather than the technical perspective of a particular system. Below diagram 2.1 depicts the classification of security metrics proposed by NIST.

In their study **[Henning et al, 2002]**, workshop on Information Security System, Scoring and Ranking (**WISSR)** provides a detailed discussion on issues related to Information Assurance (IA). They used the term IS* for Information Security. The (*) with IS is related to security measurement and can be used to denote the terms like metric, scores, ratings, rank, assessment results etc. The work shop did not propose any new specific security metric taxonomy, instead it was organized into three tracks; *technical, operational* and *organizational* based on the interest of the participants. Technical metrics are used to describe and compare the technical objects such as an algorithm, specification, design etc. Operational metrics are used to manage the risk to the operational environment and the organizational metrics are used to describe and track the effectiveness of organizational program and process.

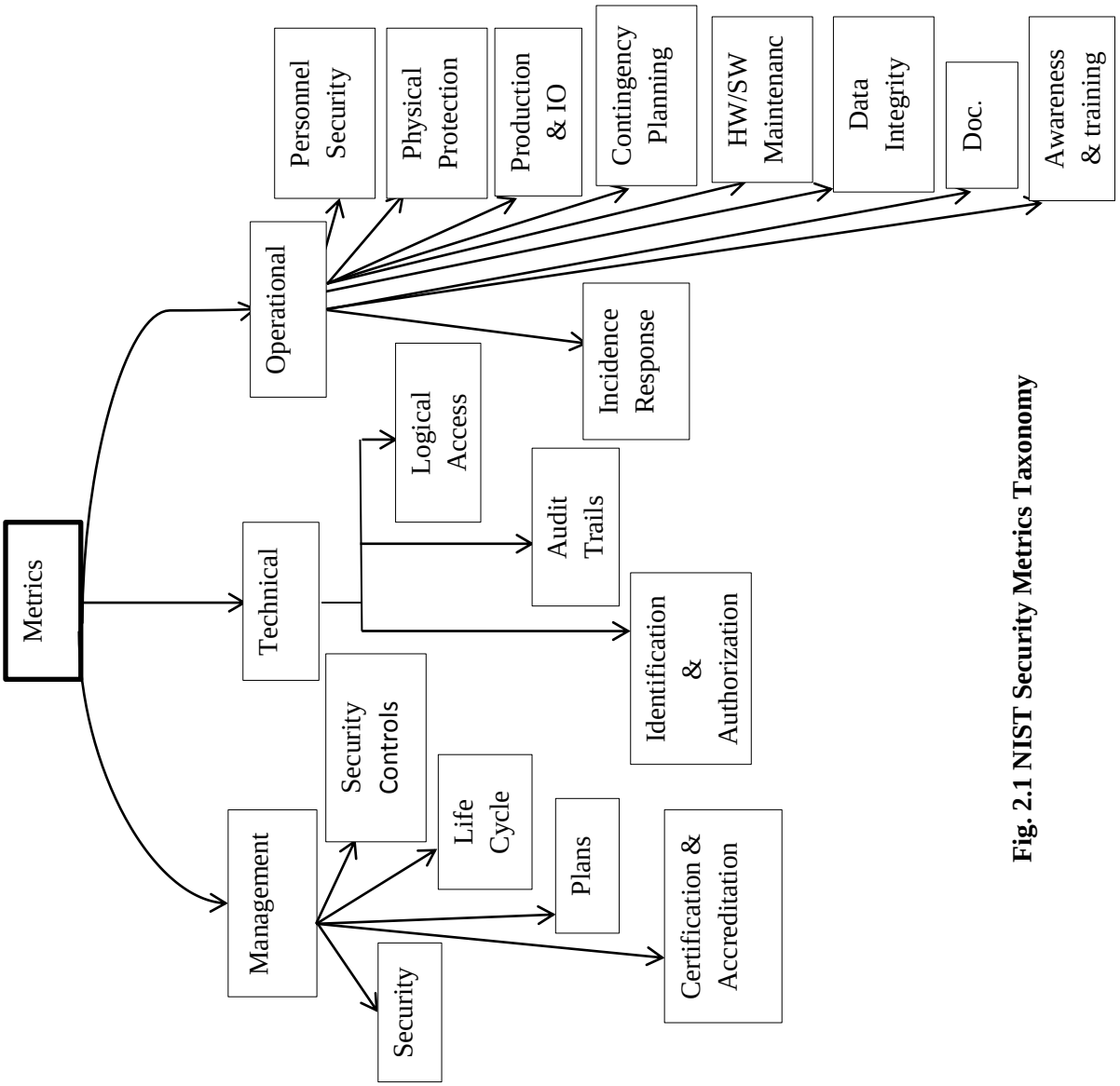

Fig. 2.1 NIST Security Metrics Taxonomy

In their study **[Vaughn et al., 2003]** proposed a taxonomy for Information Assurance (IA) metrics. They divided the taxonomy into two distinct categories of security metrics (a) *Organizational security metric* (b) *Technical Target of Assessment* (TTOA). The former aims at providing the feedback to improve the security assurance status of the organization. The second category metrics (TTOA) is intended to measure the security capability of a particular system or product. Both categories are further categorized in order to put the specificity in terms of measuring security. Above figure (2.2) shows the higher level classification of the taxonomy.

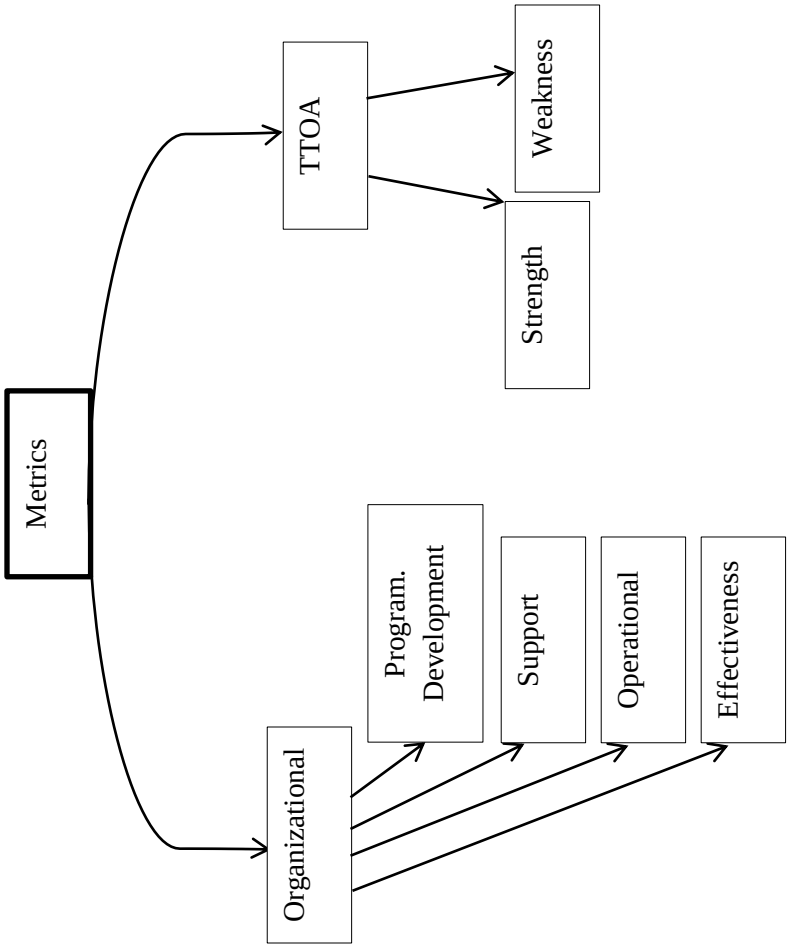

Fig 2.2 Taxonomy of Vaughn et al.

In their **[Seddigh et al., 2004]** proposed a new taxonomy for IA (information assurance) and IT networks that aim to provide the basis and motivation for the research in overcoming the challenges in the area. In Their taxonomy the metric space is divided into three categories: Security, QoS, and Availability. Each of these categories is further categorized into three subcategories as technical, organizational and operational metrics, which are further categorized into 27 classes. According to them, organizational metrics evaluate an IT organization's emphasis on IA (Information Assurance) in terms of goals and organizational policies. Technical metrics evaluate the technical components of an IA network and also the subset metrics under this category provides the rating, incident statistics and security testing. Operational metrics evaluate the operations of an IT organization in terms of complying with the goals and policies

set by the organization. Figure (2.3) shows the proposed taxonomy by the authors at the higher abstract level of classification.

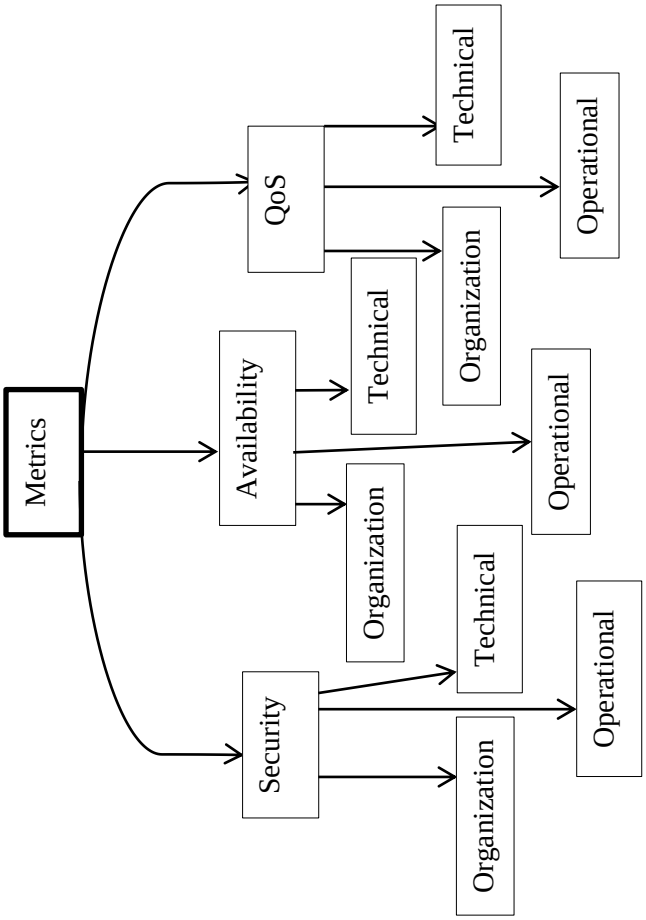

Fig. 2.3 Taxonomy of Seddigh et al.

In his study **[Savola. A, 2007]** proposed a security metrics taxonomy based on the literature survey. The main aim of the author is to bridge the gap between Information Security Management and Information and Communication Technology Industry (ICT). In this taxonomy the author's intent was to enhance the composition of feasible security metrics all the way from the business management to the lowest level of technical details. This taxonomy categorized the security metrics in a tree like structure into six levels from L0 to L5 with business level security metrics at the root (L0) and the implementation level technical metrics at the leaf nodes (L5). At the higher level, business level security metrics are divided into five sub categories, (i) Security metrics for cost-benefit analysis, containing economic measures such as ROI (return on investment) (ii) Trust metrics for business collaboration (iii) Security metrics for information security management (ISM) (iv) Security metrics for business level risk analysis (v) Security

dependability and trust (SDT) metrics for ICT product system and service. The author further provided the sub categories of the above category (iii) and (v). Below diagram shows the

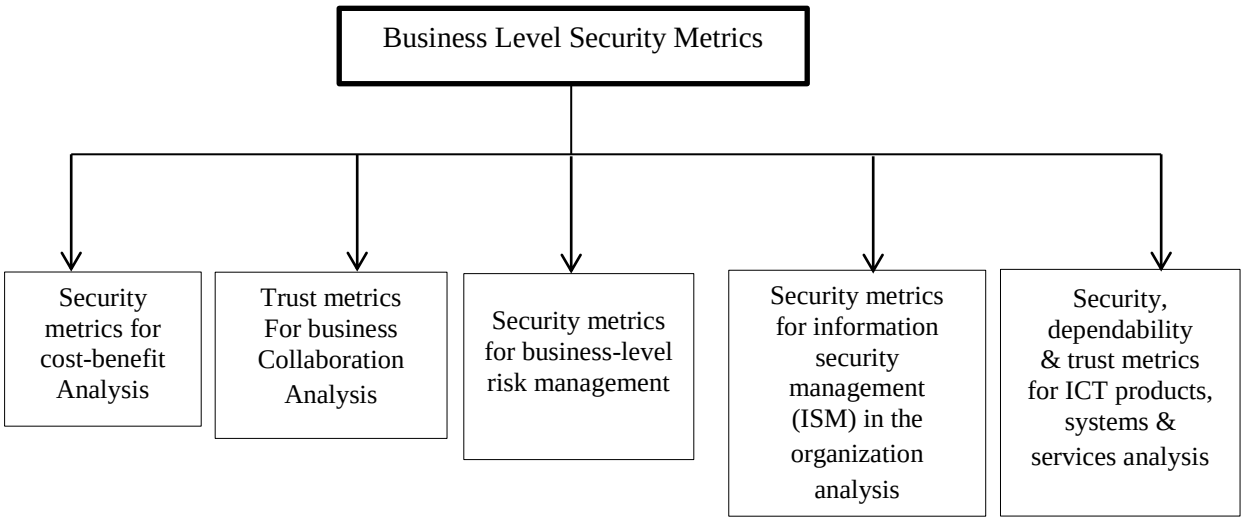

Fig. 2.4 Security Metrics Taxonomy of Rejio.S

classification of security metrics taxonomy.

All the proposed security metrics taxonomies are conceptual and abstract in nature. Very little has been reported on the actual scale of measurement. From these taxonomies it is evident that a great deal of efforts is needed to devise the metrics that can be applicable in real practices.

2.4 Software Security vs. Software Reliability Measurement: An Overview

Software reliability always remained an important quality attribute of the software system and various efforts to evaluate and measure the reliability has been made. The idea of security and reliability are technically derived from the requirement to describe correctness. Both the terms have grown up in different domains of thinking. Security can be defined as a functional statistical predictability statement where the answer to the question being secure or not is whether a given system specified can be expected to continue to function for some period in some specified manner. Reliability can be defined as a functional statistical predictive statement of predictability where a system in reliable state or not is whether a given system can be expected to continue function for some specified period in some specified manner **[Roy D. Follendore, 2002].** Reliability and security are not the isolated from one another; instead reliability has a great impact on the security of a system. The "reliability of security" is often considered but the "security of reliability" is not often considered **[Roy D. Follendore, 2002].** As far as software measurement is concerned the various efforts have been made to device the new and updated

metrics, models and measurements techniques to evaluate the reliability of the software systems and very least efforts to evaluate the security of software system has been made . The main problem behind it may be the multifaceted nature of security and the dependency of security on the various other qualities attributes like testing and reliability of the software systems. In their work **[Farooq et al., 2012]**, presented an in depth analysis of the key concept, metrics, models and measurement used in software reliability. Since the reliability is the probability of a system or components to perform its required service without failure under stated conditions for a specified period of time **[Farooq et al., 2012]**. Various probabilistic models and methods have been proposed to predict the reliability of a system. Among the proposed model and methods the software reliability growth models (SRGM) has been used to predict the reliability of the systems. SRGM shows how reliability of a system improves in a period of time when the faults are detected and repaired. SRGM is actually used to determine when to stop testing to attain a given reliability level **[Quadri S. M. K et al, 2011].** Over the last three decades many efforts have been made to develop the SRGMs. Among them **[Musa et al., 1887], [Xie, 1991], [Lyu, 1996],** are the most common SRGMs and verities of metrics like number of remaining faults, mean time between failure (MTBF), and mean time to failure (MTTR) have been derived. In the later times **[Bokhari and Ahmad, 2006], [Quadri S.M. K et al, 2006]** proposed the probabilistic software reliability growth model based on Non-homogeneous Poisson process (NHPP) which incorporates the testing efforts. Further in their work **[Quadri S.M.K et al, 2011]** a scheme for constructing software reliability growth model (SRGM based on (NHPP) have been proposed.

Table 2.1 Software Reliability Prediction Models

Model	Year	Author(s)
Ohba exponential model	1984	**[Ohba, M. 1984]**
Yamada Rayleigh model with Rayleigh curve	1986	**[Yamada, S et al , 1986]**
Yamada Rayleigh model with Weibull curve	1993	**[Yamada , S et al , 1993]**
Huang Logistic model	1997	**[Huang, C.Y. et al, 1997]**
Quadri's NHPP SRGM with generalized exponential testing efforts	2006	**[Quadri S.M.K et al, 2006]**
Quadri's SRGM based on (NHPP)	2011	**[Quadri S.M.K et al , 2011]**

Above table (2.1) summarizes the major efforts made towards modeling the reliability prediction of software system based on the software reliability growth models (SRGMs). On the other hand there exists no such probabilistic model to predict the security level of the system over a specified time period. Systematic efforts are needed develop the models and methods to predict the security of the software systems. The methods and models of reliability prediction can used to understand the state of art of prediction and measurement.

2.5 Related Work

Security metrics can be obtained at different levels within an organization or a technical system. Detailed metrics can be aggregated and rolled up to progressively higher levels. The various efforts made on actual metrics development are sporadic in nature, some taking into the consideration the knowledge of previous and current vulnerabilities; some measure the code quality some strike at the design of a system. From the literature survey, at the higher level these efforts can be categorized as following based on the major factors taken into the consideration in measuring the security of a system as:

- *Analyzing the capabilities of attacker:* Under this the security of a system is measured by taking into account the required efforts, capabilities and resources of an attacker.
- *Knowledge of Vulnerabilities:* Under this category the security evaluation is carried out by taking into account the knowledge of both the vulnerabilities reported in past and present vulnerabilities.
- *Conceptual:* Under this category the security metrics are based on the knowledge of the personnel regarding security and are conceptual in nature, having very limited use in real practices.
- *Independent:* Such security metrics are based on the analysis of the attributes of the system itself (inner attributes) and are predictive in nature. Security metrics under this category are highly desirable to ship the more secure and less vulnerable systems.

Our proposed security metrics framework in chapter (4) comes under forth category i.e. it is independent of the external security factors and is based on the internal attributes of the system.

In their study **[Manadhata et.al, 2007]** proposed the system attack surface as an indicator to the security of a system and formalized the system attack surface using I/O automata model. The attack surface of a system is comprised of three kinds of resources that an attacker can use to

carry out an attack on the system. These resources are the methods, channels and data of the system. Based on the direction of flow of data they further technically defined the entry points and exit points of the system which are actually the methods of system through which the flow of data takes places. Authors defined the attack surface as subset of system resources that an attacker can utilize carrying out attack on the system and accordingly quantifying the resultant security indicators. Larger the attack surface of a system more the system vulnerable to the attacks. Further the authors analyzed the feasibility of the proposed approach by measuring the attack surface of open source FTP daemons and two IMAP servers.

Major efforts made towards the security measurement by taking the capabilities and resources of an attacker into consideration also known as attacker-centric security metrics are **[Leversage et al., 2008], [Miles et al., 2005]** and **[Ortalo et al., 1999].** In these studies the security measurement is carried out by analyzing the ways an attacker can carry out the successful attack on the system with respect to the knowledge and resources at hand. In contrast the proposed security metric framework in this thesis chapter (4) is based on the system architecture and design. Being an independent of attacker's capabilities our proposed framework act as a tool for the software development team to measure the inherent security of the system and enables them to take the necessary decisions regarding security.

[Littlewood et al., 1993] proposed a conceptual model based on probabilistic methods which initially used for reliability analysis, to measure security of a system. In their conceptual model they proposed the usage of the efforts made by an attacker to carry out a successful attack on the system as a measure of system's security.

In the second category where the knowledge of vulnerabilities reported in past and the present vulnerabilities **[Alhazmi et al., 2006]** proposed a vulnerability density (VD) metric which is defined as the number of vulnerabilities in a unit size of code. From the VD the authors further proposed a set of metrics such as vulnerability discovery rate (VRD) which is the number of vulnerabilities identified per unit time and known vulnerabilities density (VKD). In contrast the proposed security metric framework in this thesis chapter (4) is independent of the knowledge of the past or present vulnerabilities.

[Alves-Foss et al., 1995] proposed the measurement of a system using System Vulnerability Index (SVI) as a measure of system vulnerability to common intrusion methods**.** SVI is

calculated by evaluation the various factors like "system characteristics", "potentially neglectful acts" and "potentially malevolent acts".

[Voas et al., 1996] proposed the security metric based on the technique of deliberate fault injection. The fault injection is carried out by simulating the threat classes of the system by mutating the variables during the execution and then observing the impact of threat class on the behavior of the system. Finally they have proposed a minimum-time-to-intrusion (MTTI) metric based on the time before any simulated intrusion can take place.

[Schneier, 1999] proposed the attack tree to analyze the security of a system. The attack tree is constructed by setting the attackers goal as the root and branches of the tree as the different ways that an attacker can adopt to carry out the attack on the system along with the cost involved along with each of the possible path to carrying out the attack. The cost estimation becomes the measure of the system security. The prerequisite to generate an attack tree is the knowledge about attacker's goals, system vulnerabilities, and attacker's behavior.

Many other conceptual security measurement efforts are made by **[Littlewood et al, 1993], [Madan et al., 2002],[Stuart, 2004]**. These metrics are conceptual in nature and haven't been applied in real security evaluation of the systems.

2.6 Conclusion and Future Scope

In this chapter we have analyzed the preliminaries and various concepts of the security metric field. From the presented taxonomies of the security metrics, it is evident that the field of security evaluation is multifaceted and wide open challenge for the organizations and research community. In practice very little has been delivered on the actual scale and it needs a systematic approach to make the progress in this area. Based on the literature survey we have classified the efforts made in the field of security metrics and presented the major efforts made towards the security metrics of the software systems

CHAPTER 3

Secure Software Development Process & Security Evaluation Life Cycle

3.1 Introduction

Information system encompasses many components like *people, hardware, software, data,* and *networks.* Each of these components must be secure enough for the smooth and reliable functioning of the information system. One of the most important components relative to the all other components of information systems is the software systems. Incorporating the security in software product or system is a long standing challenge for software developers and even more challenging is to evaluate the level of security achieved in the software in development and after the development phases. Developing secure software is not an advantage but has become a necessity for the software organization. Measuring security is necessary in order to mitigate the risk associated with the software **[Chandra et al, 2008].** In Todays networked environment, the software is becoming more vulnerable to both the deliberate and accidental malicious intents. The main reason behind the security holes in the software is due to the negligence of addressing security issues in the software development process. Security is always treated as an afterthought in the software development process, and dealt with after the system development stages by providing the required preventive measures. Security needs to be considered from the very beginning of the software development life cycle **[Wang et al., 2003], [Devanbu et al., 2000]** . Such secure software development process should start with the security requirements, known as NFR (Non Functional Requirements) along with the procedural software requirements, followed by the secure design and implementation. If the secure software development is followed right from scratch, still the level of security achieved remains unknown to the development team. Again the same 122 years old principle of Lord kelvin **[Kelvin, 1891]** "If you can't measure it, you can't improve it", comes into the act. So we need the adequate security metrics that can be applied during and after the software development phases to answer the one big concern," How secure our software is", so that the appropriate corrective measures can be taken earlier, in order to produce the more secure software. In order to measure the security of software system there are three main questions (i) *where it is to be measure?* (ii) *What is to be measured* and (iii) *How it is to be measured*? First question aim at the identification of feasible stage in the software development where the security evaluation process is necessary and must be carried out, second first question is about the identification of baseline to strive for in the measurement process, and

the last question is about devising an adequate mechanism, process, or a framework that can be applied with ease to evaluate the level of security of a system.

In this chapter we analyze the various approaches tools and techniques that can be utilized in secure software development process. In an attempt to answer the first question above, in this chapter we identify the key stages of software life cycle that are candidate for the evaluation of security. The identified stages becomes as a security evaluation life cycle for software systems.

Rest of the chapter is organized as: Section 3.2 presents the processes and limitation of secure software development process in general and the basis for the security evaluation in development in particular. Various stages and tools used in secure software development process are presented in the subsection 3.2.1, 3.2.2, 3.2.3 respectively. Each of these sections is further divided into subsections to present the various software artifacts specification languages in an organized manner. Section 3.3 presents the identification ideal and necessary stages of the software lifecycle for the where the security evaluation is necessary and must be carried, also called as security evaluation lifecycle. Finally the Section 3.4 presents the conclusion.

3.2 Secure Software Development Process and Evaluation

In an effort to produce a secure software product, there are mainly three approaches that are followed by the in real practices, (i) *penetrate and patch* (ii) *secure operational environment* (iii) *secure software engineering* **[Shirazi, 2009]**. The first approach aims at the application of patches after the detection of vulnerabilities in the system. There are two drawbacks to this approach ,first the application of patches results in further vulnerabilities exploitation once the attacker came to know about the patch ,secondly it is believed that fixing the bug after the release of a software is almost 100 times more expensive than putting the efforts in the system development stages **[Boehm, 1987].** Second approach focus on employing the protection mechanisms in the operational environment like firewalls, intrusion detection systems etc. Third approach is actually the most structured way of developing software secure by taking the security right from the scratch i.e. from the requirement to the design, implementation and testing stages of software development.

There is a difference between the *software security* and *application security.* Software security is concerned with addressing the security in the development phases of the software whereas application security is concerned with protecting the software after the development phases by mean of various protection mechanisms like intrusion detection , firewalls etc. Secure software

development focus on the security issues right from the beginning from requirement, design and implementation phases of SDLC. Many ideas of secure software development (SSD) methods have been proposed **[Khan et al., 2009].** However the process of software development still follows the conventional SDLC models and process. In their study **[Khan et al., 2009]** have summarized the various secure software development lifecycle (SSDLC) processes and also provided the detailed comparisons of these processes based on the identified characteristics. Typically SSDLC is comprised of following phases.

1. Software Security Requirements
2. Secure Software Architecture & Design
3. Secure Software Implementation and Coding
4. Security Assurance

3.2.1 Software Security Requirements

It is believed and estimated that an error introduced in the requirement phase can cost up to 100 times more to correct in the further development stages **[Boehm et al., 1988].** A security requirement act as the manifestation of a high-level organizational security policy in the detailed requirements of a specific system. The most current software requirement specifications fall within the following three categories **[Firesmith, 2004]** (i) totally silent regarding security (ii) merely specify vague security goals or (iii) specify commonly used security mechanisms as architectural constraints. In the first case security is not considered in the software development at all, in the second case the specified security requirements are unstructured and very hard to be evaluated. Third case binds architectural decision too early in the at the requirement phase, which results in inappropriate security mechanism. Software security requirements come under the NFRs (Non-functional Requirements) which are quite different from the functional requirements. NFR describe how the software will perform rather than what it performs. NFRs are subjective in nature like maintainability, performance, reliability security etc. From the literature survey, still there exist no concrete, well established and unambiguous non-functional security requirements specification criteria, and also the lack of metrics for specifying objectives tolerance of these requirements, which poses the difficulties in evaluating the system against the specified requirements. If software is developed without consideration of the security requirements, the more work gets shifted to later design phase of the software development to

address and remove such security holes incurred during the requirement phase. This can lead to security flaws in the design phase, because the primary focus of the developers remains on the functional parts rather than the non-functional aspect of the system. Exploitation of security flaws in such software occurs due to the circumvention of security mechanism rather than breaking such mechanisms **[Jürjens 2002].**

3.2.1.1 Security Requirements Engineering

Requirement engineering is the most important and fundamental step towards the quality software development. Security requirements are not precise enough and need to be more explicit about who can do what and when **[McGraw et al., 2004].** Generally software requirements engineering can be split into *requirement development* and *requirement management* **[Wiegers, 2009].** The requirement development involves the activities like gathering, evaluating and documenting the requirements and requirement management entails establishing and maintaining an agreement with customer on the requirements **[Paulk et al., 1995].** According to [**Davis, 1993]** and **[IEEE-STD, 1998]** the characteristics of excellent software requirements have been presented which are as:

- *Completeness:* Every specified requirement should completely describe the functionality that need to be delivered.
- *Correctness :* The specified requirement should accurately describe the functionality
- *Feasibility:* The specified requirement should be feasible in terms of the resources and limitation constraints.
- *Unambiguity:* The end result on the examination of the requirements by different persons should be same , there will be no ambiguity in requirement specification
- *Prioritization:* There should be a priority associated with each requirement with respect to the implementation of the specified requirement.
- *Verifiability:* each requirement should be properly implemented in the product.

The term security requirement has been used as a synonym to attack specification in some earlier studies **[Hussein et al., 2007]**, **[Raihan et al, 2007].** There is a subtle difference between these two, the security requirements are the constraints that need to be implemented in order to control the vulnerabilities in later stages whereas an attack specification represents the sequence

of steps of an attack. There are mainly three approaches **[Diallo et al., 2006]** *Common Criteria (CC), Attack Tree and Misuse Cases*, followed for the specification of security requirement.

Common Criteria (CC): The Common Criteria (CC) is a comprehensive method for security requirement elicitation, specification and analysis. It provides the method for documenting security requirements in a repeatable manner. In Common Criteria (CC), the security environment and security objectives are identified by examining the Target of Evaluation (TOE), which in turn are analyzed to produce the security requirements. The security environment provides the scope and nature of security problem for the system under investigation. The security environment also acts as a basis for the security objectives and the selection of security requirements of a system. The end result produced by Common Criteria (CC) is a Protection Profile (PP) and Security Target (ST) documents. The PP is intended to identify the desired security properties of the system and ST is intended to identify what a system actually accomplishes relevant to the security.

Attack Tree: Attack tree is a systematic process for the identification and specification of security based on the various attacks **[Viega et al., 2001].** In this method the security of a system is outlined by thinking in terms of an attacker's point of view. The attacker's activities and intentions are modeled in the form of a tree which depicts all the possible ways an attacker can adopt to attack the system to achieve a particular goal. In the attack tree the goal of an attacker is at the root of the tree and the different ways to accomplish the goal as the leaf nodes. There are two types of nodes in an attack tree, the AND node and OR node. OR nodes represent the alternative ways whereas AND nodes represent the different ways to achieve same goal. Attack trees have been used for the specification of security requirement of a system by presenting security in attacker's point of view. The attack tree creation to make decision regarding security involves following steps **[Diallo et al., 2006]:**

- Identification of possible goals of an attacker. Each identified goal forms a separate tree, although they may share same sub-trees and nodes.
- Identification of all attacks to achieve each goal and adding the identified attack into the tree. Repeat the process down to all nodes and branches.

Misuse Case: Misuse case is presented in the next section under Software Security Specification Languages.

3.2.1.2 Security Requirements Specification Languages

The specification of security requirements in a discrete and unambiguous form is the main concern of the software security requirements. There are various security specification languages such as UMLsec **[Jürjens, 2004],** secureUML **[Lodderstedt et al., 2002],** Secure Tropos **[Firesmith, 2003],** Misuse Case **[Sindre et al., 2005]** AbuseCase **[McDermott et al., 1999]**, UMlintr **[Hussein et al., 2006]** and AsmlSec**[Raihan et al., 2007].**

(i) ***UMLSec*** **[Jürjens, 2004]:** it is the extension over the basic UML used for the secure software development. The security requirements are specified using various stereotypes, tags and constraints. In this specification, it uses 21 stereotypes to be used to represent the security requirements like *confidentiality, integrity, authenticity, non-repudiation, secure information flow etc*. These stereotypes can be used for a use case diagrams, class diagrams, state diagrams, sequence diagrams, and deployment diagrams to specify security requirements in a UML model.

(ii) ***SecureUML*** **[Lodderstedt et al., 2002]:** This is one more extension over the basic UML used for specifying role based access security policies in a model. Since the security requirement are derived from the higher level organizational security policies that includes the roles and responsibilities on the system, such security policies are used to derive the various role based security requirements. In secureUML there are nine stereotypes used to annotate class diagram with role-based access control information. In SecureUML the constraints are specified using Object Constraint Language (OCL). Unlike UMLSecure in this the constraints can be specified for each of the individual software requirements.

(i) ***Misuse Case*** **[Sindre et al., 2005]:** Misuse Case is also a special enhancement to the UML which represents the undesirable behavior of software. As with the regular use case there is an actor, here the same is replaced by a mis-actor which is an actor with malicious intent, an actor carrying out a mistake etc. In contrast to Use-Case, a Misuse Case describes the malicious behavior of unwanted user. Misuse cases can be applied to identify potential threats to elicit the security requirements. There are two special relations called “prevent” and “detect” which relate a misuse case to a regular use case. The author provided a stepwise process to develop diagrams with both the regular use-case and misuse-case. According to **[Sindre et al., 2005]**, the misuse-case and mis-actors should be specified after the specification of regular use-case. Next the potential “include” relationships between use-case and misuse-case should be identified. After this a new use cases should be specified to detect or prevent the misuse case. These new use case

acts as a specification of high level security requirements of the software called as *security use case* **[Firesmith, 2003].** Following steps are followed to create Misuse Case **[Moore et al., 2001] :**

- Describe actors and use cases in normal way suggested by UML methods.
- Introduce the major mis-actors and Misuse Case.
- Investigate the potential "includes" relations between Misuse Case and Use cases.
- Introduce new Use case for detecting or preventing Misuse Cases.
- Continue with more detailed requirements documentation.

(ii) ***Abuse Case*** **[McDermott et al., 1999]:** This is another extension to UML to represent the undesirable behavior of a software system. It uses the actors and the abuse case to specify the harmful interactions. Like misuse case there is no notational difference between the UML use case and the abuse case diagrams. In his study **[Firesmith, 2003]** proposes that all potential harmful interactions should be specified as separate abuse cases. It also recommend for using a tree structure to elicit multiple approaches. In this the detailed specification of actors which includes skills, resources, and objectives should also be included. It also arguments that abuse case can be used to guide design and testing.

(iii) ***UMlintr*** **[Hussein et al., 2006]:** Another extension to the basic UML for the specification of security requirements is *UMLintr*. It uses stereotype and tags to specify attacks using use case diagrams, There are about 15 stereotypes, among which three are defined for classes along with the tags and twelve for the use case diagrams.

(iv) ***AsmLSec*** **[Raihan et al., 2007]:** AsmlSec is the extension of the original AsmL (Abstract State machine Language) which is based on the extended finite state machine. Attacks involving multiple steps can be specified in AsmL. AsmLSec uses states, events, and transition diagrams to represent attacks. Each transition has a source to destination state, the set of conditions for a transition and action to be performed in case of a transition takes place. Such attack scenarios represented in AsmLSec gets translated into AsmL through a specially developed compiler.

Different security requirement specification languages have different properties and limitations. Decision to choose a particular language among these is based up on these properties. In **[Khan**

et al., 2009] presented a comparative study of various security requirements specification languages based on the properties provided by each of them.

The benefit of the specification of security requirements is manifold, first these requirements make the basis for the secure software development process, secondly these requirements play role in mapping the result of risk and threat analysis to practical security requirement statements that manages (cancel, mitigate or maintain) and measure the security risk of the system. Third the security requirements can act as a scale, against which the evidence of security in a system collected through security metrics can be compared. In other words the precise and unambiguous specification of security requirements, act as the baseline to in measuring the security of system at any of the stage.

3.2.2 Secure Software Architecture & Design

Design is the blueprint of the overall structure of the software, it address the definition of the overall structure of the software. It is the design phase where the decision is to be taken to fulfill the specified requirements. The incorporation of security into the software design process is influenced by both the software design concepts and security methods. Defining the overall structure of the software from the security perspective entails identifying those components whose correct functioning is essential to security and identifying design techniques that will assure its security. Architecture & design specifies two aspects of the software: *Static structure* and *Dynamic Behavior*. According to **[Khan et al., 2009]** in Secure Software Development process (SSDP), architecture & design phase involves following steps.

1. Detailed functional design including the security mechanisms should be specified by following the secure design guidelines and patterns. The security specification languages like UMLsec can be utilized at the end.
2. The design should be inspected repeatedly for the identification and removal of errors.
3. The threat modeling which is normally carried out in the requirement phase of secure software development should also be enhanced and applied in architecture and design phase.
4. Based on the identified threats, the risk analysis should be performed to calculate the potential damage of each of the identified threat.

5. The prioritization of secure design decision should be carried out to remove threats based on cost-benefit analysis.
6. Previous specified secure design decision should be identified. The Security vulnerabilities in the existing similar software can also be used as checklist.
7. A threshold of acceptable security needs to be defined. Security index of the design must be within the threshold.
8. If the calculated security index not satisfies the threshold level, then secure design to remove the errors should be specified.

The secure software design is still in the early stages; in practice secure software design guidelines are widely used. These secure design guidelines are the suggestion and the principles based on the experience the developers have gained while solving software security problems repeatedly. In **[Doan et al., 2004]** proposed three software security design principles that are intended to associate software and security design concepts. The proposed principles are:

Principle 1: The software design has multiple iterative phases and the security features should be incorporated and adjusted during each of those phases.

This author **[Doan et al., 2004]** suggests that the *principle 1*. can be achieved by utilizing the UML (Unified Modeling language). From the UML use case diagrams representing the requirements, the UML class diagrams at the higher level of abstraction with only attributes and methods signature should be specified. The designer needs to return to the UML use case and associate the use case with the classes as required. This may also result in the design of new classes between actors and use case. With the increasing maturity of the design, the use case of requirement phase and the associated design phase classes can be combined together into the sequence diagrams. As the process of design progress further, other UML diagrams such as collaboration, object, state, activity diagrams etc. may be supplied. With each sub phase of the design phase the security level of the design gets improved repeatedly

Principle 2: Security assurance is relative to the phase of software design and the chosen Security Assurance Rules (SARs).

This principle stipulates that the security assurance be evaluated against the respective software design phase to constantly enforce Software Assurance Rules (SARs). The author in **[Doan et al., 2004]** suggests that the *principle 2* can be achieved by utilizing the SARs (Software Assurance Rules). These SARs are derived from the higher level security policies. The idea is, as

the design context changes from one to the next in UML, different SARs are to be utilized to enforce the security features for that sub-phase of design.

Principle 3: The security incorporating process should neither counter the intuition nor decrease the productivity of the software designer.

In literature various principles for secure software design have been proposed. To start with, first set of guidelines proposed in **[Saltzer et al., 1975].** These guidelines have elaborated by providing the examples in **[Bishop, 2004].** In their study, **[Howard et al., 2009]** have proposed secure sets of secure design guidelines. In his study **[Peine , 2008]** have analyzed and refined the guidelines proposed in the earlier studies as result of which a consolidated set of secure software design guidelines by adding and modifying the adequate guidelines from the existing sources and also some newly proposed have been reported.

3.2.2.1 Secure Design Specification Language:

The UML diagrams are heavily used in the design phase of the software development. UML is equipped with verities of diagrams to specify the functional and behavioral aspects of the software. UML is one of the dominant modeling languages used in the software design for specification, visualization, construction of software artifacts. The extended UML such as UMLsec **[Jürjens, 2004],** secureUML **[Lodderstedt et al.,2002],** Secure Tropos **[Firesmith, 2003]**, MisuseCase **[Sindre et al., 2005],** AbuseCase **[McDermott et al., 1999]**, UMlintr **[Hussein et al., 2006],** AsmlSec **[Raihan et al., 2007]** discussed above are also used for the specification of the secure software design. According to **[Khan et al., 2009],** there are two major concerns that should be considered in selecting a secure design language. These concerns are the verities of diagrams available to represent the design from various perspectives with certain level of abstraction and the availability of tools. Verities of tools provided by UMLsec can be utilized for secure software design. SecureUML on the other hand can also be used in secure software design; however it is limited to representing only role-based access control notions in a UML class diagram. Secure Tropos **[Bauer et al., 2001]** proposes verities of use Agent diagrams such as agent interaction diagram, plan diagrams which are similar to the UML activity diagrams and sequence diagrams .With the help of such secure design languages the security of the software can be specified explicitly in both the requirement and design phase in order to promote secure software development.

3.2.3 Secure Software Implementation (Coding)

For the secure software development process the security needs to be taken into consideration from the requirement to design followed by the implementation (coding). Secure software implementation (coding) involves the process and application of secure coding standards and guidelines , application of security testing tools including "fuzzing", static-analysis code scanning tools, and conducting code reviews in order to eliminate the vulnerabilities in the code. Various studies showed that much of the vulnerabilities creep into the system through the poor coding techniques. There is no widely accepted standard(s) for secure programming. Programmers today must instead rely on techniques, examples, guidelines, rigorous testing and an awareness of risks to be avoided, in order to write reasonable secure code **[Graff, 2003].** Many secure coding guidelines have been proposed in the literature but are very difficult to apply in practice. Some of the guidelines spread over the hundreds of rules, some aim to prevent the very specific type or errors. Also the choice of the language used is a key consideration in secure coding process which is beyond the scope of current study. Various secure coding guidelines tend to have little effect on the programmers when they write code and their benefit is very small in practice. In his study **[Holzmann, 2006]** have analyzed the problem with the existing secure coding guidelines and principle as:

- The worst aspect of various secure coding guidelines is that they not allow for comprehensive tool-based compliance check. Tool based check is required, since it is infeasible to check the hundreds and thousands lines of code manually.
- The set of the rules have to be small and clear enough, so that it can be easily understood and applied by the programmers. Regarding this aspect the author argued that, it will be beneficial significantly by restricting the number of rules below or up to ten in numbers.

In the same work **[Holzmann, 2006]** proposed a set of rules for safety critical coding. This set comprises of ten rules along with the brief rationale for the inclusion of the each of the rule in real practices.

Even if the above stated secure software development process is to be followed, still the software are produced with inherent vulnerabilities in them. The problem behind this is the lack of tools and techniques to evaluate the system for security and to analyze the result of evaluation both, before and after the application of any security mechanism. The evaluation of security before the application of any security mechanism is required to predict the most critical components of the

system to make an improvement and the evaluation afterwards is required for the assurance of the applied security process.

3.3 Proposed Security Evaluation Lifecycle

Measurement of both the product and the development has been recognized as a critical activity long back for successful software development. Appropriate measure and indicator of software artifacts such as requirement, design, and implementation can be analyzed to diagnose problem and identify solutions during the development and reduces defects, rework and cycle time **[McCurley et al., 2007].** Security is not a side role syndrome; instead it should be taken care during the development phases of a system. There is nothing like 100 percent secure. Adopting the secure software development process does not completely eliminate the problem, because adopting the secure development rely on the knowledge of the development team regarding the secure software development process, from the requirement phase to the coding. Security consideration in the system development phases is crucial for secure software but even if the secure software development process is to be followed still the level of security remains unknown to the development team. Along with the software development phases we need a mechanism to evaluate the security of the system to answer the questions like, how secure our system is? How secure it need to be and whether the necessary corrective or preventive measure needs to be taken depending on the result of the evaluation. The answer to these questions is only possible if we have such security metrics at hand that can be applied with ease to get the desired result. Without the security metrics, the security evaluation is dependent on the activities like threat modeling, Risk analysis, design decisions to mitigate risks and conducting inspection which depends upon the knowledge of the persons involved. On the other hand if we have well developed security metric framework that can act as tool for security evaluation and requires very less expertise of the personnel involved in the evaluation process can certainly eliminate the security problem to a great extent. Unfortunately, useful security measurements related to the development of system are in their infancy and no consensus exists as to what measures constitute best practice. A useful measurement related to the development of products coded to meet the security is a multifaceted issue. As far as security measurement is concerned, it must be carried out along with each phase of the software development phase. We argue that the security metrics are the core to the secure software production and along with the each phase of secure

software development process there should be a security metric framework that should act as a tool which take the software artifacts as in put input and provide preferably the quantative indicator about the security posture of the system and its components. Figure 3.1 shows the applicability of security metrics in secure software development process. It also provides the answer to our first question *"Where in the life cycle of a system security evaluation is to be carried out".*

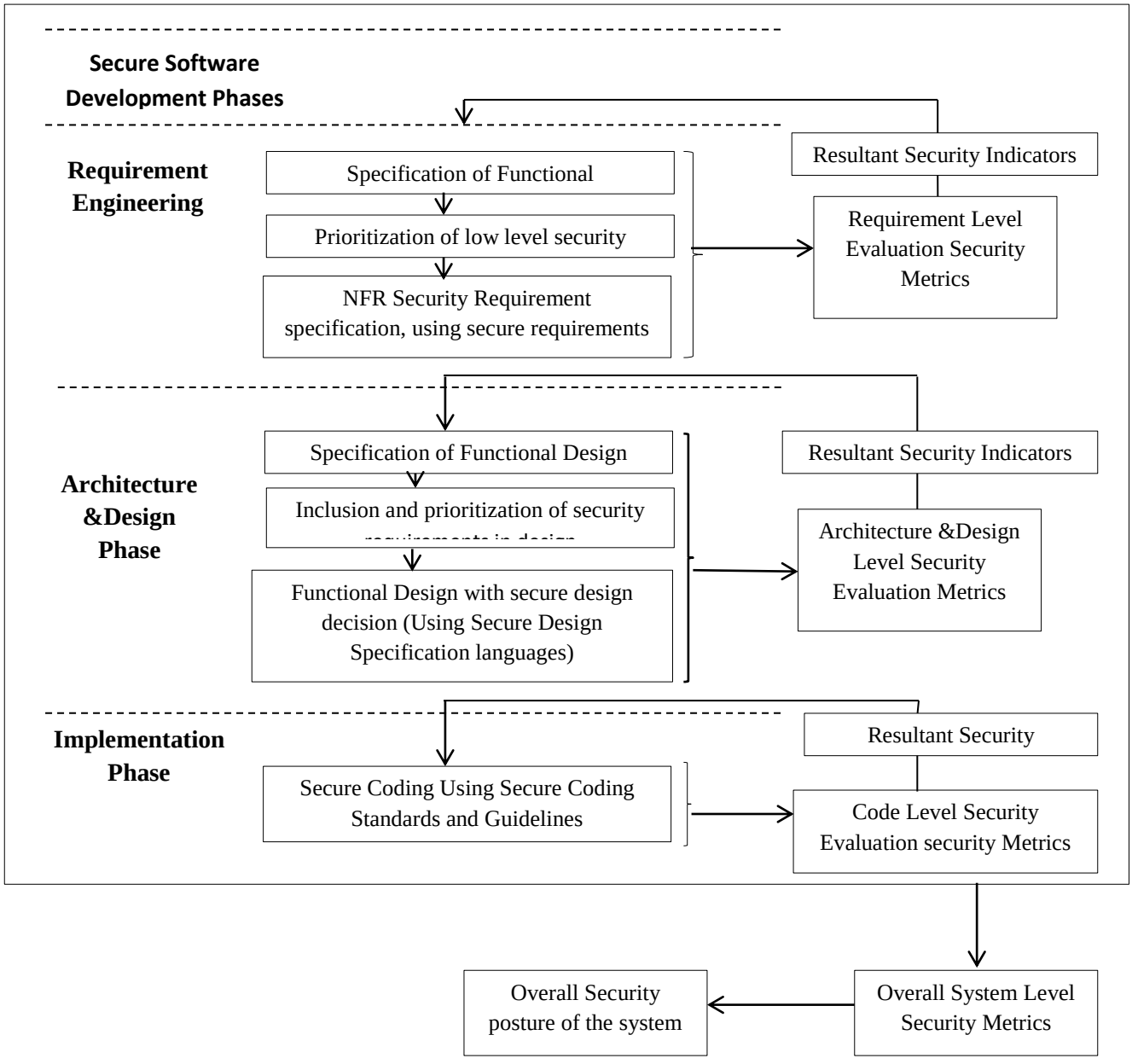

Fig. 3.1 Security Evaluation Life-Cycle in SSDP

According to our argument there are four key stages of a system life cycle where the security metrics are necessary & play vital role in order to produce the secure software. As depicted in above figure (3.1) the four key stages of system lifecycle that are the candidate for the evaluation of security are:

(i) Requirement Level Evaluation Security Metrics.

(ii) Architecture & Design Level Security Metrics.

(iii) Code Level Security Metrics

(iv) System Level Security Metrics.

3.3.1 Requirement Level Security Metrics

Some might think that requirement phase security metrics are unnecessary, but the security requirements act as a basic driving force to develop the secure system. The specified security requirements must be evaluated to check the level of security in the requirement specification. From our experience and the literature survey, there exists no such security metric framework that can provide the security indicators by evaluating the specified security requirements. Because the requirements specification involves much of the personnel interpretation about the security and the effect of the changing environment in which the requirements have been elicited, the evaluation process seems to be very complex. An ideal requirement phase security metrics framework should be able to assess the level of security present with respect to well-defined security attributes of security like *confidentiality*, *integrity* and *availability* preferably on a discrete and quantitative scale. The output of the evaluation should depict the level of security in the requirements with respect to the specified security attributes.

3.3.2 Architecture & Design Level Security Metrics

The specified requirements lead to the design phase of the system. Architecture & design phase is the most critical phase of the software development. It acts as a blueprint for overall system's functionality and behavior. The quality of a software system is heavily dependent up on the quality of its architecture & design. A security flaw or hole at this level can lead to the vulnerabilities at the next levels of system life cycle and can cause the severe damage. Like other quality attributes, evaluation of system for security at this level is very vital to reduce the cost and efforts required at the further stages of the system life cycle. The security metrics that can

measure security level of each of the component of the system, the design activities involved and identify the critical elements of the system, are highly desirable. Further such metrics that provide the development team with preferably the quantative indicators of the level of security can certainly provide the basis to the software developers to take the necessary corrective measures early in the system life cycle in order to produce the more secure system, that can withstand both with intentional and unintentional malicious intents. The metrics should be easy to apply i.e a non-security professional should also be able to easily apply the security metrics. In their study **[Chandra et al., 2008]** proposed the security estimation life cycle for the design software systems. It comprised of three stages as:

Stage 1: Input Process

- HLD/LLD

Stage 2: Security Estimation Process

- Identify Security Factors
- Identify/Design Metric Suite
- Validate Metric Suite
- Quantify Security Factors
- Estimate Security

Stage 3: Output Process

- Qualitative Analysis
- Overall Security Analysis

The security estimation life cycle begins design diagrams, High Level Diagrams (HLD) or Low Level diagram as the input to the estimation process. In the second stage the process to use security metrics comprises of five steps. First the identification of the factors that needs to be measured followed by the design metric suite. The design metric suite is actually a set of metrics derived from a security evaluation framework to be used as a tool for evaluating the security. The selection of the metric framework is followed by the validation of metric suite, to validate and check the effectiveness of the metric selected metrics suit. Validation process is followed by the quantification of security factors and the estimation of final security. The proposed security estimation **[Chandra et al., 2008]** can be applied in iterative manner until certain predefined security level is not achieved.

Identification of security factors and the selection of metric suit of above security estimation lifecycle is in itself a critical and hot research area. Such security metrics that can evaluate each and every aspect of the system architecture & design is very complex problem and is still in its infancy stage. Considering the importance of a system architecture and design, very few efforts have been made towards the security metrics that can be applied at this phase of the system life cycle. The main focus is either remained at the top overall system level or the lower code and implementation level. In chapter 4; we have proposed a security metric framework for the architecture & design phase of software development. The proposed framework somewhat follows the same security estimation life cycle and is easy to be used by the development team.

3.3.3 Code Level Security Metrics:

No matter how secure the design is, still vulnerabilities get introduced into the system through poor coding techniques and language specific issues. There are various factors responsible for the induction of vulnerabilities in the code. The poor programming skills i.e. programmers having least knowledge of secure programming, language specific security issues, deliberate attempt to put backdoors etc. Design phase security metrics are intended to evaluate the security issues pertaining to software artifacts at design phase, whereas code level metrics are intended to look specifically into the structure of code and implementation language issues. Varieties of security metrics to measure the code level imperfections have been proposed in the literature, with varying level of achieved success. The main difficulty with code level security metrics is that each programming language used having its own issues and complexities, which may not come under the same evaluation framework. As an example in their work **[Chowdhury et al., 2008]** proposed three set of metrics, Stall Ration (SR), Coupling Corruption Propagation (CCP) , and Critical Element Ratio (CER) in order to evaluate the security of the source code. The proposed set of metrics not measure the security of source code directly, instead it measures the code quality properties that can enhance program security. The primary objective is to quantify the intentions of an attacker in attacking the system by vulnerabilities in the code. In another attempt **[Alhazmi et al, 2006]** proposed a metric known as *vulnerability density* (VD) metric. It is defined as the number of vulnerabilities in the unit size of code. Based on the VD a set of metrics have been derived such as *vulnerability discovery rate* which shows the number of vulnerabilities reported per unit time , *known vulnerability density* (VKD) which are the number

of already known vulnerabilities, *residual vulnerability density* (VRD) which is calculated as: VRD= VD-VKD.

3.3.4 System Level Security Metrics:

System level security metrics take the whole system as a single entity as an input including the environmental issues in which the system operates, after the development phases. From the literature survey it is evident that most of the security evaluation of the software systems has been carried out at the system level. System level security metrics quantify the security status of the system as whole and such security metrics are also known as high-level metrics. An example of system level security metrics is measuring the attack surface of a system **[Manadhata et.al, 2007].** The author proposed security metrics at the system level by quantifying the system security in terms of three kinds of resources used to carry out an attack on the system: *Methods, Channels* and *Data*. The system level security metrics are also intended to look into the system environment beyond the boundaries of technical software components. These issues covered under such metrics ranges from the current system configuration to even the percentage of employees with significant security responsibilities who have received the training **[Chowdhury et al., 2008].** As an example the System Vulnerabilities Index (SVI) is calculated by evaluating factors such as system characteristics, potentially neglectful acts and potentially malevolent acts as a measure of computer system vulnerability **[Alves-Foss 1995].**

Analyzing the security metrics efforts made according to the identified stages of a system life cycle, the design and architecture level is the most critical to the security of a system and evaluation of security at this level is even more crucial in order to make an improvement. Very least progress towards evaluation of security has been made at the architectural and design phase of the system.

In their study **[Hamilton et al., 2010],** authors identified the major obstacles in the Secure Software engineering (SSE) process. The goal of the authors is to function effectively when faced with security requirements and proposed a list of system engineering security capabilities to improve the SSE. As shown in below figure (3.2) each of the listed capability should be well-defined individually and then must be combined together to demonstrate value in addressing overall problem of evolving system security engineering practices, The proposed list of

capabilities (figure 3.2) , provides a conceptual foundation for the research roadmap in secure software development.

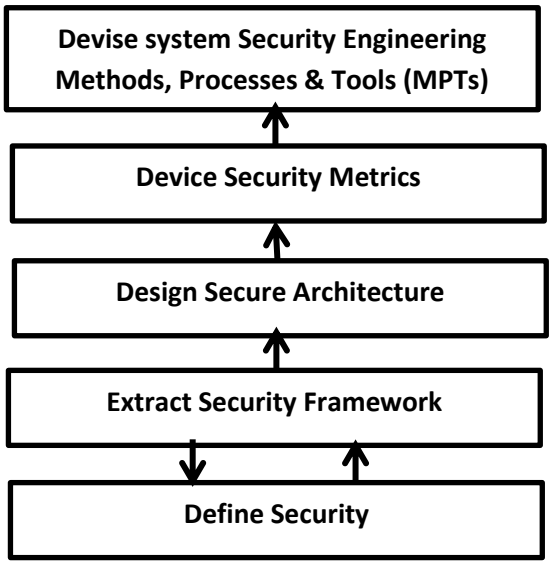

Figure 3.2 Capabilities to Improve SSE

As evident from the above figure (3.2), security metrics are corner stone for the secure system engineering. After defining the secure architecture, security metrics are needed in order to apply the necessary security engineering methods, processes and tools.

3.4 Conclusion and Future Scope

There are two main concerns in secure software development process. On one hand as the new attacks and vulnerabilities are emerging, efforts are required towards the invention of new and updated protection mechanisms to cope up with these attacks. On the other hand the system engineers and user needed the assurance regarding the achieved security level. For such assurance, software developers must be equipped with effective tools and metrics to evaluate the security posture of a system in order to take necessary preventive or corrective measures. Both dimensions of security must be carried out in parallel to improve the state of art. In this chapter we have analyzed the various tools and techniques applied to the secure software development process and identified the key stages of the system life cycle where the security evaluation is necessary in order to promote the secure software. From the survey of literature it is evident that there is no systematic approach to measure the security in the system development phases

especially at the software architectural and development stages. Lots of efforts are needed to device the metrics at each of the identified stages of the system lifecycle, which should take the software artifacts as an input and provide the resultant security indicator in a controlled manner.

CHAPTER 4

Security Metrics Framework: Architecture & Design Level

4.1 Introduction

The most challenging task in the software development is the transformation of both the functional and non-functional requirements into the design and architecture of the software compared to other activities. Design and Architecture depicts the overall structure of the system by transforming the higher level requirements into the design artifacts. The main issue during the design and architecture phase is to identify the core abstractions from the requirements based on which system is structured. These core entities are very hard to find and needs the investment of skills and creative design process **[Bosch et al., 1999]**. Once the abstractions are identified, the detailed interactions between them are defined. Decisions made at the architectural and design level are the earliest decisions in the software development process and can have the great impact on the software quality [**Williams et al., 1998].** Software Design act as a blueprint for overall system development activities, a minor imperfection in the design can cause a serious damage and requires extra efforts and cost to mitigate the risk in later stages. For example fine tuning only the code without any consideration of security evaluation at design phase to neglect the security risk can eliminate some risk but the developed system will not remain secure for much longer period of time , because new vulnerabilities gets introduced with time once the weakness in the architecture and design gets identified. From the literature and our experience most of the security vulnerabilities start stemming into the system due to lack of consideration of security issues in early development stages especially at the design and architecture phase of a system. As stated by **[Williams et al., 1998]:**

"Whether or not a system will be able to exhibit its desired (or required) quality attributes is largely determined by the time the architecture is chosen."

So system architecture and design should be evaluated for the security attributes in order to answer such question and to take the necessary corrective measures. Evaluation of a system design and architectures for the quality attributes such as reliability, reusability, modification performance, response time, throughput etc. remained a central focal point of the research community. Security specification in the requirement phase through both the Functional Requirements (FR) and Non-Functional Requirements (NFR) once transformed into higher level

architecture and design, needs to be evaluated for the security to ensure that the design meeting the specified requirements and to take any corrective or preventive measures if required. The security evaluation framework and the metrics should act as a tool for the development team without requiring any personnel expertise, because most of the software developers focus much on the functionality and having least knowledge about the security issues. More over the result of the metrics should be reproducible i.e. the result of the evaluation should remain same if carried out by multiple persons.

In chapter three (Section 3.1), we have posed three questions regarding the security evaluation of a system: *Where in development stages the security evaluation is to be carried out?* The answer to this question is given in Chapter (3) by identifying the key stages of a system life cycle where the security evaluation is necessary. Second and third question," *What is to be measured in order to evaluate security posture of a system"* and "*How to measure security* "are concerned to find out the factors or attributes against which the security is to be evaluated and a mechanism to evaluate the security respectively. In this chapter we answer the second and third question by proposing a security metric framework and derive the security metrics for the architectural and design phase of the software development using mathematical modeling techniques. Our evaluation framework in based on the Component Based Architecture and Design (CBAD) and Component Based Software Engineering (CBSE), because we believe that Component Based Architecture Design (CBAD) provides moderate level of abstraction , which is neither too coarse-grained like Service oriented Architecture & Design (SOAD) nor too fine-grained like Object Oriented Architecture and Design (OOAD). Also it requires least efforts to transform the other software architectural and design specification into CBAD.

The rest of the chapter is organized as: In identifying the baseline to strive for in the security evaluation process Section 4.2 provides the answer to the question, what is to be measured? Since the proposed evaluation framework aim at the architectural & design phase of system life cycle, Section 4.3 presents the various architectural and design techniques. In the Subsection 4.3.2 the decision regarding the selection of architectural and design has been presented. Section 4.4 prsents the components based software architecture the various issues and modeling techniques present. Section 4.5 presents the proposed security evaluation framework. Subsections 4.5.1 and 4.5.2, presents the derived security metrics for four major security attributes (*dependency, availability, confidentiality and integrity have been proposed)* using

mathematical modeling techniques. Section 4.6 prsents the security metric framework with respect to these attributes in a semi-automated, algorithmic form. Finally Section 4.7 prsents the conclusion and future work.

4.2 What Is To Be Measured?

The first and foremost step towards any performance related software evaluation process is identification of the factors that should act as a base line for the evaluation process. The factors responsible for the security vary from system to system and are heavily dependent upon the target environment. How a system should behave comes from the higher level organizational policies and the security threats, which are ultimately, framed into the system's requirements especially the non-functional requirements. Security requirements provide the foundation upon which the security of the system is built. Since both the functional and non-functional security requirements provide the overall specification about "what constitutes a system "and "how should the system behave" respectively. One can think of a security evaluation framework that measures the security of a system against specified non-functional security requirements but it is not that simple, because of the difficulty incurred in the specification of non-functional security requirements itself, which bounds the evaluation to a particular type of system, hence limits the scalability of the security evaluation process. The reason behind the immaturity of security requirements specification is the lack of concrete, well-established standard for the specification of security requirements. Also devising a security evaluation framework that evaluates security based on the specified security requirements limits the usability and applicability of evaluation process to a specific domain and environment. The factors for which system engineers strive for should be clearly specified in order to evaluate the system. From our experience and literature, we believe that the all the specified security requirement for any type of system should ultimately come under one or more well-defined and well-established security attributes as shown in below figure (4.1). The difference lies in the context in which these attributes are to be defined such as network, software, environment etc. Our security evaluation framework **(**is based upon the fundamental security attributes, because we believe that security evaluation against these well-established attributes certainly improves the scalability of the proposed framework.

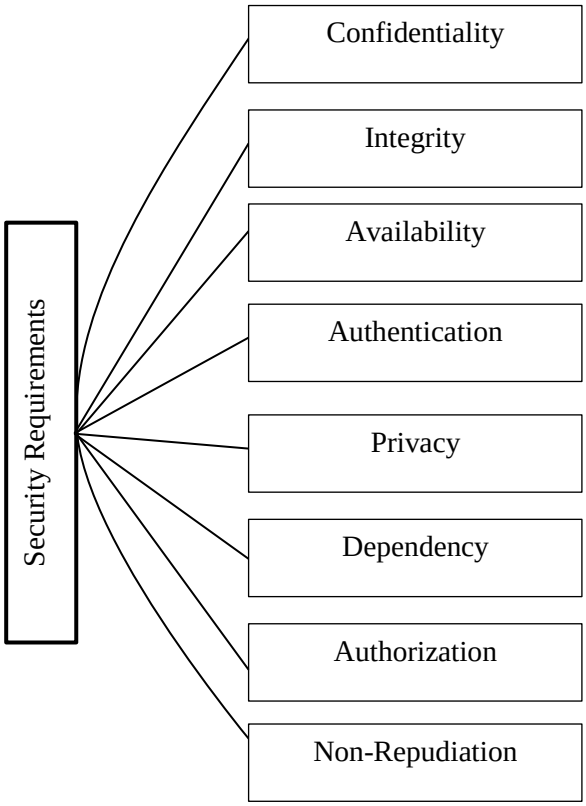

Fig. 4.1 Software Security Requirements Span over Security Attributes

Figure (4.1) depicts the span of non-functional security requirements over the well-established security attributes. The security is not only limited to these attributes, instead the notion of security encompasses both the traditional security attributes and classical dependability attributes **[Nicol et al., 2004].** Among these attribute the *confidentiality, Integrity*, and *availability (CIA)* are the fundamental to the security. Other attributes are also important to the security but are the extension over the basic CIA to encompass each and every aspect of security of a system. In our proposed security evaluation framework we evaluate the system and its components against these specified security attributes.

4.3 Security Evaluation: Software Architecture and Design Phase.

In this section we present our security evaluation framework for the architecture and design phase of the software development. In answering the question above section (4.2), the specified security attributes are the baseline for our security evaluation framework and are defined later in

the due course of time. Since our evaluation framework aim at the software architectural and design phase, so our foremost focus is to identify the commonly used software architecture and design techniques that have been adopted by the developers in real practices and their feasibility in security evaluation .

4.3.1 Software Architecture and Design

Constructing a software system by composing prefabricated or newly developed components is always an initiative and attractive vision for software development **[Mei et al., 2000]**. There is a consensus on the fact that for any large software systems the overall structure, i.e. the high-level organization of computational elements and interaction between those elements is the critical aspect of the design **[Perry et al., 1992].** Many definitions and concepts regarding the software architecture have been mentioned in literature. These definitions of software architecture not only acknowledge structural elements but the composition of architectural elements, their relationship, the connecters needed to support their relationships, their interface and their partitions again. Architecture is a fundamental organization of a system embodied in its components, their relationship to each other and to environment and the principles its design and evolution **[Jen et al., 2000].** Software Architecture of a system is a depiction of the system that aids in the understanding of how the system will behave. Obviously the first step to make a system good is to make the design good. The quality of software is highly dependent on the architecture defined in the early stages of the development process. The architecture can influence the functional as well as the non-functional requirements. The correction of bad/wrong design decisions takes a lot of efforts, therefore it is useful to be able to analyze software architecture in an early stages of the software development process **[Muskens et al., 2002**]. Software architecture and design serve as a blueprint for both the system and the project developing it. The architecture is the primary carrier of system qualities such as performance, modifiability, and security. These qualities cannot be achieved without a unifying architectural vision. Architecture is an artifact for early analysis to make sure that the system design yields an acceptable system. There is a slight difference between a software architecture and software design of a system. Software architecture is more about what we want the system to do and software design is about how we want to achieve that. Architecture is at higher level of abstraction than the software design. Software architecture is concerned with issues beyond the

data structure and algorithms in the system. There are various definitions and understanding about software architectural in the literature. Some of them are as:

According to **[Bass et al., 2003],** Software architecture of a system is the structure or structures of the system comprise of software components, the externally visible properties of those components, and the relationship among them. The externally visible properties are the assumption such as provided service, performance characteristics, resource sharing that other components in the architecture make about a component. According to the definition the purpose of software architecture is to provide an abstraction to hide some information from the system (otherwise there is no point looking at the architecture, we are simply viewing the entire system) **[Muskens et al., 2002]** and yet provide enough information which act as the basis for analysis, evaluation, decision making, and hence risk reduction.

According to **[Booch et al., 1999]** an architecture is a set of significant decisions about the organization of a software system, the structural elements and their interfaces by which the system is composed together with their behavior as specified in the collaboration among those elements,. Further the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization.

In his study **[Kruchten, 1995]** presented a model *"4+1 view model"* for describing the architecture of a software intensive system based on the concurrent views. Software architecture is in place to deal with the design and implementation of high-level structure of the software. It is the result of assembling a certain number of elements in some well-chosen form to satisfy the major functional as well as well as non-functional requirements. The *"4+1 view model"* proposed by the author addresses large challenging architectures which is made up of five main views as shown in figure (4.20. The first four views are the core around which the software architecture can be organized and the fifth view is for the illustration by use cases, or scenarios. The core views are:

- *Logical view*
- *Process View*
- *Physical View*
- *Development View*

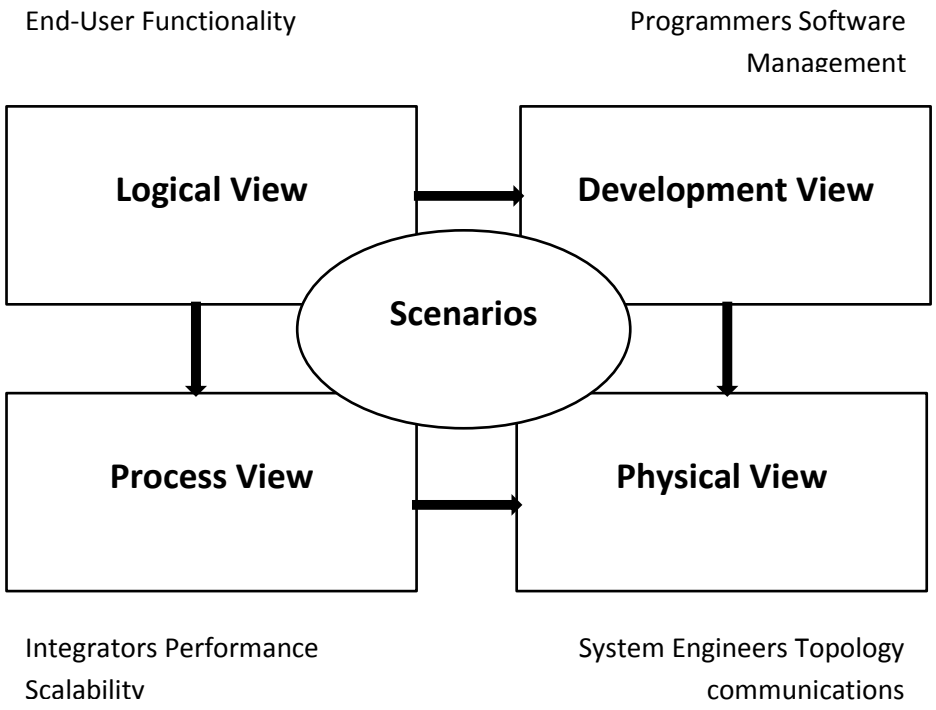

Fig. 4.2 "4+1" View Model for Software Architecture

Logical View: Logical architectural view supports the functional requirements of the system and depicts the details about what services it provides to its users. The system is decomposed into a set of key abstractions in the form of objects and classes. Beside the functional decomposition in the various components, the logical view shows the logical dependencies among the components. Later in our Security evaluation framework we have used the component diagrams comprised of component, relationships, interface and description defined and discussed later in the due course of time.

Process View: Process view architecture covers some non-functional requirements. It captures the concurrency and synchronization aspects of the design. Process architecture of a system can be described at many level of abstraction each addressing different concerns. At the higher level , it can be viewed as a set of independently executing logical networks of communicating programs (processes), distributed across a set of hardware resources connected by a LAN or WAN.

Development View: Development view of the architecture focus on the actual software module organization on the software development environment. The software is packaged into small chunks like program libraries or subsystems developed by a small number of developers. The subsystems are organized in a hierarchy of layers, each layer providing narrow and well-defined interfaces to the layer above it. The development view serve as the basis for the requirement allocation, allocation of work to teams, cost evaluation and planning, progress monitoring of the project.

Physical View: The physical view is concerned with the non-functional requirements of the system such as reliability, performance and scalability. The software executes on computers in a network or processing nodes. The various identified elements such as tasks, processes and objects need to be mapped on to the target processing nodes. There may be several different physical configuration will be used. The mapping of software to the nodes needs to be highly flexible and have a minimal impact on source code.

Scenarios: Scenarios are intended to put together the four basic views to work together. Scenarios are the instances of the of more general use cases and in some sense an abstraction of the most important requirements. This view is redundant with the other four (hence +1) but it is there to provide:

- A driver to discover the architectural elements during the architecture design.
- A validation and illustration role after this architecture design is complete, both on paper and as the starting point for the tests of architectural prototype.

The "4+1" View model **[Kruchten, 1995]** for software architecture covers almost each aspect of the software and with the diverse range of notation such as objects, classes, components, sequence, and other UML diagrams to present the both functional and behavioral aspect of the system and makes this architectural style best for the architectural and design quality assessment.

4.3.2 Architecture and Design Selection

In recent years, much of the attention in the software engineering community has focused on design approaches, processes, and tools supporting the concept that large software systems can be assembled from independent, reusable collections of functionality. Some of the functionality may already be available and implemented in-house or acquired from a third party, while the remaining functionality may need to be created **[Brown et l.,. 2002]**. As mentioned earlier

Software Architecture and design can influence the functional requirements as well as the non-functional requirements. Architecture and design is the best level for analyzing and evaluating the various quality concerns of a system. Since there are various software architectural and design modeling techniques among which the Object oriented Analysis and design (OOAD), Enterprise Architecture Framework (EA), Service oriented Architecture and Design (SOAD) have been presented and adopted in the software development process, Component Based Design(CBD) are the most common. The question is which of the architectural and design modeling technique to be followed for the specification of system structure so that it will be feasible for the analysis and evaluation of quality attributes. Since each of the modeling technique such as OOAD, SOA, CBD and EA, have their own merits and range of tools used for the specification of the architecture and design, choosing one style over other for the security evaluation is a very difficult task. Among the above mentioned architectural and designs approaches, Service Oriented Architecture (SOA), Component based development (CBD) and Object Oriented Analysis and Design (OOAD) are the core software architectural and design modeling techniques extensively adopted by the software developers. The SOA, CBD, and OOAD are not isolated from one another; instead they can be applied progressively in the system development, each with different level of abstractions. Below diagram 4.3 **[Brown et al., 2002]** shows the application architecture layers of the software architectures. As depicted in the diagram 4.3. Three level of technology layers are there for application architecture. At the higher level there is a service level which act as a great way to expose an external view of a system, with internal reuse and composition using traditional component design which may in turn use the object oriented design. The three layers (fig. 4.3) of application architecture are:

- *Service Level*
- *Component Level*
- *Object/Class Level*

Service Level: Service layer provides the higher level of abstraction which provides the functionality with a loose coupling among interacting software agents to its client. Service layer is the most coarse-grained view of the software architecture and design i.e an abstraction over the lower level design details and service encompasses more functionality and huge set of data compared to the Component Based Architecture &Design (CBAD) and Object/ Class Design . Service oriented Architecture (SOA) is a paradigm that organizes and utilizes distributed

capabilities, which may be under the control of different ownership domains, implemented using a variety of technology stack **[OASIS, 2009]**, **[Kanneganti et al., 2008].** In SOA a service encompasses more functionality and data sets to operate on as compared to the component-based design.

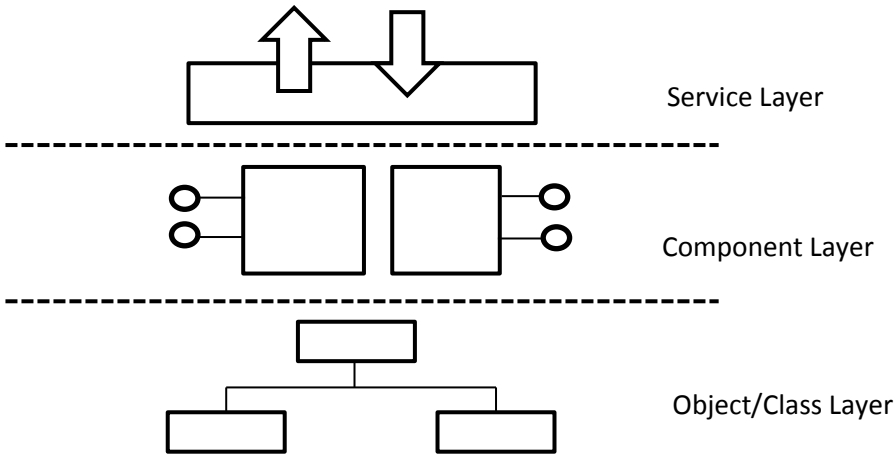

Fig. 4.3 Application Architecture and Implementation Layers

Component Layer: Component layer is at the second level of abstraction; both SOAD and Component Based Architecture and Design (CBAD) support certain level of abstraction, the service via contracts and components via interfaces. The difference lies in the level of abstractions provided by the both. On one hand Service exhibit higher level of abstraction where as a component exhibit comparatively lower level of abstraction. In one may think of SOA as a high level form of CBAD, which internally designed as the interacting components. A component is a software object, meant to interact with other components in the system composition to provide certain functionality to its client or other components. A component has clearly defined interfaces (both provided and required) and conforms to a prescribed behavior **[Petritsch, 2006].**

Object/Class Layer: Object/Class layer is at the lowest level of software architecture and design layers. This is the most fine-grained level of software architecture and design with comparatively very low level of abstraction. In Object/Class design a system is composed of objects and the behavior of the system is achieved through the collaboration between these objects by sending messages to each other. Object oriented paradigm can be found at the design and implementation level of software development. At design level Object Oriented (OO)

enables the rapid and efficient design, development and execution of applications that are flexible and extensible. From an OO perspective everything is an object. The Fundamental principles of OO are **[Zimmermann et al., 2004]:**

- *Encapsulation:* The Binding of physical properties (data) and the functionality (behavior) together.
- *Information Hiding:* Hiding internal mechanisms by simple interfaces.
- *Classes and interfaces:* Classes act as a template to define the properties and behavior of a software object and instances are the actual individual object having values for those properties.

The architectural layers in above fig 4.3 do not mean that one architectural style is better than others, or limits the applicability of one architectural and design style over other. Instead they differ in the level of abstractions provided by each layer, ease of use and to encompass the functional behavior of the system. The level of granularity differs at each layer, on one extreme (Object and Class) the level of granularity is focused at class level, which is at the too low level of abstractions for modeling business logic and the strong association between objects crates the tight coupling between them. On the other hand the service layer is rather more loosely coupled; agile in nature encompasses more functionality. In spite of the difference between these layers of software architecture there is the dependency and association between them. If Service Oriented Architecture (SOA) is to be followed still the Object Oriented and Component Based approach are needed for the underlying design of the classes and component structure with in a defined service.

Choosing particular software architectural and design approach for the evaluation of quality attributes in general and for the evaluation of security in particular, is highly dependent up on the levels of abstraction provided by each of them. On one extreme the SOAD is much coarse-grained in nature with granularity focused on the loosely coupled services which are at the very high level of abstraction hiding the internal functional units of the service. Selection of SOAD of a system for the evaluation of security restricts the evaluation process from capturing the lower level details of the system. On the other extreme Class/Object level provides the granularity at the class and object level which is very fine-grained view of a software architecture and design. Selection of Class/Object level architecture for the evaluation of security , no matter provides the

lower level details of the design and architecture of the system but at the same time, it makes the evaluation process too complex and bind the lower level details too early at the architectural and design phase of the system. The Component Layer in above diagram 4.3 is in between the two extremes. A service internally may exhibit the components composed together in a composition to provide the required service, these components may internally build by the lower level classes and objects. The position of Component Based Architecture and Design (CBAD) in the architectural hierarchy are neither too coarse-grained like services that hide the internal functional units nor too coarse-grained to make the evaluation process too complex to bind the lower implementation level details. From the literature survey and our experience, The Component layer is the best suited for the evaluation process of any quality attribute of the system. Keeping in view the various factors, like granularity, level of abstraction, level of functionality, flexibility provided and required efforts, in our Security Evaluation Framework we have chosen the Component Based Architecture and Design (CBAD) approach of software architecture and design for the specification of design and architecture of software artifacts. In next section we provide an overview of Component based Architecture and Design and the various modeling techniques involved.

4.4 Component Based Software Architecture and Design (CBAD)

As mentioned previously Verities of system modeling disciplines such as Object Oriented Analysis and Design (OOAD), Enterprise Architecture Framework (EA), Service Oriented Architecture and Design (SOAD) have been presented in the literature and are applied in the software development process. Component Based Architecture and Design or Component Based Software Engineering (CBSE) is a successor of OOAD **[Szyperski et al., 2002]** and has been supported by commercial component frameworks such as Microsoft's COM, Sun's EJB and CORBA. In Component based Architecture and Design (CBAD) the fundamental unit of a large scale software construction is a component. The system in CBAD is structured as a collection of components and their interconnection and composition. A software component is a unit of composition with contractually specified interfaces and explicit context dependencies [95] **[Lau L. et.al, 2007].** Similarly according to **[Meyer, 2003]** "A component is a software element satisfying the following conditions:

- It can be used by other software elements i.e. its clients.

- It possesses an official usage description, which is sufficient for a client to use the provided services.
- Not tied to any fixed set of clients.

The abstract view of a component is shown in below figure 4.4. It consists of three main parts

- *Component Name:* The specified name of the component.
- *Code:* The functionality of the component.
- *Interfaces:* Both the required and provided interface to reveal the usage and functionality of the component in the system composition.

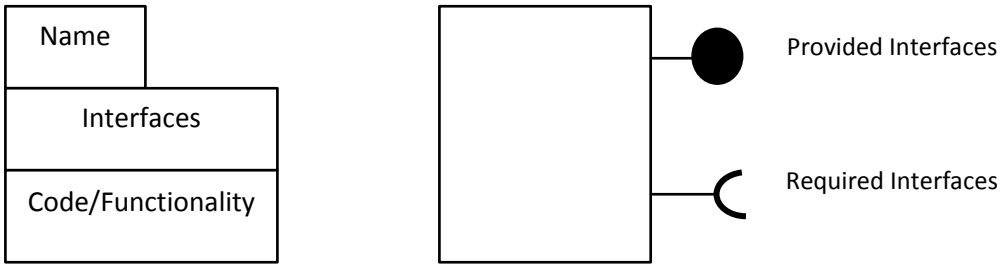

Fig. 4.4 Software Component Model

Above figure (4.5) shows the abstract view of a component. From the developer's and designer's point of view a component can be viewed as shown in below diagram (4.4) **[Abdellatief et al., 2011].** From the figure 4.5 component specification defines the behavior and functionality of the component. The specification is composed of a component body and interface part. The specification of the interface is visible to the component designer, and on the other hand specification of body and interface are visible to component developer.

In his study, **[Tyan, 2002]** the author related the software design modularity to hardware modularity which changed the programmer's view of software from sequential execution of statements to a collection of objects and their interactions. But the design realizations of large object software are subject to so called *hyper spaghetti* objects and *subsystem phenomenon* where the tight coupled objects/classes in the system are virtually impossible to extract from the system for reuse **[Tyan, 2002].** On the other hand a component in the component based architecture constitutes the partition of large software systems which closely mimic the design of digital circuits in terms of the assembly and specifications of components.

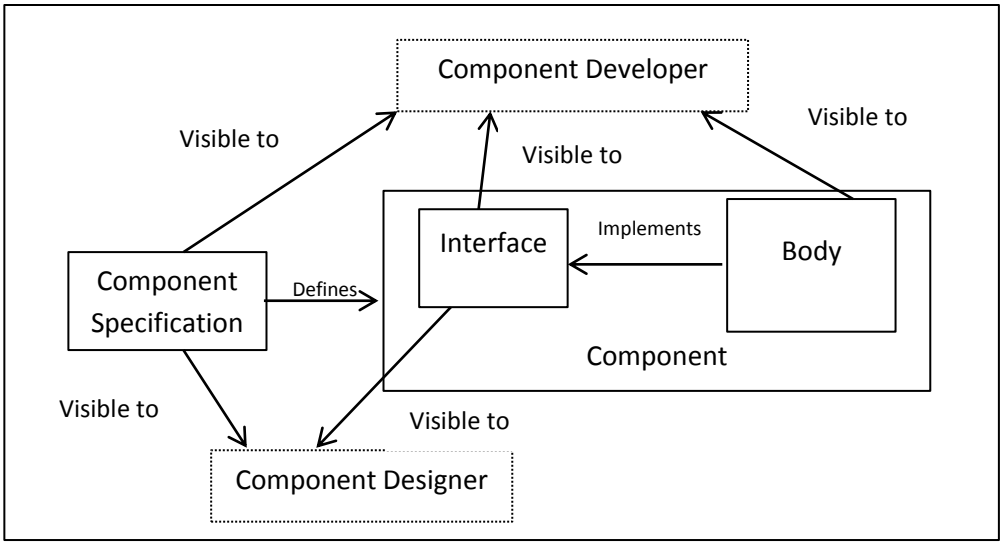

Fig. 4.5 Software Component: Designer & Developers Perspectives

In the Component Based Software development the code is organized as a collection of components. Each Component respond to its incoming data through its entry points also known as ports or required interfaces and provides its data external through ports also known as provided interfaces. According to **[Tyan, 2002]** component based software design brings the following benefits:

- ***Composability:*** Software system is composed of software components that can be independently constructed and interconnected together to provide the business functionality.
- ***Evolvability:*** Flexibility in terms of replacement of component in the system composition, i.e. an existing component can be easily replaced with the upgraded version without disturbing other components in the composition.
- ***Extensibility:*** The addition of a new component can be done easily which requires the right interface specification
- ***Testability:*** As mentioned earlier Component Based Software Design realization makes it very feasible for testing and evaluation of quality attributes. Instead of evaluation the system by tracing the lines of code, components with the input and output ports are best suited to probe various quality related evaluation.

4.4.1 Software Component Models

According to **[Broy et al., 1998]** the Component Based Software Design (CBD) lacks in universally accepted terminology, particularly it lacks in a standard criterion for what constitutes a component. The widely accepted definitions of components are provided without a particular software component model. The concept of reusability in software has remained an issue for many years. The components offer a mechanism to support software reusability to a greater extent. However at the same time a number of questions rise such as: How to describe components? How to bind them together in a composition? The answers to these questions are only possible with a universally acceptable software component model which covers the composition, the syntax as well as the semantics of components. As mentioned earlier the various definition regarding software component model exclude the modeling technique involved. One such definition which takes into account the component modeling by **[Heineman et al., 2001]** is:

"A component is a software elements that conforms to a component model and can be independently deployed and composed without modification according to the composition - standard"

Several Architectural and design description languages (ADL) have been developed for modeling and analyzing software architecture. The examples of such languages in general architecture are: Aesop **[Garlan et al., 1994]** , Rapide **[Luckham et al., 1995]**, Dawin **[Magee et al., 1995]**, C2 **[Medvidovic et al., 1996]**, SADL **[Moriconi et al., 1995]**, UniCon **[Shaw et al., 1995]** . In all these ADLs the (i) notion of a component as a reusable entity with well-defined interfaces, (ii) The interaction mechanism between components through connectors, are common across these ADLs. Beyond these general ADLs for the specification of software architecture there are many component models which are : EJB, COM, .NET **[Wigley et al., 2009]**, JavaBeans **[DeMichiel et al., 2001]**, CCM, Web Services **[Alonso et al., 2010]** , Kola **[Van et al., 2000]** , KobrA **[Atkinson et al., 2000]** , SOFA **[Plasil et al., 1998]** , Aceme-Like ADLs **[Clements, 119]** , UML 2.0 **[Uml OMG, 2004]** , PESCOS **[Genssler et al., 2010]**, Fractal **[The Fractal, 2012]**, **[Bruneton et al., 2002].** The main focus in all of the mentioned component models remains on two main issues:

– ***Component Semantics:*** Component Semantics is the view of the structure of a component. Such a view constitutes the *provided* and *required* services of the component, the

provided and required interfaces. The provided services are the sole functionality of the component which is used by its clients, and the required services are the services which are needed from other components in the system composition, for the smooth operation of a component to provide the requested functionalities/services. Such scenarios where components need the services of other components impose the dependencies among the components. The interface specification of components reveals both the provided and required services of a component. The component semantic should be able to explicitly specify the dependencies between the provided and required services among components. Various software component modeling having same syntax but differ in their semantic. For instance in JavaBeans and EJB are syntactically identical but differ in their semantics **[Lau et al., 2007].** In their semantics the javaBean is a class hosted by a container such as BeanBox, and interacts with others via adapter classes generated by the container. On the other hand EJB is a Java class that is hosted by and managed by EJB container provided by the Java 2 Enterprise Edition (J2EE) Server through the home and remote interface.

– ***Component Syntax:*** Component syntax comes after the component semantic. Component syntax covers the definition which requires a component definition language. The definition language is different from the implementation language. In our case a software component is an architectural unit of functionality, the definition language is an ADL or an ADL-like language such as in Aceme and UML 2.0. In Kola, the definition Language is CDL and the implementation language is C. ADL allows an engineer to reason about system properties at higher level of abstraction. Without a clear specification of ADL the design and architecture of a software results in poor understanding of developers, architectural and design choices are based more on default than solid engineering principles. Also without proper ADLs, the architecture and design cannot be analyzed and evaluated for the quality attributes. There are various ADLs as mentioned in the section above to represent the syntax and semantics of the components. In all of these ADLs there is a considerable difference in their capabilities. There are some core conceptual basis or ontologies **[Garlan, 1995], [Medvidovic et al., 2000]** common through all the ADLs, which are:

- *Component:* A computational element of the system and data store of a system
- *Connectors:* Representation on interaction among components, that provides the communication and coordination among them.

- *System:* Representation of the overall system in a graphical notation which constitutes the components and the connectors.
- *Properties:* Representation of the semantic information of a component and the system. This information includes the extra functional property that goes beyond the structure.
- *Constraints:* Representation of the conditions that should remain true when the design and architecture evolve over time.
- *Styles:* Represents and defines a vocabulary of design elements types and the rules that should be applies in their composition.

4.4.2 Component Composition

In Component Based Software Development (CBD), a single component provides a unit of functionality and it may consume the services provided by other components and produce the output which may be consumed by other components in the system. Such interaction among components results in a system with components interlinked with one another in certain hierarchy which is also known as component composition. Composition is a central issue in Component Based Software Development (CBD). In CBD a component may be composed of several components and the entire system forms a component hierarchy. This notion of component composition allows developers to organize a software system in any desirable hierarchy. Since component provides a mean of reusability, it is fairly possible that an existing component from a component repository be fetched and plugged into the system. In a composition in which verities of components are involved, a composition language is desired. The composition language should have suitable semantics and syntax compatible with the components in the component model **[Lau et al., 2007].** There is no composition language in the most current component model [**Lau et al., 2007].** For instance Kola uses connecter as glue code for composition. Korba and UML 2.0 use the UML notations.

According to **[Crnkovic et.al, 2002],** component composition can take place during different stages of the component life cycle, which are:

- ***Design Phase:*** In this phase the components are to be constructed, designed and defined in source code or possibly compiled into binaries which are then stored into a design repository. The repository should provide the storage and management of components.

- ***Deployment phase:*** As apparent from the name, deployment phase composition, the components are to be compiled to binary codes which intern can be composed and deployed into the target system.
- ***Runtime phase:*** In this phase the components are to be compiled into the binaries, instantiated with data and finally executed in the running system.

Since the component composition depicts the overall structure of the system including interaction, cooperation and coordination. In next coming sections we investigate the component composition of some well-known component modeling approaches.

4.4.3 Various Component Modeling and Composition

Our Security evaluation framework is at the design and architectural level of the software development process. In this section we look at the most common design phase component compositional modeling of some common component models

.

4.4.3.1 Aceme Component Modeling

In Aceme [**Clements, 1996]** a component is functional architecture unit representing a computational element or a data store Interfaces to the components are represented by a set of ports to expose the functionality of the components to its clients. The specified ports of the components can either act as sink or source to receive or send operations respectively. Below figure 4.6 depicts an Aceme component.

Fig. 4.6 Aceme-Like Component

As in any ADL, the composition of components in a system is specified by connectors, that represents the interaction, coordination and cooperation among the components. A connector can mediate the connection between multiple components through the component connector interfaces. Each connector may have multiple interfaces having multiple roles, these roles defines the participant of the interaction. The composition of the components in a system using Aceme component modeling can be viewed as the interconnection of components for both the

provided and required functionalities as shown in figure 4.7. The visual Builder such as AcemeVisual can be used as a tool for such interconnection. In Aceme, the system specification

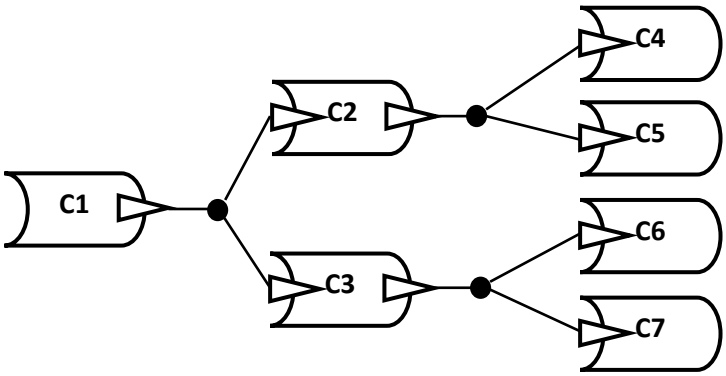

Fig. 4.7 Aceme Component Composition

constructed in the design phase can be compiled to a system in programming language directly **[Lau et al., 2007].** As an example in Aceme the specification of architecture in ArchJava **[Aldrich et Al., 2002]** can be compiled into java provided that the java code for both the code and connectors is available **[Aldrich, et al., 2004]** .

4.4.3.2 Kola Component Modeling

In Kola component modeling **[Van et al., 2000]** a component is a unit of design having a specification and implementation. Syntactically Kola components are defined in ADL-like language which includes an IDL for interface definition, CDL for the definition of component itself and also the DDL for local data specification in component.

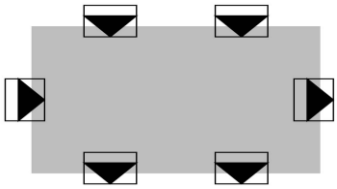

Fig. 4.8 Kola Component

As depicted in the figure 4.8 above, Kola component may have multiple interfaces each specifies the signature of function that it implements. The pointing head of each triangle in the interface represents the direction of the function call.

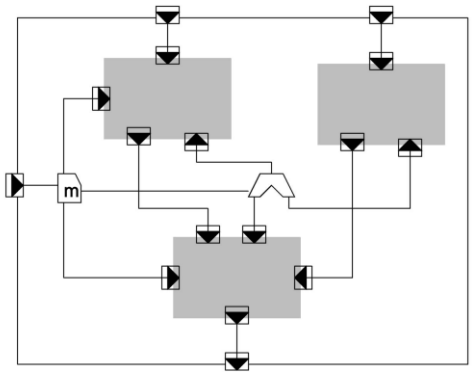

Fig. 4.9 Kola Component Composition

Figure 4.9 above shows the component composition of Kola component modeling. As depicted in the diagram the composition is the interconnection of various components together which externally treated as a single functional unit.

4.4.3.3 UML Component Modeling

The UML (Unified Modeling Language) **[Cheesman et al., 2001]** is best suited for construction, visualization, documentation and specification of the artifacts of software-intensive systems. There are several advantages of adopting UML **[Wu, Ye et al., 2003]**:

- Firstly, UML provider the higher level of information that characterizes the internal behavior of the component. The so provided information can be easily processed and used for the analysis purpose.
- UML as an industry standard provides a variety of software modeling notations and diagrams for many component providers.
- UML provides a set of models for the specification of a model at different levels of capacity and accuracy.

In UML a component is a modular unit of system. It defines the behavior by one or more required and provided interfaces which implement its required and provided services. As shown in figure 4.10. Each required service is depicted as a socket and provided services with a lollipop.

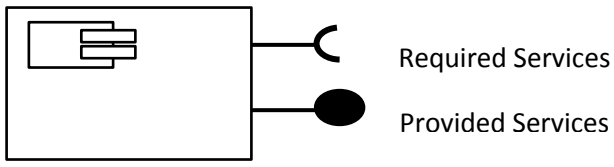

Fig. 4.10 A UML Component

The composition of component in UML 2.0 can be achieved by UML connectors to encompass coordination, cooperation and collaboration among the components in order to provide the desired system functionality. Mainly two types of connectors are there:

- *Assembly Connectors:* is used to provide the required interface of a component to the provided interface of other component (a ball and socket type joint)
- *Delegation Connectors:* represented by an arrow depicts the requested and provided service from inside the environment of composite component to its outer environment.

The dependencies among the components can also be depicted by a dashed line with arrow. Below figure 4.11 shows the component composition of UML component model.

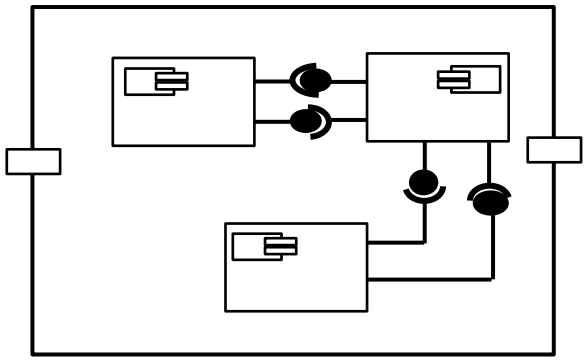

Fig. 4.11 UML Component Composition

In our security evaluation framework next section, we have adopted the UML component modeling notations, because it is widely accepted standard to model and document the software artifacts with different level of capacity, abstraction and accuracy.

4.5 Security Evaluation Framework: Architectural and Design Phase.

In previous sections we have analyzed verities of software architectural & design approaches. From a broader perspective there are three layers of software architecture and design as shown in figure 4.3. For our security evaluation we have chosen the middle layer, the Component Based Software Architecture and Design (CBAD) based on the various advantages of this layer. First it is a manageable layer of software architecture and design, in a sense that it is neither much coarse-grained to hide much of the required details nor too fine-grained that incorporate the much detailed view of software architecture to make the evaluation process too complex.

Secondly there are various component modeling tools available for the specification of a Component Based Architecture and Design (CBAD) of a system. As mentioned in above sections, in component based software engineering there are various modeling approaches each with own merits and demerits. We have selected the UML 2.0 Component Modeling in our security evaluation framework. In chapter 3. Section 1, we have mentioned that, in order to evaluate a system for security we must have to answer three questions (i) *where to measure?* (ii) *What to measure* (ii) and (iii) *How to measure*? Question (i) is about the identification of the appropriate stages in the lifecycle of software where the security evaluation is necessary and should take place in a feasible manner. In chapter 3 figure 3.1 we have analyzed and presented various stages in the software life cycle where security evaluation should play a vital role to promote secure software. Question (ii) is about the identification of the factors that act as a baseline for the security evaluation process and we answered the question in section (4.2) by relating the non-functional security requirements (which are very difficult to be specified precisely and concretely) to the well-established security attributes. The Question (iii) is about a process or a framework, a model or a method that can measure the security of the system efficiently without requiring any special personnel knowledge regarding the measurement. In other words the evaluation method should act as a tool for the development team to measure the security. In our proposed security evaluation framework, we measure software system at the design and architectural level of the software lifecycle. Because the design and architecture of a system act as a blueprint for the overall system, a mistake or a flaw at the design and architectural phase of a system can lead to great penalties and cost in further stages of the system life cycle. In the current networked environment, a system can only be secure if it is isolated in a protected environment operated by a single person i.e. the administrator. But with the advent of fast growing networks, the applications architectures are now shifting from standalone to networked applications, and are being accessed by multiple users over the insecure network simultaneously. Such networked applications results in communication, coordination, cooperation, and resource sharing (both data and other system resources) between the various components of a system. As a result of which the security becomes the more critical aspect of such application architectures. Keeping in view this fact our proposed security evaluation framework is based on the following factors and further based on our framework we propose the

security metrics for the main four attributes of security, the ***confidentiality, integrity*** and ***availability*** and ***dependency***.

1. *Component Composition and Dependencies*
2. *Inter-Component Data/information and resource sharing.*

4.5.1 Component Composition and Dependencies.

A single component seems to be atomic in nature. To provide the services to its client a component normally calls upon the service of other component. Such a scenario where a component calls upon the service of other components which in turn may call upon the service of other components and so forth, inter-linked together in certain order to properly and efficiently provide the required functionality to its clients is known as composition of the system or component composition. With the advent of a networked environment the composition of such components may be either local (bound to local server) or remote (on multiple servers). Similar to the object-oriented systems, in which the object is the basic building block, in CBSs, component is the basic, but usually a black box building block. As new component gets plugged into the composition, it has the effect on that part or whole system. The newly added component in the composition may refer to other components and can be used by others in the composition.

In a composition the components interact, cooperate and coordinate which results in the dependencies among them in order to provide the complex system functionality. At the top there exist two types of dependencies which are:

- *Direct Dependency* : involves a direct association between two components
- *Indirect Dependency*: involves the association between component through intermediate components

According to **[Li, 2003],** there exists at least four types of dependencies between components, *explicit direct dependency*, *explicit indirect dependency*, *implicit direct dependency* and *implicit direct dependencies.* These dependencies show the nature of dependencies which are either direct or indirect and also implicit or explicit. Based upon the functionality and the business logic there are several dependencies that get incurred into the system among the components **[Li, 2003], [Pour, 1998].** These dependencies are:

1) *Data Dependency:* occurs if there is an exchange of data/information. Such dependency occurs if data in one component is used by the other in the composition.

2) *Control Dependency:* occurs if there is control integration among the components. Through component interfaces.
3) *Interface Dependency:* occurs by user-interface integrations. In this dependency when one component required the functionality of other component, it triggers an event through its interface by sending a message. The so triggered event causes another component to respond to the message through its own interfaces. Interface dependency exists as relationship among different functionalities and parameters of software components.
4) *Time Dependency:* Components in compositions are invoked one after the other in a sequence of time, whether the invocation of a component is preceded or followed by the other results in the time dependency among the components.
5) *State Dependency:* when a behavioral change in a component is dependent up on a particular state in the system, it is referred as state dependency.
6) *Cause and Effect Dependency:* if the behavioral change in one component in the system composition results in a change in other component then it is cause and effect dependency.
7) *Input/output Dependency:* occurs when a component provides/requires the information to/from other components.
8) *Context Dependency:* represents that a component run must be under special context environment.

A system composed of several components (functional units) can be represented in graphical form. In our case we use UML based component modeling, because of its various advantages over the other modeling techniques, as mentioned earlier. In order to simplify the process we take into account the system composed of multiple component for illustrative purpose as shown in figure 4.12 below using Visual paradigm for UML 9.0. As depicted in diagram (4.12) there exists dependencies that occurs due to the interconnection of provided and required interface (includes both data and interface dependencies) for the provided and required functionalities. Beside these implicit dependencies, the dependencies are also specified explicitly by dashed arrow lines. The direction of the line depicts the source component on the tail of the line which depends up on the destination component on the arrow head.

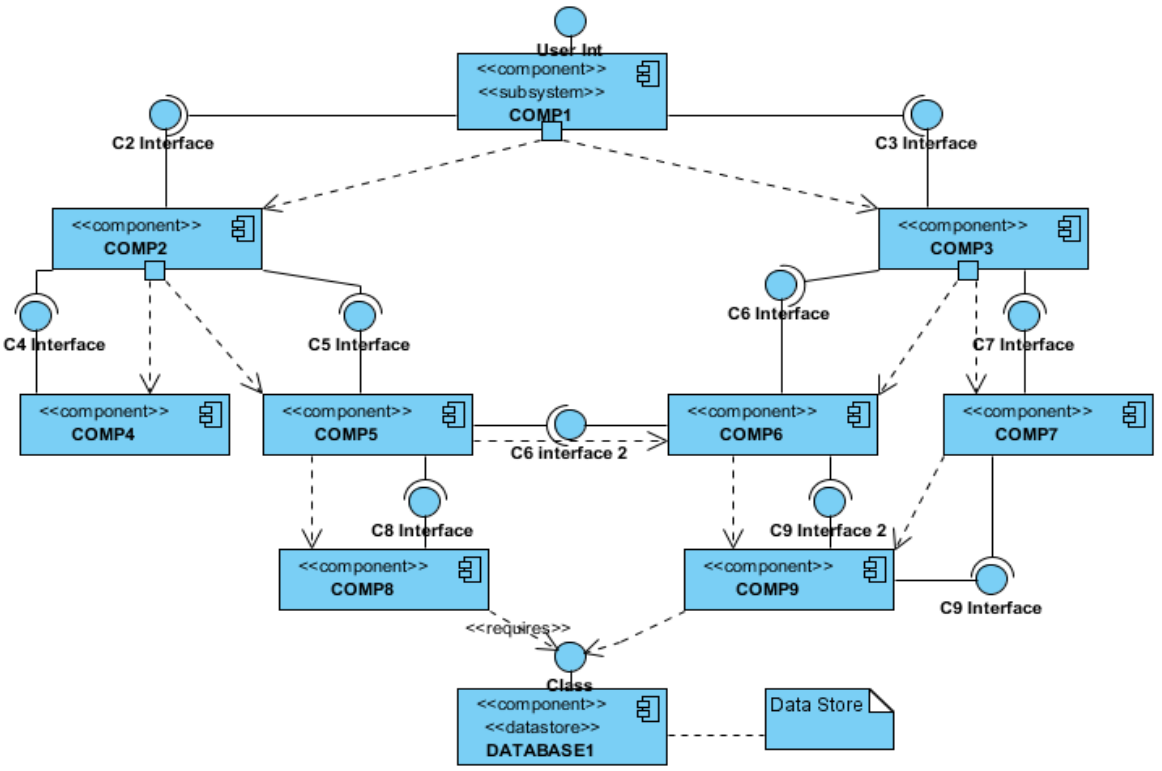

Fig. 4.12 An Illustration of a System's Component Composition and Dependencies

We depict the dependencies among the components into an adjacency matrix (AM) representation **[Li, 2003],** The Components are organized as rows and columns with index i *and* j respectively. If a component C_i in row i is dependent on other component C_j in column j then the corresponding element in the adjacency matrix (AM) is marked as1, otherwise it is 0. In general the values for each of the element of adjacency matrix $DM_{(n*n)} = dij_{(n*n)}$. Where:

$$dij = \begin{cases} 1, & if\ ci \rightarrow cj \\ 0, & otherwise \end{cases} \quad ------ \quad (4.1)$$

From The above equation (4.1) the dependency matrix (DM) for the system composition above figure 4.10 is:

$$DM = \begin{array}{c} \\ c1 \\ c2 \\ c3 \\ c4 \\ c5 \\ c6 \\ c7 \\ c8 \\ c9 \\ c10 \end{array} \begin{array}{c} \begin{array}{cccccccccc} c1 & c2 & c3 & c4 & c5 & c6 & c7 & c8 & c9 & c10 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Fig 4.13 Direct Dependencies Matrix

The matrix above figure (4.13) is a direct dependency matrix i.e. it depicts the direct association among the components in the composition. Further in order to calculate all the indirect possible dependencies among the components (direct as well as indirect) we apply the Warshall's algorithm of transitive closer **[Rosen, 1994]** to compute the Full Dependency matrix of the illustrative system composition in below figure (4.12). The Full Dependency matrix in above figure (4.14) shows all possible dependencies (direct as well as indirect dependencies of each of the component of figure (4.12).

$$FDM = \begin{array}{c} \\ c1 \\ c2 \\ c3 \\ c4 \\ c5 \\ c6 \\ c7 \\ c8 \\ c9 \\ c10 \end{array} \begin{array}{c} \begin{array}{cccccccccc} c1 & c2 & c3 & c4 & c5 & c6 & c7 & c8 & c9 & c10 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Figure 4.14 Full Dependencies Matrix

Beside the dependencies presented earlier we define two more types of dependencies associated with a component in the system composition, the ***In-Dependency*** and ***Out-Dependency***.

- ***In-Dependency:*** In-Dependency of a component (C_i) is defined as the other components in the composition that are directly or indirectly dependent up on the component(C_i).
- ***Out-Dependency:*** Out-Dependency of a component (C_i) is defined as the other components in the composition up on which component (C_i) depends (directly or indirectly) for their provided functionalities.

We further define two more terms related to the *In-Dependency* and *Out-dependency* of a component which are:

- ***Degree-In***: denoted by $\boldsymbol{DegIN(C_j)}$ is defined as the number of component in the *In-Dependency* of the component (C_j) . Degree-In of a Component can be easily calculated by counting the number of $1's$ in the corresponding column j of the Full Dependency Matrix(FDM).
- ***Degree-Out:*** denoted by $\boldsymbol{DegOUT(C_i)}$ of component (C_i) is defined as the number of other components in the system composition upon which the (C_i) is dependent. Degree-Out of a component can be easily calculated by counting the number of $1's$ in the corresponding row i of the Full Dependency Matrix (FDM).

  Mathematically:

$$degIN(C_i) = \sum_{j=1}^{n} (FBD_{ji}) - - - - - - - - (4.2)$$

  And

$$degOUT(C_i) = \sum_{j=1}^{n} (FBD_{ij}) - - - - - - - - (4.3)$$

Based upon the dependencies (direct as well as indirect) we derive and propose the security matric for the ***Dependency*** and A***vailability*** for each of the component and then for overall system.

4.5.1.1 Dependency Metric Derivation

Dependency is one of the major attribute of security that must be analyzed. If a component is an isolated entity, i.e. neither it depends upon any other component nor any other component is dependent on it, then there is no problem at all. But in reality components interact coordinate and cooperate to provide the complex functionalities to its clients. The level of dependency of each

of the component in a system composition is necessary to be evaluated, because such dependency level provide us the indicators about the criticality of the component and its effect on the overall system if gets compromised.

As stated earlier a component exhibit two types of dependencies **In-Dependency** and **Out-Dependency,** equation (4.2) and equation (4.3) respectively. In order to calculate the overall dependency (direct and indirect) we derive the total dependency metric of a component $\boldsymbol{C_i}$ as:

$$\boldsymbol{Dep(C_i) = \frac{1}{degOUT(C_i) + degOUT(C_i)}} \quad -----(\mathbf{4.4})$$

Where

$degOUT(C_i)$ is the *Out-Degree* of component C_i

$degOUT(C_i)$ is the *In-Degree* of component C_i and $degOUT(C_i), DegOUT(C_i) > 0$.

Note that the resultant values are in a controlled scale between (0-1). Lower the resultant values higher will be the effect of the component.

The *dependency* metric for the overall system $\boldsymbol{Dep(Sys)}$ can be derived as:

$$\boldsymbol{Dep(Sys) = \frac{\sum_{i=1}^{N} Dep(Ci)}{N}} \quad ------(\mathbf{4.5})$$

Where N is the total number of component in the system.

4.5.1.2 Availability Metric Derivation

Availability ensures that the service provided by software, its component or a network should remain available to its clients in timely manner. In the current networked environment the software architectures are now shifting from simple standalone application to large distributed architectures based on OSI or J2EE n-tiers. In case where the composition of components is remote, the provided functionalities of the components are accessed by remote procedure call (RPC) which requires the invocation, marshalling and unmarshallingg of the parameters. In a functional dependency graph of a system, one component may invoke a call and wait for the provided services of second component which in turn requires the functionality of third component and so forth. Such a chain of functional dependencies among components will

certainly results in delayed response time at each hop of the dependency chain. The delay involved at each hop of the dependency chain is mainly due to the two major factors which are:

- ***Processing Delay:*** Time taken by a component to successfully process the request of its clients and return the result from the invocation to the end result returned.
- ***Transmission Delay:*** In case the composition of components is remote then the transmission delay is the transmission time alone over the network excluding the processing delay.

The delayed response at each of the component can certainly affect the *availability* (one of the main attribute of security) of the system. Software developers must need to know preferably quantitatively the possible delay incurred at each of the components in the system composition and the aggregated level of availability for the overall system.

As mentioned previously a component can act as a hub in which it handles the request from one group of components for the required functionality and may in turn call up on the provided services of other group of components on their behalf .This process can form a chain which results in a system with a delayed response time especially in distributed and multiuser environment, which in turn affect availability of the system. Software developers must be able to identify the possible level of risk involved with each of the component in the system composition. The measurement of availability especially results in to find the availability critical components, so that the alternative corrective measures can be applied. In this section we derive the availability metric for a component (C_i) by taking into account the following factors.

- $DegIN(C_j)$.
- $DegOUT(C_i)$.
- Processing Delay P.
- Transmission Delay T.

For a component C_i the possible maximum number of components that may call up on the services of (C_i) =$DegIN(C_j)$.

The maximum number of components that C_i can call for their provided services =$DegOUT(C_i)$.

As the dependency level of a component (both in and out) increases, it will certainly affect the *availability* of the component and the overall system, because of the processing delay P and transmission delay T of each of the component in the dependency graph.

We put forward the above theoretical concept of availability into mathematical form. There may be a $1-N$ relationship between each of the component in $\boldsymbol{DegIN(C_i)}$ and $\boldsymbol{DegOUT(C_i)}$, i.e. for each of the component in $\boldsymbol{DegIN(C_i)}$, $\boldsymbol{C_i}$ may call some or all of the components for required services on the of behalf of invoking components . Based on these factors we propose the availability matrix of a component $\boldsymbol{C_i}$ denoted by $\boldsymbol{A(C_i)}$ as:

$$\boldsymbol{A(Ci)} = \frac{\mathbf{1}}{\boldsymbol{DegIN(Ci)} + \sum_{\boldsymbol{j=1}}^{\boldsymbol{DegIN(Ci)}} \boldsymbol{DegOUT(Ci)} + \sum_{\boldsymbol{j=0}}^{\boldsymbol{DegOUT(Ci)}} \boldsymbol{Pj} + \boldsymbol{Tj}} \quad ----(\mathbf{4.6})$$

Where

$DegIN(Ci), DegOUT(C_i) > 0$

$DegIN(C_i)$ is the in-degree of component $\boldsymbol{C_i}$.

$DegOUT(C_i)$ is the out-degree of component $\boldsymbol{C_i}$.

P_j is the processing delay of the j^{th} component in $DegOUT(C_i)$.

T_j is the transmission delay involved of j^{th} component in $DegOUT(C_i)$

Note that the index of j starts from 0 to take into account the processing and transmission delay of C_i itself.

The proposed metric in equation (4.5) takes into account the components that can invoke $\boldsymbol{C_i}$ which in turn adds the component in the $DegOUT(C_i)$ by the $DegIN(C_i)$ number of times , because of the fact that for each of the component dependent on the provided services of C_i, C_i in turn may possibly call some are all the components in its Out-Dependency i.e. in $DegOUT(C_i)$. Further the proposed metric takes into account the transmission delay T and processing delay P for each of the component which C_i possibly call for their required services including the delay of C_i.

Above equation (4) can be simplified as:

$$\boldsymbol{A(Ci)} = \frac{\mathbf{1}}{\boldsymbol{n} + \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{m} + \boldsymbol{T} + \sum_{\boldsymbol{j=1}}^{\boldsymbol{m}} \boldsymbol{Pj}} \quad -------(\mathbf{4.7})$$

Where

n is $DegIN(C_i)$ of component C_i

m is $DegOUT(C_i)$ of component C_i

T is the total transmission delay of components the component C_i and the transmission delay of the of the all component in the $DegOUT(C_i)$ invoked by C_i.

The proposed *availability* metric provides the software developer the early indicator about the level of availability of each of the component and can easily identify the *availability critical* component of the system and enable to provide the necessary corrective or preventive measures accordingly. The range of output values of the above availability matric in equation (4 and 5) will remain in a $(0,1)$. More the value tends towards 0 on the scale higher the effect on the availability of the component. The aggregated *availability* of the overall system denoted $\boldsymbol{A(Sys)}$ can be computed as:

$$\boldsymbol{A(Sys) = \frac{\sum_{i=1}^{N} A(Ci)}{N}} \quad ------(\mathbf{4.8})$$

Where:

$A(Ci)$ is the level of the availability of the individual component in the system composition

N is the total number of components in the system.

The so proposed *availability* metrics is for the design and architectural phase of the software development lifecycle but it can also be applicable to the already developed systems by performing reverse engineering to get to the design and architecture of the system. The proposed metric of *availability* serve as a prediction about the criticality of the components in the system composition.

4.5.2 Inter-Component Data/Information Flow and Resource Sharing.

The dependencies among the component (data dependency, functional dependency, interface dependency, etc.) results in the flow of data/information across the components. Such a flow of data/information must be analyzed for the secure system operations. In this and the next subsequent section we look at the data/ information flow among the components and based on the dependencies and the flow of data/information, we derive and propose the security metrics for two fundamental attributes of the security, the ***confidentiality*** and ***integrity*** for each of the component and in the composition and then for the overall system.

As depicted in figure (4.5) the implementation of the component is isolated from its interface specification which provides flexibility in the implementation of the component using different

implementation or programming languages. These interfaces can be defined as the component access point **[Crnkovic, et al., 2002] .**The access points enable the access to the provided functionality of the components for its clients. A component normally has multiple access points for the different functions provided in the interface **[Szyperski et al., 2002].** As mentioned earlier the *coordination*, *cooperation* and *interaction* among components results in various types of dependencies among the components. Such dependencies in turn results in the exchange of data/information (both in and out flow) across the components. Since in component Based Software Development (CBD) each component is a separate entity designed with varying level of protection then plugged together to provide the overall system functionality. It becomes the most critical to analyze the flow of data/information and resource sharing among the components. The main aim of our evaluation process is to provide a prediction about the security and accordingly provide the security indicators of each of the component in order to point out the most critical component in the composition and then further the security indicators of overall system.

The flow of data/information is characterized by two types of flows, Inter-component flow and Intra-component flow **[Abdellatief et al., 2011].**

- *Inter-Component Flow:* The exchange of data/information (both *Out-flow* and *In-flow*) across the components takes place through provided and required interfaces. A component provides information to its clients (a user, a required interface or an engineering device) by an *Out-flow* through its provided interfaces through a setof of *Out-parameters* and receives data/information from the client by an *In-flow* through its required interfaces by a set of *In-parameters*. Since only the interfaces are visible to the clients of a component, such an exchange of data/information takes place across the components interface boundary and is known as *Inter-Component Information Flow.*
- *Intra-Component Flow:* As a result of *Inter-Component* data/information flow, provided and required interfaces of a component pass /receive the data to/from component body. For instance, in case of an *In-flow* in Inter-component data/information flow the required interfaces of the component pass the data/information to the component body for further processing or to update a backend data source or a data structure. The *In-flow* and *Out-flow* across the component interfaces in the *Inter-component data/information flow* gets mapped to *Write-flow* and *read-flow* with in the component body respectively. So one

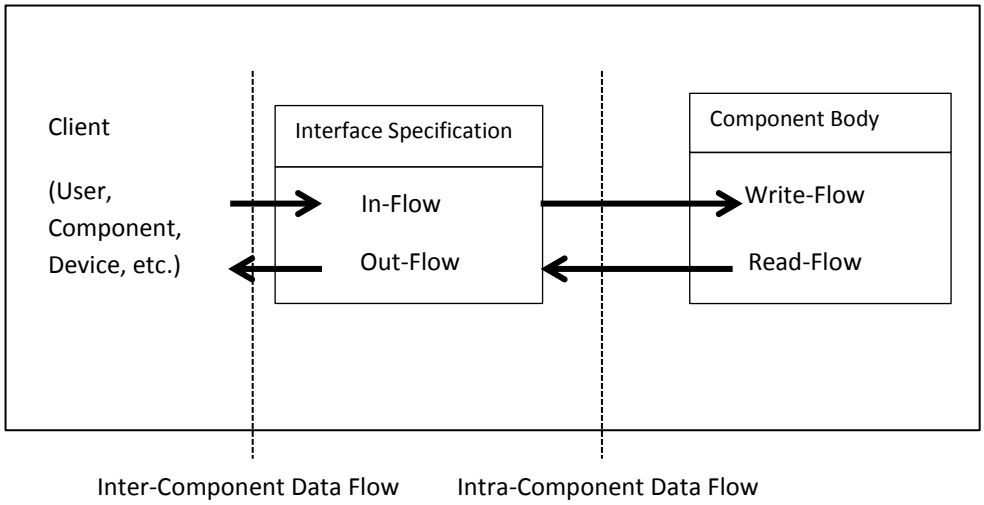

Figure 4.15 Component Information Flow

can say the *In-flow* as *Write-flow* and an *Out-flow* as *Read-flow*. In this thesis we use these terms interchangeably. As depicted in figure (4.14), In case of *Intra-component information* and data/flow boundary is confined to the component body. Above figure (4.15) shows the data/information flow of a component.

Like dependencies, the data/information flow among the components can be either *direct-flow* or *indirect-flow.*

- *Direct Data/Information-flow:* In the direct information flow the flow of information (to/from) between components occur directly without passing through any intermediate node or component. For instance if a component C_1 invokes other component C_2 and passes the data through a set of parameters to it or C_2 returns something back to C_1 is known as direct data/information flow
- *Indirect Data/Information-flow:* In the indirect flow of data/information the flow of data/information takes place indirectly through intermediate nodes or components. As an example if a component C_1 invokes component C_2 which processes and in turn invokes C_3 and passes the data/information passed by C_1 to C_3 or C_3 returns something to C_1 via C_2 is known as indirect-data/information flow.

Software development team must need a concrete way to analyze the flow of data/information (both direct and indirect) across the components in a system. UML component modeling provides a variety of tools and notations for the specification of components, components

interaction, and interface specifications in order to design a system. The so design of the system using UML modeling, the flow of data / information can easily be analyzed.

Each component in the system composition possesses certain resource such as database access, files, data/ information. Components in the composition can access the resources of other components with certain privileges. As examples say a component C in a composition has the access to a database. Other components can invoke C along with certain parameters to perform the operation on the backend database. Such a scenario is most common in central database applications.

As the design and architecture of a system evolve it becomes complex and cumbersome to keep track of the flow of data/information across the components. Such a scenario results in security threat to the system especially the two main concerns of the security of any system, the ***confidentiality*** and ***integrity***. A security flaw at one of the system component, if compromised can result in the adverse effect on the functioning of overall system. Software developers must be able to predict and assess the possible criticality level of each of the component and then of the overall system. In the next subsequent sections we propose the metrics for the ***confidentiality*** and ***integrity*** of a component and then for the overall system based on the following parameters.

- Component dependencies.
- Data/Information flow.
- Component interfaces.

4.5.2.1 Confidentiality Metric Derivation

Confidentiality of any system, resource or a network ensures that unauthorized disclosure of data/information should not occur. In the large complex software systems (comprised of multiple components both local and remote) the flow of data/information across the components becomes a critical factor for the security of the overall system. Even if a system built with the protection mechanisms in place, the data/information breach at one of the component can cause the serious problem to the overall system. Our proposed metric aim to asses, predict and provide the quantitative indicators about the level of *confidentiality* of each of the component and further for the overall system.

As mentioned earlier the inter-component data/information flow takes placed through the provided and required interfaces of a component. For the simplicity we redefine these two types of interfaces as:

- *Write-Interfaces:* denoted by I_w , are the required interfaces of a component through which the *In-flow* of data/information takes place.
- *Read-Interfaces:* denoted by I_r , are the provided interfaces through which the *Out-flow* of information takes places.

The confidentiality is likely to be affected through a component when both the read and write operations by different components in the composition takes place. The idea is to identify the level of such operations on a component and accordingly quantify the resultant indicators. With respect to our confidentiality metric we put forward the following argument.

- *Argument 4.1:* The confidentiality of a component is likely to be affected more as the number of reading components in the system composition and dependency (direct/indirect) increases with respect to the writing components. For instance a component C having n number of reading components (the components that are functionally dependent on the provided services of C) and m number of writing components (the components in the system composition whose services are required by C). As the n increases for each of the component in m the confidentiality of the component gets affected more, i.e. each of component in n is likely to have read access to component C and all the components in the dependency (direct/indirect) that are capable of writing to C .

The above specified argument doesn't mean that that the effect of writing components m is completely undesirable. If there is no writing component the confidentiality will not be affected at all, instead the argument states that an increase in n (reading components) has relatively higher impact on the confidentiality than m (writing components). Also as specified earlier a component may have multiple interfaces (both provided and required) which we defined earlier as *Write-Interface* (I_w) and *Read-Interface*(I_r) through which the *in-flow* and *out-flow* of data/information takes place. Because these interfaces are the ports of a component for the flow of data/information across the component, so the number of these ports is also the candidate for the overall confidentiality level of a component.

In order to derive the *confidentiality* metric for a component $\boldsymbol{C}$ we have following parameters at hand.

- From equation (4.3), number of components that can likely make an *In-flow* (write operation) directly or indirectly on C is$DegOUT(C_i)$.
- From above equation (4.2), number of components that can likely responsible for the *Out-flow* (read operation) of data/information directly or indirectly from $\boldsymbol{C}$ is $DegIN(C)$.
- Possible number of read-interfaces (provided interfaces) through which the *Out-flow* (read operation) can take place on C is I_r.
- Possible number of write-interface (required interfaces) through which the *In-flow* (write operation) can takes place on C is I_w.

Keeping in view the above argument (1), that in both the read and write operation the confidentiality will be affected more as the number of reading components ($DegIN(C)$) increases for each of the writing component ($DegOUT(C)$) the confidentiality metric is :

$$\boldsymbol{COD(C_i) = \frac{1}{(DegIN(C_i)^2 * I_r)/\, DegOUT(C_i)\ + I_w}} \quad -----(\mathbf{4.9})$$

Where

$DegIN(C_i), DegOUT(C_i), I_r, I_w > 0$

$COD(C_i)$ is the confidentiality of component$\boldsymbol{C_i}$.

$DegIN(C_i)$ is the degree of in-dependency (number of components that directly or indirectly read from) component $\boldsymbol{C_i}$.

$DegOUT(C_i)$ is the degree of out-dependency (number of components that directly or indirectly write to) component C_i.

$\boldsymbol{I_r}$ and $\boldsymbol{I_w}$ are the read and write (provided and required) interfaces of component $\boldsymbol{C_i}$.

The above equation (7) can be simplified as:

$$\boldsymbol{COD(C_i) = \frac{1}{(n^2 * I_r)/\, m\ + I_w}} \quad -----(\mathbf{4.10})$$

Where

n is the $DegIN(C_i)$

m is the $DegOUT(C_i)$

The motivation behind the derived metric is to predict the level of confidentiality of each of the component and based on the output values, the software development team will be able to take the required action. The above derived confidentiality metric, equations (4.9, & 4.10) provides the quantative indicator about the criticality of each of the component with respect to the confidentiality. In other words it acts as a tool for the development team to identify the most critical component and enables them to take the necessary action if needed. As stated earlier the numbers of reading components have the higher impact on the confidentiality of the component with respect to the writing components. In the proposed confidentiality metric such a scenario is taken into the consideration by squaring the $DegIN(C_i)$.

The aggregated confidentiality metric for of the overall system is

$$\boldsymbol{COD(S) = \frac{\sum_{i=1}^{N} COD(Ci)}{N}} \text{ - - - - - -} (\mathbf{4.11})$$

Where

$\boldsymbol{N}$ is the total number of components in the composition.

4.5.2.2 Integrity Metric Derivation.

Integrity is the third main pillar of the security of any software, network, or any other system. The main objective of the *integrity* is to ensure that unauthorized modification to the data/information and information processing resource should not take place. As with *confidentiality*, as the architecture and design of the system evolves from the scratch to the full-fledged design and architecture, it becomes very difficult to keep track of the information flow and various write operations taking place in the in the system. System developers need some tools that can be applied at the early architecture & design and afterwards to to identify the most critical elements of the system and preferably the quantative indicators of the level of availability of each of the component and for the whole system, in order to make necessary decisions and adjustment early to reduce the cost and efforts needed in further system life cycle. The integrity of a component C is likely to be affected when multiple component in the system composition perform a write operation (*In-flow* of data/information) on C though the multiple *write-Interfaces (I_w) (Required interfaces) of* a component. For instance a component C may having certain resources say a backend database or a file, other components can call upon the services of C and

pass the data through the *Write-interfaces* (I_w) to update the data/base and file. A weaker component in the system composition can be attacked to violate the integrity of the overall system. Both software developer and the user must need to know the potential risk associated with each of the component. Taking into account all the parameters we propose the *integrity* metric of a component as:

Let $\boldsymbol{C}$ be a component whose integrity level is to be evaluated.

The possible number of components in the system composition that are responsible for the *In-flow* (write) of data/information to $\boldsymbol{C}$ would be in the *Degree-Out* of . As mentioned earlier degree-out of a component $\boldsymbol{C}$ is the number of components in the system composition on which $\boldsymbol{C}$ depends for their provided services. These components are bound to $\boldsymbol{C}$ through its Write-Interfaces (I_w) or simply the required interfaces. Mathematically if $\boldsymbol{m}$ be the number of components that are responsible for the *In-flow* data/information then:

$$\boldsymbol{m} \in \boldsymbol{DegOUT(C)} ----- \quad (\mathbf{4.12})$$

From the above equation (4.12), $DegOUT(C)$ represents all the possible components on which $\boldsymbol{C}$ depends (directly or indirectly) for the provided services and that can perform a write-operation on $\boldsymbol{C}$ directly or indirectly.

Also the number of ports or *Read-Interfaces* through which the *In-flow* of data and information can take place is $\boldsymbol{I_w}$. The complete integrity metric for a component $\boldsymbol{C_i}$ is:

$$\boldsymbol{INT(Ci)} = \frac{\mathbf{1}}{\boldsymbol{Iw} * \boldsymbol{degOUT(Ci)}} ------(\mathbf{4.13})$$

Where

I_w are the write-Interfaces of component C_i .

$DegOUT(C_i)$ is the out-dependency of C_i .

The so proposed metric (equation (4.13)) provides the early indicator of the level of integrity of each of the system. As with the earlier equations, the resultant value lie in between (0,1) with the lower values on the scale the higher chance of integrity breach.

The aggregated *integrity* metric for the overall system $\boldsymbol{S}$ is

$$\boldsymbol{INT(S) = \frac{\sum_{i=1}^{N} INT(Ci)}{N}} \quad \boldsymbol{------(4.14)}$$

Where $\boldsymbol{N}$ is the total number of components in the system composition.

Where

$\boldsymbol{N}$ is the total number of components in the system composition

$\boldsymbol{I_w}$ are the write-Interfaces of component $\boldsymbol{C_i}$.

4.6 Algorithmic Specification

Algorithm is a way of presenting the solution of a problem in discrete, unambiguous and finite set of steps that can be easy to apply and implement. In this section we present our security evaluation framework and derived metrics into an applicable algorithmic form with respect to four predefined security attributes, ***dependency***, ***Availability***, ***Confidentiality*** and ***Integrity***.

4.6.1 Algorithm 4.1: (*Dependency Evaluation*).

Step 1: Input System Architecture and Design

(i) Using UML Modeling Specify the system (such as using Visual paradigm for UML 9.0) as in above diagram (4.11)

(ii) Specify the dependencies explicitly using dashed arrow lines beside the implicit interface dependencies.

Step 2: Specify The System into an Adjacency Matrix Representation $\boldsymbol{AM_{(n*n)}}$.

(i) **If** there is dependency between component $\boldsymbol{C_i \rightarrow C_j}$ **then**

$$\boldsymbol{AM_{(i,j)} = 1}$$

Else

$$\boldsymbol{AM_{(i,j)} = 0}$$

End If

Step 3: Compute and Calculate the Full Dependency Matrix from the Adjacency Matrix $\boldsymbol{AM_{(n*n)}}$

(i) Using Warshall's Algorithm for transitive closure compute all the direct and indirect dependencies into a Full Dependency Matrix ***(FDM).***

Step 4: Compute the ***In-Degree*** and ***Out-Degree*** of each of the component in the composition.

(i) Computer the ***In-Degree*** of a component as $\boldsymbol{degIN(C_j)}$ from the corresponding column j of ***(FDM)*** as:

$$\boldsymbol{degIN(C_j)} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{FDM(i,j)}$$

(ii) Compute the Out-Degree of a component as $\boldsymbol{degIN(C_i)}$ from the corresponding row i of ***(FDM)*** as:

$$\boldsymbol{degIN(C_i)} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{n}} \boldsymbol{FDM(i,j)}$$

Step 5: Compute the total dependency metric $\boldsymbol{degTOT(C)}$ of a component $\boldsymbol{C}$ on a scale of $(\boldsymbol{0-1})$ as:

(i)

$$\boldsymbol{degToT(Ci)} = \frac{\boldsymbol{1}}{\boldsymbol{degIN(C_1) + degOUT(C_i)}}$$

Step 6: Quantify the level of *Dependency* for the overall System $\boldsymbol{DEP(Sys)}$ having $\boldsymbol{N}$ number of total components in the system composition as:

(i) $$\boldsymbol{DEP(Sys)} = \frac{\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{degTOT(Ci)}}{\boldsymbol{N}}$$

Step 7: Stop

Step 8: End.

4.6.2 Algorithm 4.2: (*Availability Evaluation*)

Step 1: Input System Architecture and Design

(i) Using UML Modeling Specify the system (such as using Visual paradigm for UML 9.0) as in above diagram (4.11)

(ii) Specify the dependencies explicitly using dashed arrow lines beside the implicit interface dependencies.

Step 2: Specify The System into an Adjacency Matrix Representation $\boldsymbol{AM_{(n*n)}}$.

(i) **If** there is dependency between component $\boldsymbol{C_i \rightarrow C_j}$ **then**

$$\boldsymbol{AM_{(i,j)} = 1}$$

Else

$$\boldsymbol{AM_{(i,j)} = 0}$$

End If

Step 3: Compute and Calculate the Full Dependency Matrix from the Adjacency Matrix $\boldsymbol{AM_{(n*n)}}$

(i) Using Warshall's Algorithm for transitive closure compute all the direct and indirect dependencies into a Full Dependency Matrix ***(FDM).***

Step 4: Compute the ***In-Degree*** and ***Out-Degree*** of each of the component in the composition.

(i) Computer the ***In-Degree*** of a component as $\boldsymbol{degIN(C_j)}$ from the corresponding column j of ***(FDM)*** as:

$$\boldsymbol{degIN(C_j) = \sum_{i=1}^{n} FDM(i,j)}$$

(ii) Compute the Out-Degree of a component as $\boldsymbol{degOUT(C_i)}$ from the corresponding row i of ***(FDM)*** as:

$$\boldsymbol{degOUT(C_i) = \sum_{j=1}^{n} FDM(i,j)}$$

Step 5: To quantify the *Availability* level of a component $\boldsymbol{C_i}$ Compute the *Processing* and *Transmission* delay for each of the component in $\boldsymbol{degOUT(C_i)}$ as along with the *Processing* and *Transmission* delay of $\boldsymbol{C_i}$:

(i) $$\boldsymbol{Delay = \sum_{i=0}^{degOUT(C_i} P_{i+} T_i}$$

Step 6: Quantify the *Availability* level of each of the component $\boldsymbol{A(C_i)}$ within the range $(\boldsymbol{(0-1)}$ by *Availability* metric as:

(i) $$\boldsymbol{A(C_i) = \frac{1}{DegIN(Ci)+\sum_{j=1}^{DegIN(Ci)} DegOUT(Ci)+Delay}}$$

Step 7: Quantify the level of *Availability* for the overall System $\boldsymbol{A(Sys)}$ having $\boldsymbol{N}$ number of total components in the system composition as:

(ii) $$\boldsymbol{A(Sys) = \frac{\sum_{i=1}^{N} A(Ci)}{N}}$$

Step 8: Stop.

Step 9: End.

4.6.3 Algorithm 4.3: (*Confidentiality Evaluation*)

Step 1: Input System Architecture and Design

(iii) Using UML Modeling Specify the system (such as using Visual paradigm for UML 9.0) as in above diagram (4.11)

(iv) Specify the dependencies explicitly using dashed arrow lines beside the implicit interface dependencies.

Step 2: Specify The System into an Adjacency Matrix Representation $\boldsymbol{AM_{(n*n)}}$.

(ii) **If** there is dependency between component $\boldsymbol{C_i \rightarrow C_j}$ **then**

$$\boldsymbol{AM_{(i,j)} = 1}$$

Else

$$\boldsymbol{AM_{(i,j)} = 0}$$

End If

Step 3: Compute and Calculate the Full Dependency Matrix from the Adjacency Matrix $\boldsymbol{AM_{(n*n)}}$

(i) Using Warshall's Algorithm for transitive closure compute all the direct and indirect dependencies into a Full Dependency Matrix ***(FDM).***

Step 4: Compute the ***In-Degree*** and ***Out-Degree*** of each of the component in the composition.

(i) Computer the ***In-Degree*** of a component as $\boldsymbol{degIN(C_j)}$ from the corresponding column j of ***(FDM)*** as:

$$\boldsymbol{degIN}(\boldsymbol{C_j}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{FDM}(\boldsymbol{i},\boldsymbol{j})$$

(ii) Compute the Out-Degree of a component as $\boldsymbol{degOUT}(\boldsymbol{C_i})$ from the corresponding row *i* of ***(FDM)*** as:

$$\boldsymbol{degOUT}(\boldsymbol{C_i}) = \sum_{\boldsymbol{j=1}}^{\boldsymbol{n}} \boldsymbol{FDM}(\boldsymbol{i},\boldsymbol{j})$$

Step 5: To quantify the *Availability* level of a component $\boldsymbol{C_i}$:

(i) Quantify the possible number of components that can read from $\boldsymbol{C_i}$ as $\boldsymbol{Cr}$:

$$\boldsymbol{Cr} = \boldsymbol{degIN}(\boldsymbol{C_j})$$

(ii) Quantify the possible number of components that can write to $\boldsymbol{C_i}$ as $\boldsymbol{Cw}$:

$$\boldsymbol{Cw} = \boldsymbol{degOUT}(\boldsymbol{C_j})$$

(iii) Count and quantify the Reading $\boldsymbol{I_r}$ and writing $\boldsymbol{I_w}$ interfaces of $\boldsymbol{C_i}$ as:

$$\boldsymbol{I_r} = \boldsymbol{No.of\ possible\ Reading\ Intefaces\ of\ C_i}$$

$$\boldsymbol{I_w} = \boldsymbol{No.of\ possible\ Writing\ Intefaces\ of\ C_i}$$

Step 6: Quantify the *Confidentiality* level of each of the component $\boldsymbol{COD}(\boldsymbol{C_i})$ within the range ($\boldsymbol{0-1}$) by *Availability* metric as:

(i) $$\boldsymbol{COD}\ (\boldsymbol{C_i}) = \frac{\boldsymbol{1}}{(\boldsymbol{Cr})^{\boldsymbol{2}} * \boldsymbol{I_r})/\ \boldsymbol{Cw} + \boldsymbol{I_w}}$$

Step 7: Quantify the level of *Availability* for the overall System $\boldsymbol{A}(\boldsymbol{Sys})$ having $\boldsymbol{N}$ number of total components in the system composition as:

(i) $$\boldsymbol{COD}(\boldsymbol{Sys}) = \frac{\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{COD}(\boldsymbol{Ci})}{\boldsymbol{N}}$$

Step 8: Stop.

Step 9: End.

4.6.4 Algorithm 4.4: (*Integrity Evaluation*)

Step 1: Input System Architecture and Design

(i) Using UML Modeling Specify the system (such as using Visual paradigm for UML 9.0) as in above diagram (4.11)

(ii) Specify the dependencies explicitly using dashed arrow lines beside the implicit interface dependencies.

Step 2: Specify The System into an Adjacency Matrix Representation $\boldsymbol{AM_{(n*n)}}$.

(i) **If** there is dependency between component $\boldsymbol{C_i \rightarrow C_j}$ **then**

$$\boldsymbol{AM_{(i,j)} = 1}$$

Else

$$\boldsymbol{AM_{(i,j)} = 0}$$

End If

Step 3: Compute and Calculate the Full Dependency Matrix from the Adjacency Matrix $\boldsymbol{AM_{(n*n)}}$

(i) Using Warshall's Algorithm for transitive closure compute all the direct and indirect dependencies into a Full Dependency Matrix ***(FDM).***

Step 4: Compute the ***In-Degree*** and ***Out-Degree*** of each of the component in the composition.

(i) Computer the ***In-Degree*** of a component as $\boldsymbol{degIN(C_j)}$ from the corresponding column j of ***(FDM)*** as:

$$\boldsymbol{degIN(C_j) = \sum_{i=1}^{n} FDM(i,j)}$$

(ii) Compute the Out-Degree of a component as $\boldsymbol{degOUT(C_i)}$ from the corresponding row i of ***(FDM)*** as:

$$\boldsymbol{degOUT(C_i) = \sum_{j=1}^{n} FDM(i,j)}$$

Step 5: To quantify the *Integrity* level of a component $\boldsymbol{C_i}$:

(i) Quantify the possible number of components that can perform write operation on $\boldsymbol{C_i}$ as $\boldsymbol{Cw}$:

$$\boldsymbol{Cw = degOUT(C_i)}$$

(ii) Count and quantify the Reading $\boldsymbol{I_r}$ and writing $\boldsymbol{I_w}$ interfaces of $\boldsymbol{C_i}$ as:

$$\boldsymbol{I_w = No.of\ possible\ Writing\ Intefaces\ of\ C_i}$$

Step 6: Quantify the *Integrity* level of each of the component $\boldsymbol{INT(C_i)}$ within the range ($\boldsymbol{(0-1)}$ by *Availability* metric as:

(i) $$\boldsymbol{INT(C_i) = \frac{1}{I_w * Cw}}$$

Step 7: Quantify the level of *Integrity* for the overall System $\boldsymbol{INT(Sys)}$ having total $\boldsymbol{N}$ number of total components in the system composition as:

(i) $$\boldsymbol{INT(Sys) = \frac{\sum_{i=1}^{N} INT(Ci)}{N}}$$

Step 8: Stop.

Step 9: End.

4.7 Conclusion and Future Scope

Measuring security always remained a difficult problem to solve, because of the multifaceted nature of the problem. Security of a system is dependent up on the diverse range of factors such as personnel awareness, operating environment, protection mechanisms at hand, and knowledge of the development team and so on. Taking all the factors at one place for evaluating the system for the security makes the evaluation process too complex and inefficient to apply in real practices. From literature survey about 70-80 percent of the vulnerabilities in a system gets introduced during the system development process (requirements, design and coding). Adopting the secure system development process (based on the secure system guidelines) will eliminate the security flaws to some extent but the level of security achieved remains unknown. Software development team must need to take the decision regarding the security of the system and its components based on some quantifiable measures. In this chapter we have proposed the security evacuation framework and derived metrics using mathematical modeling techniques with in a controlled bound (scale). The proposed security evaluation framework strikes at the architectural and design phase of a system lifecycle. The architecture and design phase act as a blue print for the overall system and the evaluation of security at this stage can certainly reduce the cost and efforts required on the further stages of system lifecycle. The selection of Component based Architecture and Design (CBAD) for the proposed framework is influenced by its level in the current architectural and design layer and the easy transformation of system architecture and design using other architectural styles into CBAD. As stated by **[Connolly, 2001]** the completely secure system is one that is disconnected from a network, encased in concrete, and lying at the bottom of the ocean. But to provide the complex business functionalities, in today's networked environment, systems normally composed of independent functional units

(components) which interact, cooperate and coordinate with one another to provide the expected services to its clients. Such interaction, cooperation and coordination results in the dependencies flow of information and resource sharing among the components. The aim of the measurement was to device a framework and derives the metrics that provides the quantative indicators to predict the security level of the system and its components with respect to the four main security attributes: *Dependency, Confidentiality, Integrity* and *Availability.* The proposed framework is at the early stages and needs the further efforts to expand it to take the other security attributes into account. The future efforts are also required to the formalism of the proposed approach.

CHAPTER 5

Empirical Evaluation of Proposed Framework

5.1 Introduction

The most challenging part of the security metrics research is the security metrics validation. There exists no consensus in the research community on a validation framework **[Schneidewind, 1992], [Austin et al., 1990], [Lionel et al., 1995].** Security measurement itself is a hard problem and the validation of the security measures is even harder. From the literature survey, there exists no such validation process of security metrics. In chapter (4) we have proposed a security evaluation framework and derived the metrics and derived the metrics for the four main attributes of the security, *Dependency, Confidentiality, Integrity* and *Availability.* In order to validate the proposed security metrics and also to analyze the feasibility and applicability, in this chapter we perform an empirical evaluation of the proposed framework. Empirical evaluation is often needed in software development, as in all science and technologies to assess the qualities of a method, a tool, or any other entity. The three most common methods used in empirical evaluation are, experiments, case studies and surveys **[Blom, M., 2006].** Our empirical evaluation falls under first category, the experimental approach to present the process of utilizing the security evaluation framework proposed in chapter (4) and to validate the results provided by the metrics in real applications. We demonstrate the use of our security evaluation framework by measuring the security of a running system," Fingerprint Attendance Automation System "(**FAAS)** developed for the department of computer science UoK.

The rest of the chapter is organized as: Section 5.1 prsents the data collection process of the FAAS (figure 5.1) for the security evaluation and summarizes the collected data in the data collection table 5.1. Section 5.3 prsents the security evaluation process using the proposed framework in chapter 4, for each of the component of the system and overall system. The result of the security metrics is presented in table 5.2 & 5.3. It also prsents the resultant security posture of each of the component and overall system in a graphical form in figure 5.4 and 5.5 respectively. The result analysis and conclusion is presented in Section 5.4.

5.2 Data Collection

The empirical evaluation is carried out on a running system "Fingerprint Attendance Automation System "(**FAAS)** developed by the author, for the department of computer science UoK. The main features of the system are:

- It is a web application that can be accessed over the network.
- Having a backend database for the storage of data from various components.
- Provides facility for the automated attendance through finger print scanner attached to the server through in the LAN. The output from the device gets stored to the back end database. And also the information from the system flows out to the Fingerprint scanner.
- Beside the general attendance through Fingerprint scanner, it provides an interactive interface to faculty for individual class attendance for each of the subject taught.
- Provides automated shortage generation.
- Account management for administrator faculty and users with necessary privileges
- Provides the facility to add/update/delete a member or subject from the system.

The selected system has been developed in .NET with C# as programming language and Microsoft SQL Server 2005 as back end database. With the reverse engineering process to get to the design and architectural level of the system, we have captured the component based design and architecture of the system in figure (4.15). The reverse engineering is carried out using tools provided with the .NET framework to collect the various classes and objects and their relations and dependencies with each other in the system. Further the component level design and architecture of the system is prepared using UML 2.0 component based design and architectural notations just like in the proposed framework (chapter (4)) along with the interfaces (provided and required) and the dependencies among the components in "Visual Paradigm for UML 9.0" design tool.

In empirical evaluation we focused less on the specification of the functionality of the system and collected the data required for the evaluation which is.

- *In-Dependency*: denoted by $\boldsymbol{DegIN}(\boldsymbol{C_i})$ in equation (4.2) for each of the component of the FASS architecture and design of figure (5.1).
- *Out-Dependency :* denoted by $\boldsymbol{DegOUT}(\boldsymbol{C_i})$ in equation (4.3) for each of the component of the FASS architecture and design of figure (5.1)
- *Read-Interfaces:* denoted by $\boldsymbol{I_r}$, the number of the reading (provided) interfaces for each of the component of the FASS architecture and design under study in figure (5.1).
- *Write-Interfaces:* denoted by $\boldsymbol{I_w}$, the number of the writing (required) interfaces for each of the component of the system architecture and design under study in figure (5.1) .

As depicted in the figure (5.1) there are 24 fine grained components in the system composition. Beside the implicit interface dependencies among the components, the dependencies (functional control and interface dependencies) are also specified explicitly by dashed arrow lines with the direction of arrow specifies the source and destination of a particular dependency. The system has been developed using three tier architecture, the *application tier*, *business tier*, and the *data tier*. The components in the *application tier* interact with the user of the system to get or provide the relevant information. The name of these components is suffixed by INT in figure (5.1). *Business tier* contains the sole business logic of the system and *data tier* for performing direct operation on the back end data storage.

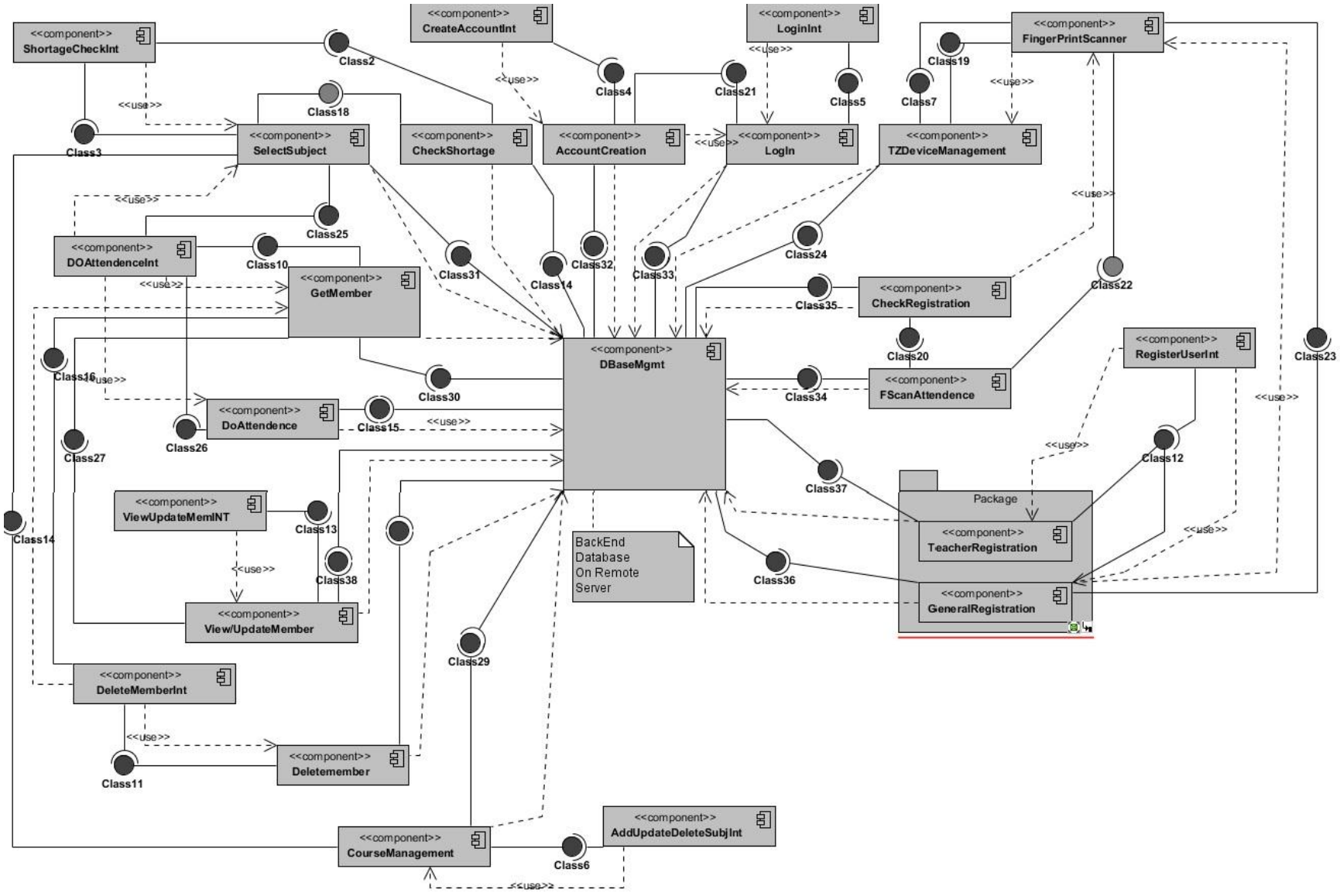

Fig. 5.1 Component Based Architecture and Design (CBAD) of Finger Print Attendance Automation System (FAAS)

5.3 Security Evaluation Process.

By numbering the components form 1 to 24 as shown in below table(5.1). From equation (4.1) the direct dependencies matrix DM of the system architecture and design of FASS in figure (5.1) is:

DM =

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
14	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
21	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
22	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
23	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 5.2 Direct Dependency matrix (DM) of FAAS

From the above figure 4.13 the Full Dependency matrix (FDM) of FAAS by applying Warshall's algorithm is:

FDM =

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1
6	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1
7	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
14	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
21	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
22	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
23	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5.3 Full Dependency matrix (FDM) of FAAS

Table 5.1 Data Collection Table

C_i	Component Name	Direct In-dependency	Direct Out-dependency	$degIN(C_i)$	$degOUT(C_i)$	Read-Interfaces (I_r)	Write – Interfaces (I_w)	Processing +Transmission Delay (P+T)
1	ShortageCheckINT	1	3	1	3	2	1	0.11
2	DoAttendanceINT	1	4	1	4	2	2	0.11
3	ViewUpdateMemINT	1	1	1	3	3	4	0.11
4	DeleteMemINT	1	3	1	3	2	3	0.11
5	AddUpdateDelSubINT	1	3	1	3	4	2	0.11
6	RegisterUserINT	1	2	1	4	2	3	0.11
7	FingerPrintScanner	5	2	6	3	2	3	0.30
8	LoginINT	1	2	1	2	1	1	0.11
9	CreateAccountINT	1	2	1	3	2	2	0.11
10	ViewUpdateMemINT	1	3	1	3	2	2	0.11
11	CheckShortage	2	3	1	2	2	1	0.17
12	AccountCreation	2	3	1	2	2	1	0.21
13	Login	3	2	3	1	2	1	0.23
14	TZDeviceManagement	2	3	6	3	4	4	0.33
15	DoAttendance	2	3	1	2	3	1	0.17
16	SelectSubject	6	2	5	1	2	2	0.15
17	GetMember	7	2	7	1	1	2	0.15
18	ViewUpdateMember	3	3	2	2	2	2	0.20
19	DeleteMember	2	3	1	2	2	2	0.21
20	CourseManagement	2	3	1	2	3	2	0.23
21	Registration	2	3	1	3	3	2	0.19
22	CheckRegistration	2	3	1	3	2	2	0.15
23	FscanAttendance	1	4	1	4	2	1	0.35
24	DBaseMgmt	14	1	24	1	3	5	0.43

By applying equation(4.2) and equation(4.3), on the Full Dependency Matrix (FDM) above figure(5.2), we have calculated $\boldsymbol{DegIN}(\boldsymbol{C_i})$ and $\boldsymbol{DegOUT}(\boldsymbol{C_i})$ respectively of each of the component of the system (FASS) composition in figure (5.1) .

- $\boldsymbol{DegIN}(\boldsymbol{C_i})$ represents the *In-dependency* of component $\boldsymbol{C_i}$ **i.e.** the number of components in the system composition that are directly or indirectly dependent up on the provided services of component $\boldsymbol{C_i}$**.**
- $\boldsymbol{DegOUT}(\boldsymbol{C_i})$ represents the *Out-dependency* of component $\boldsymbol{C_i}$ **i.e.** the number of components in the system composition on which $\boldsymbol{C_i}$ depends for their provided services.

The resultant computation of both the $\boldsymbol{degIN}(\boldsymbol{C_i})$ and $\boldsymbol{degOUT}(\boldsymbol{C_i})$ of is given in below data collection table (5.1). Further the number of *read-interfaces* ($\boldsymbol{I_r}$) and *write-interfaces* ($\boldsymbol{I_w}$) have been computed and presented in table (1). Beside the $\boldsymbol{degIN}(\boldsymbol{C_i})$ and $\boldsymbol{degOUT}(\boldsymbol{C_i})$, we have also computed the direct *In-dependency* and *Out-dependency* of each of the component. The complete data collected from the architecture and design of the system under study (figure (5.1)) is presented in below table (5.1).

The selected system is a web application that runs on a local server over a LAN. We have calculated the average processing and transmission delay together of each of the component by invoking a remote procedure call (RPC) on each of the component and calculated the elapsed time using time stamping from the invocation till the result returned. The components from 1-9, below table (5.1) are responsible for the direct user interaction, so there processing and transmission delay is same. The so calculated processing & transmission delay for each of the component is also presented in data collection table (5.1)

Note: if any of the component $\boldsymbol{C_i}$ having parameter value $\{\boldsymbol{degIN}(\boldsymbol{C_i}),\boldsymbol{degOUT}(\boldsymbol{C_i}), \boldsymbol{I_r}, \boldsymbol{I_r}\} = \mathbf{0}$ then we set that value to $\mathbf{1}$ inorder to eliminate any divide by $\mathbf{0}$ error. The impact of setting value to 1 has very minimum effect of the resultant values.

Applying the steps of algorithms 4.1 to 4.4 from the proposed framework and derived metrics chapter (4), we calculate the level of security with respect to the following attributes for each of the component in the system composition.

- *Dependency:* By applying the steps of algorithm(4.1), chapter(4), and derived metric of dependency in equation (4.4) we compute the dependency level of each of the component.

- *Availability:* By applying the steps of algorithm(4.2), chapter(4), and derived metric of *availability* in equation (4.6) we compute the availability level of each of the component.
- *Confidentiality:* By applying the steps of algorithm(4.3), chapter(4), and derived metric of *confidentiality* in equation (4.9) we compute the *confidentiality* level of each of the component.
- *Integrity:* By applying the steps of algorithm(4.4), chapter(4) and derived metric of *integrity* in equation (4.13) we compute the *integrity* level of each of the component.

The result of the above computed attributes for each of the component is presented in result table(5.2). As mentioned earlier the range of scale for output values is $(0\ to 1)$. Lower the values on the scale higher the effect on the component. The resultant security posture of each of the component is also shown in graphical form in figure(5.4).

In order to calculate the security posture with respect to the *Dependency, Availability, Confidentiality and Integrity* of the overall system, we apply the following derived metrics.

- *Dependency*: Applying the steps of algorithm(4.1), chapter (4) and derived metric of *dependency* of overall system in equation (4.5) we calculate the *dependency* level for the overall system.
- *Availability*: Applying the steps of algorithm(4.2), chapter (4) and derived metric of *availability* of overall system in equation (4.8) we calculate the *availability* level for the overall system.
- *Confidentiality*: Applying the steps of algorithm(4.3), chapter (4) and derived metric of *confidentiality* of overall system in equation (4.11) we calculate the *confidentiality* level for the overall system.
- *Integrity:* Applying the steps of algorithm(4.4), chapter (4) and derived metric of *integrity* of overall system in equation (4.14) we calculate the *integrity* level for the overall system.

The resultant indicators for the overall system are provided in below table (5.3). The resultant security posture of overall system is also shown in a graphical form in figure(5.5).

Table 5.2 Individual Components Resultant Security Indicators

C. No	Component Name	Dependency	Availability	Confidentiality	Integrity
1	ShortageCheckINT	0.25	0.210526316	0.6	0.333333
2	DoAttendanceINT	0.2	0.169491525	0.4	0.125
3	ViewUpdateMemINT	0.25	0.209205021	0.2	0.083333
4	DeleteMemINT	0.25	0.208768267	0.272727273	0.111111
5	AddUpdateDelSubINT	0.25	0.207900208	0.3	0.166666
6	RegisterUserINT	0.2	0.16	0.285714286	0.083333
7	FingerPrintScanner	0.111111111	0.03990423	0.076923077	0.333333
8	LoginINT	0.333333333	0.273224044	0.333333333	0.25
9	CreateAccountINT	0.25	0.205338809	0.375	0.166666
10	ViewUpdateMemINT	0.25	0.209205021	0.272727273	0.111111
11	CheckShortage	0.333333333	0.279329609	0.5	0.5
12	AccountCreation	0.333333333	0.273224044	0.333333333	0.25
13	Login	0.25	0.155520995	0.052631579	1
14	TZDeviceManagement	0.111111111	0.03990423	0.019230769	0.083333
15	DoAttendance	0.333333333	0.279329609	0.4	0.5
16	SelectSubject	0.166666667	0.095877277	0.019230769	0.5
17	GetMember	0.125	0.069300069	0.019607843	0.5
18	ViewUpdateMember	0.25	0.151975684	0.166666667	0.25
19	DeleteMember	0.333333333	0.279329609	0.333333333	0.25
20	CourseManagement	0.333333333	0.279329609	0.285714286	0.25
21	Registration	0.25	0.197628458	0.333333333	0.166666
22	CheckRegistration	0.25	0.197628458	0.375	0.166666
23	FscanAttendance	0.2	0.161030596	0.666666667	0.25
24	DBaseMgmt	0.04	0.020648358	0.000577034	0.2

Table 5.3 FAAS Overall Security Indicators

Security Attributes	SYSTEM (FAAS)
Dependency	**0.2355**
Availability	**0.1822**
Confidentiality	**0.2759**
Integrity	**0.2762**

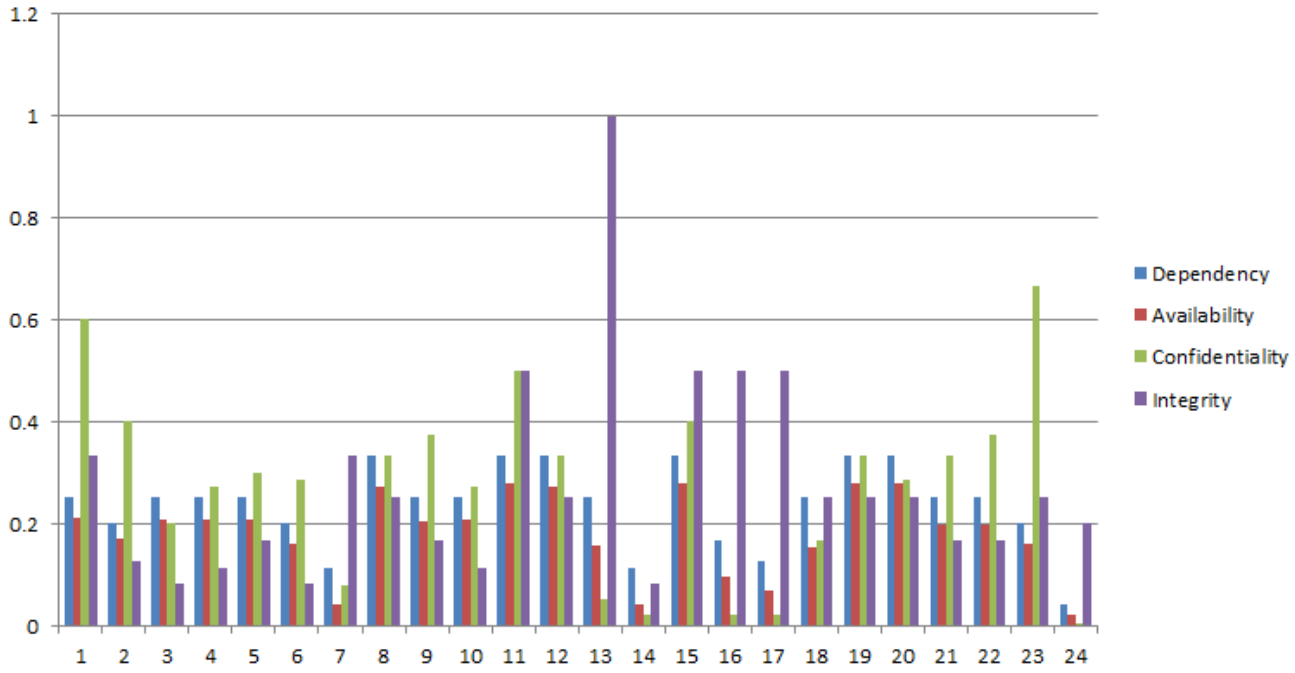

Fig. 5.4. Individual Components Security Graph

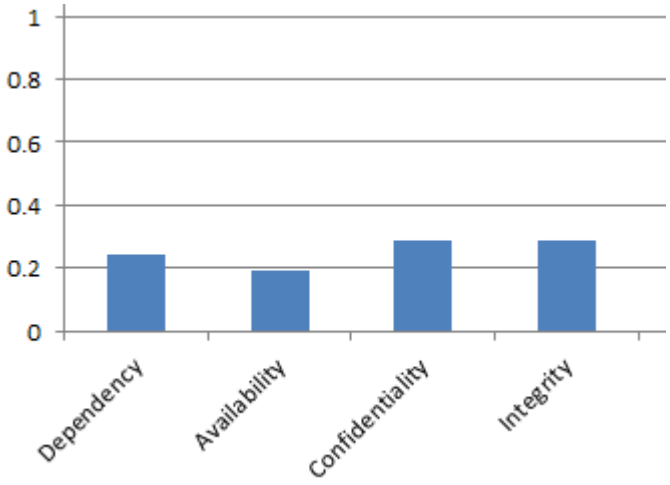

Fig. 5.5 Overall System Security Graph

5.4 Result Analysis and Conclusion

Empirical evaluation is needed to validate the metrics with respect to effectiveness, efficiency, applicability and relevance of a method, model or a framework. We have carried out the experiment empirical evaluation of the proposed security evaluation framework and the derived metrics on a running system (FAAS) in order to check the feasibility and applicability of the

proposed framework. The selected system has been developed by the author for the department of computer sciences UoK. Since the proposed frame work strike at the architecture and design level of the system life cycle , we have performed the reverse engineering process using the tools provided by Microsoft .NET 2008 to extract the class level design and architecture of the system. Further the class level design and architecture is transformed into component level design and architecture using UML modeling techniques in Visual Paradigm for UML 9.0. From the empirical evaluation it is evident that the specification of the proposed framework and the derived metrics in the step wise algorithmic form (Chapter 4) minimized the required efforts in the application of the framework and security metrics in real practices, requiring least personnel knowledge regarding the security.

Comparing the result in table (5.2) with the system architecture and design in figure (5.1) and from our experience with the system development and in running environment, it showed us a great positive response regarding the feasibility and applicability of the proposed framework. Table(5.2) depicts the resultant security indicators in a range of $(0\ to\ 1)$ for each of the component in the system composition, with respect to the four major attributes of security (*Dependency, Availability, Confidentiality* and *Integrity*). We have categorized the component according to the resultant security indicators into five levels $(L1\ to\ L5$, with $L1$ the most severe and $L5$ the least severe) with respect to each of the security attributes. The categorization of the components is based on the following ranges of output values.

- Components having the resultant value between $(0-0.1)$ marked as red in result table(2) are at level$L1$.
- Components having the resultant value in a range $(0.1-0.2)$ marked as purple in result table(2) are at level$L2$.
- Components having the resultant value in a range $(0.2-0.3)$ marked as black in result table(2) are at level$L3$.
- Components having the resultant value in a range $(0.3-0.4)$ marked as blue in result table(2) are at level $L4$.
- Components having the resultant above (0.4) marked as green in result table(2) are at level $L5$.

The resultant categorization of components is shown in table(5.4). Also table (5.3) shows the overall security indicators of the system with respect to four major security attributes on a scale of$(0-1)$.

Table 5.4 Component Categorization Based On Severity Levels

Security Attributes	Component Number(s)	Total No. of Components	Severity Level
Dependency	24,	1	
Availability	7,14,16,17,24	5	**LEVEL(L1)**
Confidentiality	7,13,14,16,17,24	6	
Integrity	3,6,14	3	
Dependency	7,14,16,17	4	
Availability	2,6,13,18,21,22,23	7	**LEVEL(L2)**
Confidentiality	18	1	
Integrity	2,4,5,9,10,21,22	7	
Dependency	1,2,3,4,5,6,9,10,13,18,21,22,23	13	
Availability	1,3,4,5,8,9,10,11,12,15,19,20	12	**LEVEL(L3)**
Confidentiality	3,4,6,10,20	5	
Integrity	8,12,18,19,20,23,24	7	
Dependency	8,12,11,15,19,20	6	
Availability	---	0	**LEVEL(L4)**
Confidentiality	5,8,9,12,19,21,22	7	
Integrity	1,7	2	
Dependency	---	0	
Availability	---	0	**LEVEL(L5)**
Confidentiality	1,2,11,15,23	5	
Integrity	11,13,15,16,17	5	

Beside the numerical representation of the result, the graphical representation about the individual components and the overall system has been presented in figure (5.4) and (5.5) respectively.

As shown in table (5.4) most of the components fall under level $(L3)$ of severity i.e. between $(0.2 - 0.3)$. The components falling under level $(L1\ and\ L2)$ are the most severe with respect to the specified security attributes. Level $(L4)$ is the moderate level and level $(L5)$ is in the safe zone. The output of the security evaluation can act as the input to the decision making in-order to apply the necessary corrective and preventive mechanism with varying level of priority.

CHAPTER 6

Conclusion & Future Work

6.1 Conclusion Drawn

Security evaluation and security metrics are the central issue to make a progress to the secure system development and maintenance. The field of security metrics is young and lacks in the proper terminologies and taxonomies. The main focus in this field remained at the definitional and theoretical aspect, very little has been reported on the actual measuring scale. This thesis presents three main studies.

Since the field is still young and new Chapter 2 presents the preliminaries and common definitions, terms, taxonomies and the main efforts made in security evaluation.

Chapter 3 is an exploratory work which explores the processes tools and techniques used in secure system development process from the requirement to implementation. The chapter also analyzes the need and issues with respect to the security measurement in the system life cycle. Answering the first question "*Where in system life cycle security evaluation is necessary"?* In chapter 3, four key stages have been identified of system lifecycle, which are the candidate for the evaluation of security in order to produce the secure system. From the literature it is evident that the architectural and design phase of a system lifecycle is up to 60-70 percent responsible for the quality of the overall system, and so is the focus of our study..

Chapter 4 is the central focus of the thesis. It presents a novel security evaluation framework and derived metrics using mathematical modeling techniques. The proposed framework strikes at the architectural and design phase of the system lifecycle. Selection of well-defined and well-established security attributes as a baseline to strive for in the evaluation makes the proposed framework scalable, because the evaluation with respect to the specified non-functional security requirements limits the applicability of the evaluation framework to a particular environment and domain. The component based architecture and design (CBAD) using UML modeling techniques has been adopted in the proposed framework. Decision regarding the selection CBAD is influenced by the various advantages like granularity level, required efforts etc. of the selected architectural model and also the easy transformation of other architectural and design models into CBAD. The proposed security evaluation framework takes into consideration the current networked environment and cooperation, coordination & interaction among the various components in the system composition.

Mathematical modeling techniques have been adopted to derive the security metrics in a controlled manner. The specification of overall security metric framework into algorithmic form made it a semi-automated tool to be applied to evaluate the security of the system and its components, in which the input is a system or a component and the out of the algorithm is the resultant security indicator with respect to the specified security attribute. The proposed framework and the derived metrics not only limited to the system in development stages but can be applied on the running systems which requires a pre-evaluation reverse engineering process to catch the architecture and design of the system.

Chapter 5 presents the empirical study of the proposed framework in order to check the effectiveness, efficiency, applicability and relevance of the prosed framework. The result of empirical evaluation showed us a great initial positive response if we compare the results with the system architecture and design and also from the experience with the system in running environment. A major conclusion of the study is that security evaluation is an ongoing process, capturing each and every factor responsible for the security of the system and devising a security evaluation framework to predict the level of security is very hard problem to be solved due to the multifaceted nature of the security, but at the same time the efforts to extract the controlled parameters can be made to provide the indicators of security for decision making in applying necessary corrective and preventive measures.

6.2 Future Work

The research community has acknowledged that security metrics and measurements is avery challenging area in security research **[Manadhata , 2007].** There is, however, a pressing need for practical security metrics and measurements today. The field of security metrics and measurement is a multifaceted and multidimensional in nature which needs to be solved systematically and progressively. At the higher level the problems identified in the thesis is very wide. We strongly believe that each of the identified stages of the system (chapter 3) is a candidate for the evaluation of security and a great deal of efforts are needed to in the future studies to device the new techniques and framework to evaluate the security on each of the identified stage such that each effort made should result in a tool for measuring the security at the respective stage. In particular to the proposed framework (chapter 4) further efforts are required in following ways:

1. Expansion of the framework taking into all the possible security attributes with an insight into the factors responsible in order to derive the metrics for each of them.
2. Measuring security is hard but at the same time it is even harder to validate the security metrics. It is now became a great challenge for the research community. There exists no consensus on a validation framework, very little work has been reported on validation process of the security metrics. Great deal of efforts is needed towards the validation process of the proposed metrics.
3. The empirical evaluation is carried out on a small scale; a large scale empirical evaluation of the proposed framework in needed to analyze the behavior, feasibility, effectiveness, relevance, and applicability of the proposed framework.

Publications

Papers Published

1. **Towards the Application of Security Metrics at Different Stages of Information Systems** (**Irshad Ahmad Mir, Mehraj-U-DinDar, S.M.K Quadri**) *Journal of Global Research in Computer Science* . Volume 2, No. 2, Pages 24-28. ISSN: 2229-371, February 2011.

2. **Analysis and Evaluating Security of Component-Based Software Development: A Security Metrics Framework** (**Irshad Ahmad Mir, S.M.K Quadri**) *International Journal of Computer Network and Information Security(IJCNIS).* Volume 4, No. 11, Pages 21-30, ISSN: 2074-9090 (Print), 2074-9104(Online), October 2012.

3. Security Metric Framework for the Software Architecture and Design Level :An Empirical Evaluation (Irshad Ahmad Mir, S.M.K Quadri), *Second International Conference on Advances in Computer, Electronics and Electrical*, Elsevier , Seek Digital Library , 2013 (Accepted for Publication).

4. **Security Metrics Framework for SOA Systems: A Shift from behavioral to Architectural Perspective** (Irshad Ahmad Mir, Mehraj-U-DinDar, S.M.K Quadri) *In Proceedings of sixth National Conference on Computing for Nation Development, Bharti Vidyapeeth's Institute of Computer Application and Management.* Pages ,pISSN: 0973-7529, pISBN:978-93-80544-00-7, March 2012.

5. Performance Evaluation of Pro-Active Routing Protocols with Fading Models: An Empirical Evaluation using Ns-2 (Bilal Maqbool Beigh, Irshad Ahmad Mir et al.) International Journal of Engineering Science and Technology (IJEST™). Volume 3, No 1, Pages 368-378, ISSN : 0975-5462, January 2011.

Papers Presented

1. **Attack Surface Based security Metrics for Information System** (Irshad Ahmad Mir, Mehraj-U-Din Dar, S.M.K Quadri) *Presented at 6th JK Science Congress, University of Kashmir*, India 2010.

2. **Comparison of Open Source and Closed Source Software Based On Security Evaluation** (Irshad Ahmad Mir, S.M.K Quadri), *Presented at National Seminar on Open Source Softwares: Challenges & Opportunities*, University of Kashmir India, Jun 2011.

References

ABDELLATIEF, MAJDI, et al. Component-based Software System Dependency Metrics based on Component Information Flow Measurements. *ICSEA 2011, The Sixth International Conference on Software Engineering Advances*. 2011.

ALDRICH, JONATHAN, CRAIG CHAMBERS, AND DAVID NOTKIN. ArchJava: connecting software architecture to implementation. *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*. IEEE, 2002.

ALDRICH, JONATHAN, et al. Modeling and implementing software architecture with acme and archJava. *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. ACM, 2004.

ALHAZMI, O. H., Y. K. MALAIYA, AND I. RAY. Measuring, analyzing and predicting vulnerabilities in software systems. *Computers & Security* 26.3 (2006): 219-228.

ALGER, JOHN I. On Assurance, Measures, and Metrics: Definitions and Approaches. *Information Security System Scoring and Ranking (WISSSR).* 2001.

ALONSO, GUSTAVO, et al. Web Services: Concepts, Architectures and Applications. *Springer Publishing Company, Incorporated*, 2010.

ALVES-FOSS, JIM, AND SALVADOR BARBOSA. Assessing computer security vulnerability. *ACM SIGOPS Operating Systems Review* 29.3 (1995): 3-13.

AMERICAN NATIONAL STANDARDS INSTITUTE, Inc., American National Standard T1.523-2001, Telecom Glossary 2000, *Alliance for Telecommunications Industry Solutions*, Feb. 2001.

ANDREWS, MIKE, AND JAMES A. WHITTAKER. Computer security. *Security & Privacy*, IEEE 2.5 (2004): 68-71.

ATKINSON, COLIN, JOACHIM BAYER, AND DIRK MUTHIG. Component-based product line development: the KobrA approach. *KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE* (2000): 289-310.

AUSTIN C. MELTON, DAVID M GUSTAFSON, JAMES M. BIEMAN, AND ALBERT L. BAKER. A mathematical perspective for software measures research. *Software Engineering Journal*, 5(5):246–254, 1990. 5.1

BASS, LEN, PAUL CLEMENTS, AND RICK KAZMAN. *Software architecture in practice.* Addison-Wesley Professional, 2003.

BAUER, BERNHARD, JÖRG P. MÜLLER, AND JAMES ODELL. Agent UML: A formalism for specifying multiagent interaction. *Agent-oriented software engineering*. Vol. 1957. Springer, Berlin, 2001.

BELLOVIN, STEVEN M. "On the brittleness of software and the infeasibility of security metrics." *Security & Privacy*, IEEE 4.4 (2006): 96-96.

BERINATO. S. A. Few Good Information Security Metrics, *CSO Magazine*. 2005. At http://www.csoonline.com/article/220462/A_Few_Good_Information_Security_Metrics?contentId=220462&slug=&

BISHOP, MATT. What is computer security? *Security & Privacy*, IEEE 1.1 (2003): 67-69.

BISHOP, MATT. *Introduction to computer security.* Addison-Wesley Professional, 2004.

BISHOP A, MATT. *Computer Security: Art and Science*. 2003. Westford, MA: Addison Wesley Professional (2003): 4-12.

BLOM, MARTIN. Empirical Evaluations of Semantic Aspects in Software Development. Diss. *Karlstad University*, 2006.

BOEHM, BARRY W. Industrial software metrics top 10 list. *IEEE software* 4.5 (1987): 84-85.

BOEHM, BARRY W., AND PHILIP N. PAPACCIO. Understanding and controlling software costs. *Software Engineering, IEEE Transactions on* 14.10 (1988): 1462-1477.

BOKHARI, M. U., AND NESAR AHMAD. "Analysis of a software reliability growth models: the case of log-logistic test-effort function." *Proceedings of the 17th International Conference on Modelling and Simulation (MS'2006), Montreal, Canada*. 2006.

BOSCH, JAN, AND PETER MOLIN. Software architecture design: evaluation and transformation. *Engineering of Computer-Based Systems, 1999. Proceedings. ECBS'99. IEEE Conference and Workshop on*. IEEE, 1999.

BOOCH, GRADY, JAMES RUMBAUGH, AND IVAR JACOBSON. *The unified modeling language user guide.* Pearson Education India, 1999.

BROWN, ALAN, SIMON JOHNSTON, AND KEVIN KELLY. Using service-oriented architecture and component-based development to build web service applications. *Rational Software Corporation* (2002).

BRUNETON, ERIC, THIERRY COUPAYE, AND JEAN-BERNARD STEFANI. Recursive and dynamic software composition with sharing. *Proceedings of the 7th ECOOP International Workshop on Component-Oriented Programming (WCOP'02).* 2002.

BROY, MANFRED, et al. What characterizes a (software) component? *Software-Concepts & Tools* 19.1 (1998): 49-56.

CENTER FOR INTERNET SECURITY (CIS). The CIS Security Metrics Service. 2008 *http://securitymetrics.org/content/attach/Metricon3.0/metricon3-kreitner%20handout.pdf*

CHANDRA, S., AND R. A. KHAN. Object oriented software security estimation life cycle-design phase perspective. *Journal of Software Engineering* 2.1 (2008): 39-46.

CHEESMAN, JOHN, AND JOHN DANIELS. *UML components.* Addison-Wesley,2001.

CHOWDHURY, ISTEHAD, BRIAN CHAN, AND MOHAMMAD ZULKERNINE. Security metrics for source code structures. *Proceedings of the fourth international workshop on Software engineering for secure systems*. ACM, 2008.

CLEMENTS, PAUL C. **A** survey of architecture description languages. *Proceedings of the 8th international workshop on software specification and design*. IEEE Computer Society, 1996.

CONNOLLY, P. J., 2001. Security protects bottom line. *InfoWorld*, Vol. 23, No. 15, p. 47

CRNKOVIC, IVICA, et al. Specification, implementation, and deployment of components. *Communications of the ACM* 45.10 (2002): 35-40.

CRNKOVIC, IVICA, AND MAGNUS PETER HENRIK LARSSON. *Building reliable component-based software systems.* Artech House Publishers, 2002.

DAVIS, ALAN M. Software requirements: objects, functions, and states. *Prentice-Hall, Inc*., 1993.

DEVANBU, PREMKUMAR T., AND STUART STUBBLEBINE. Software engineering for security: a roadmap. *Proceedings of the conference on The future of Software engineering*. ACM, 2000.

DIALLO, MAMADOU H., et al. A comparative evaluation of three approaches to specifying security requirements. *12th Working Conference on Requirements Engineering: Foundation for Software Quality, Luxembourg*. 2006.

DEMICHIEL, LINDA G., L. ÜMIT YALÇINALP, AND SANJEEV KRISHNAN**.** Enterprise javabeans TM specification, version 2.0." *On-line at <http://java.sun.com/products/ejb/docs.html>, year 2001 (2001).*

DOAN, THUONG, ET AL. "MAC AND UML FOR SECURE SOFTWARE DESIGN. Workshop on Formal Methods in Security Engineering: *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*. Vol. 29. No. 29. 2004.

FAROOQ, S. U., QUADRI, S. M. K., & AHMAD, N. (2011). Software Measurements And Metrics: Role In Effective Software Testing. *International Journal of Engineering Science and Technology (IJEST)*, *3*(1), 671-680.

FAROOQ, S. U., QUADRI, S. M. K., & AHMAD, N. (2012, January). Metrics, models and measurements in software reliability. In *Applied Machine Intelligence and Informatics (SAMI), 2012 IEEE 10th International Symposium on* (pp. 441-449). IEEE.

FIRESMITH, DONALD G**.** Security use cases. *Journal of object technology* 2.3 (2003).

FIRESMITH, DONALD. Specifying reusable security requirements. *Journal of Object Technology* 3.1 (2004): 61-75.

FIPS PUBLICATION 199**:** Standards for Security Categorization of Federal Information and Information Systems. *Federal Information Processing Standards Publication*. 2004.

FOWLER. K AND SCHMALZEL. J**.** Why do we care about measurement? *Instrumentation & Measurement Magazine, IEEE*, 7(1):38–46, March 2004.

GARLAN, DAVID, ROBERT ALLEN, AND JOHN OCKERBLOOM. Exploiting style in architectural design environments. *ACM SIGSOFT Software Engineering Notes*. Vol. 19. No. 5. ACM, 1994.

GARLAN, DAVID. First international workshop on architectures for software systems workshop summary. *ACM SIGSOFT Software Engineering Notes*20.3 (1995): 84-89.

GENSSLER, THOMAS, et al. PECOS in a Nutshell. *PECOS project homepage: http://www.pecos-project.org* (2010).

GRAFF, MARK, AND KENNETH VAN WYK**.** Secure coding: principles and practices. *O'Reilly Media, Incorporated*, 2003.

GOLLMANN. D**,** *Computer Security. Chichester, England*: John Wiley & Sons, 1999.

HAMILTON, DREW, BARRY HOROWITZ, AND CLIFFORD NEUMAN. Systems Security Engineering Final Technical Report. *SERC-2010-TR-005*. (2010).

HEINEMAN, GEORGE T., AND WILLIAM T. COUNCILL. *Component-based software engineering: putting the pieces together*. Vol. 17. USA: Addison-Wesley, 2001.

HENNING. R et al., Proceedings of Workshop on Information Security System, Scoring and Ranking – Information System Security Attribute Quantification or Ordering (Commonly but improperly known as "Security Metrics"), *ACSA and MITRE, Williamsburg, Virginia*, 2002.

HUSSEIN, MOHAMMED, MOHAMMAD RAIHAN, AND MOHAMMAD ZULKERNINE**.** Software Specification and Attack Languages. *Advances in Enterprise Information Technology Security* 2007.

HOWARD, MICHAEL, AND DAVID LEBLANC**.** *Writing secure code.* Microsoft press, 2009.

HOLZMANN, GERARD J. The power of 10: rules for developing safety-critical code."*Computer* 39.6 (2006): 95-99.

HUSSEIN, MOHAMMED, AND MOHAMMAD ZULKERNINE**.** UMLintr: a UML profile for specifying intrusions." *Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on*. IEEE, 2006.

IEEE-STD 830-1998, IEEE Recommended Practice for Software Requirements Specifications, *IEEE Computer Society Press, Los Alamitos*, CA, 1998.

JANSEN, WAYNE**.** Directions in Security Metrics Research. *DIANE Publishing*, 2010.

JEN, LIH-REN, AND YUH-JYE LEE. Working Group. IEEE recommended practice for architectural description of software-intensive systems. *IEEE Architecture*. 2000.

JELEN, G. SSE-CMM. Security metrics. *NIST and CSSPAB Workshop*. 2000.

JÜRJENS, JAN. Using UMLsec and goal trees for secure systems development."*Proceedings of the 2002 ACM symposium on Applied computing*. ACM, 2002.

JÜRJENS, JAN**.** Secure systems development with UML. *Springer*, 2004.

KANNEGANTI, RAMARAO, AND PRASAD CHODAVARAPU. *SOA security.* Dreamtech Press, 2008.

KELVIN, BARON WILLIAM THOMSON**.** Popular Lectures and Addresses: *Navigational affairs*. Vol. 3. Macmillan and co., 1891-1894.

KHAN, MUHAMMAD UMAIR AHMED, AND MOHAMMAD ZULKERNINE. A Survey on Requirements and Design Methods for Secure Software Development. No. 2009-562. *Technical Report*, 2009.

KHAN, MUHAMMAD UMAIR AHMED, AND MOHAMMAD ZULKERNINE. Activity and Artifact Views of a Secure Software Development Process. *Computational Science and Engineering, 2009. CSE'09. International Conference on*. Vol. 3. IEEE. 2009.

KNOWLEDGE Roundtable. Metrics. http://www.knowledgeroundtable. com/app/content/knowledgesource/section/149 (2011).

KRUCHTEN, PHILIPPE B. The 4+ 1 view model of architecture. *Software, IEEE*12.6 (1995): 42-50.

LAU, KUNG-KIU, AND ZHENG WANG. Software component models. *Software Engineering, IEEE Transactions on* 33.10 (2007): 709-724.

LEVERSAGE, DAVID JOHN, AND E. R. I. C. JAMES. Estimating a System's Mean Time-to-Compromise. *Security & Privacy*, IEEE 6.1 2008: 52-60.

LI, BIXIN. Managing dependencies in component-based systems based on matrix model. *Proceedings Of Net. Object. Days*. 2003.

LITTLEWOOD, BEV, et al. Towards operational measures of computer security. *Journal of Computer Security* 2.2 (1993): 211-229.

LIU, M. Y., AND ISSA TRAORÉ. Properties for security measures of software products. *Applied Mathematics and Information Science (AMIS) Journal* 1.2 (2007): 129-156.

LODDERSTEDT, TORSTEN, DAVID BASIN, AND JÜRGEN DOSER. SecureUML: A UML-based modeling language for model-driven security. *«UML» 2002—The Unified Modeling Language* (2002): 426-441.

LIONEL BRIAND, KHALED EL EMAM, AND SANDRO MORASCA. Theoretical and empirical validation of software product measures. Technical Report ISERN-95-03, Fraunhofer Institute for Experimental Software Engineering, 1995. 5.1

LUCKHAM, DAVID C., et al. Specification and analysis of system architecture using Rapide. *Software Engineering, IEEE Transactions on* 21.4 (1995): 336-354.

LYU, MICHAEL R. *Handbook of software reliability engineering*. Vol. 3. CA: IEEE Computer Society Press, 1996.

MADAN, BHARAT B., et al. Modeling and quantification of security attributes of software systems. *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on.* IEEE, 2002.

MAGEE, JEFF, et al. Specifying distributed software architectures. *Software Engineering—ESEC'95 (1995):* 137-153.

MANADHATA, PRATYUSA K., et al. An approach to measuring a system's attack surface. *No. CMU-CS-07-146. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE*. 2007

MCGRAW, GARY, AND BRUCE POTTER. Software security testing. *IEEE Security and Privacy* 2.5 (2004): 81-85.

MCDERMOTT, JOHN, AND CHRIS FOX. Using abuse case models for security requirements analysis. *Computer Security Applications Conference, 1999.(ACSAC'99) Proceedings*. 15th Annual. IEEE, 1999.

MCCURLEY, JAMES, DAVE ZUBROW, AND CAROL DEKKERS. Measures and measurement for secure software development. (2007): 02-05.

MEADOWS, CATHERINE. The feasibility of quantitative assessment of security. *NRL white paper* (1994).

MEI, HONG, JICHUAN CHANG, AND FUQING YANG. Composing software components at architectural level. *IFIP WCC2000, Beijing* (2000).

MEDVIDOVIC, NENAD, RICHARD N. TAYLOR, AND E. JAMES WHITEHEAD JR. Formal modeling of software architectures at multiple levels of abstraction. *ejw* 714 (1996): 824-2776.

MEDVIDOVIC, NENAD, AND RICHARD N. TAYLOR. A classification and comparison framework for software architecture description languages. *Software Engineering, IEEE Transactions on* 26.1 (2000): 70-93.

MEYER, BERTRAND. The grand challenge of trusted components. Software Engineering, 2003. *Proceedings. 25th International Conference on*. IEEE, 2003.

MILES A. MCQUEEN, WAYNE F. BOYER, MARK A. FLYNN, AND GEORGE A. BEITEL. Time-to-compromise model for cyber risk reduction estimation. *In ACM CCS Workshop on Quality of Protection,* September 2005. 7.2.

MOORE, ANDREW P., ROBERT J. ELLISON, AND RICHARD C. LINGER. Attack modeling for information security and survivability. No. CMU-SEI-2001-TN-001. *CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST,* 2001.

MORICONI, MARK, XIAOLEI QIAN, AND ROBERT A. RIEMENSCHNEIDER. Correct architecture refinement. *Software Engineering, IEEE Transactions on* 21.4 (1995): 356-372.

MUSA, J.D., IANNINO, A. AND OKUMOTO, K**.,** *Software Reliability: Measurement, Prediction and Application,* McGraw-Hill, 1987.

MUSKENS, JOHAN, MICHEL CHAUDRON, AND ROB WESTGEEST. Software architecture analysis tool. *Proceedings of the 3d Progress Workshop on Embedded System*. 2002.

NICOL, DAVID M., WILLIAM H. SANDERS, AND KISHOR S. TRIVEDI. "Model-based evaluation: From dependability to security." *Dependable and Secure Computing*, IEEE Transactions on 1.1 (2004): 48-65.

OASIS, Reference Architecture Foundation for Service Oriented Architecture Version1.0. 2009, *OASIS*. p. 119. At *http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.pdf*

OHBA, M. "Software reliability analysis model" IBM Journal. Research Development, Vol. 28, no. 4, 1984. pp. 428-443.

ORTALO, RODOLPHE, YVES DESWARTE, AND MOHAMED KAÂNICHE. Experimenting with quantitative evaluation tools for monitoring operational security. *Software Engineering, IEEE Transactions on* 25.5 (1999): 633-650.

PAULK, M. C., et al. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison Wesley. 1995.

PAYNE, SHIRLEY C. A guide to security metrics. *SANS Institute Information Security Reading Room* (2006).

PEINE, HOLGER. Rules of thumb for developing secure software: Analyzing and consolidating two proposed sets of rules. *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*. IEEE, 2008.

PFLEEGER, SHARI LAWRENCE, AND ROBERT K. CUNNINGHAM**.** Why measuring security is hard. *Security & Privacy*, IEEE 8.4 (2010): 46-54.

PERRY, DEWAYNE E., AND ALEXANDER L. WOLF**.** Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes* 17.4 (1992): 40-52.

PETRITSCH, HELMUT. Service-Oriented architecture (SOA) vs. component based architecture. *Vienna University of Technology white paper, available at http://whitepapers.techrepublic.com/abstract.aspx* (2006).

PLASIL, FRANTISEK, DUŠAN BÁLEK, AND RADOVAN JANECEK**.** SOFA/DCUP: Architecture for component trading and dynamic updating. *Configurable*

Distributed Systems, 1998. Proceedings. Fourth International Conference on. IEEE, 1998.

POUR, GILDA. Component-based software development approach: new opportunities and challenges. *Technology of Object-Oriented Languages, 1998. TOOLS 26. Proceedings*. IEEE, 1998.

QUADRI, S. M. K., N. AHMAD, AND SHEIKH UMAR FAROOQ. "Software Reliability Growth modeling with Generalized Exponential testing–effort and optimal SOFTWARE RELEASE policy." *Global Journal of Computer Science and Technology* 11.2 (2011).

QUADRI, S.M.K., AHMAD, N., PEER, M.A. AND KUMAR, M**.,** "Non homogeneous Poisson process software reliability growth model with generalized exponential testing effort function", RAU Journal of Research, Vol., 16 (1-2), 2006 pp. 159-163.

RAIHAN, MOHAMMAD, AND MOHAMMAD ZULKERNINE. AsmLSec: an extension of abstract state machine language for attack scenario specification. *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*. IEEE, 2007.

RAIHAN, MOHAMMAD, AND MOHAMMAD ZULKERNINE. AsmLSec: an extension of abstract state machine language for attack scenario specification. *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*. IEEE, 2007.

ROSEN, KENNETH H., *Discrete Mathematics and its Applications*, Third Edition, McGraw-Hill, Inc, 1994.

ROY D. FOLLENDORE **III** On The Relationship of Security & Reliability. Retrieved from *http://www.noisetoknowledge.com/on_the_relationship_of_security_and_reliability.htm* on 01-03-2013

SALTZER, JEROME H., AND MICHAEL D. SCHROEDER. The protection of information in computer systems." *Proceedings of the* IEEE 63.9 (1975): 1278-1308.

SAVOLA R. A Taxonomical Approach for Information Security Metrics Development. *Nordsec '07 Supplemental Booklet of Short Papers, Reykjavik, Iceland.* Pages 11-12, 2007.

SAVOLA,REIJO M., AND HABTAMU ABIE. On-line and off-line security measurement framework for mobile ad hoc networks. *Journal of Networks* 4.7 (2009): 565-579.

SAVOLA, R. A. Taxonomical approach for information security metrics development. *Nordsec '07 Supplemental Booklet of Short Papers, Reykjavík, Iceland* (2007): 11-12.

SAVOLA, REIJO**.** Towards a security metrics taxonomy for the information and communication technology industry. *Software Engineering Advances, 2007. ICSEA 2007. International Conference on.* IEEE. 2007.

SAVOLA, REIJO. A novel security metrics taxonomy for R&D Organizations. *In the Proc. of the 7th Annual Information Security South Africa (ISSA'08) Conference, Johannesburg, South Africa.* 2008.

SAVOLA, REIJO. A security metrics taxonomization model for software-intensive systems. *Journal of Information Processing Systems* 5.4 (2009): 197-206.

SAVOLA, REIJO. Towards a security metrics taxonomy for the information and communication technology industry. *Software Engineering Advances, 2007. ICSEA 2007. International Conference on.* IEEE,2007.

SCHNEIER. B**.** Attack trees: Modeling security threats. *Dr. Dobb's Journal*, 1999. 7.2.

SCHNEIDEWIND N.F. Methodology for validating software metrics. *IEEE Transactions on Software Engineering*, 18(5):410–422, 1992. 5.1.

SEDDIGH, NABIL, et al**.** Current trends and advances in information assurance metrics. *Proceeding of the Second Annual Conference on Privacy, Security and Trust*. 2004.

SHAW, MARY**,** et al. Abstractions for software architecture and tools to support them. *Software Engineering, IEEE Transactions on* 21.4 (1995): 314-335.

SHIRAZI H. M., A New Model for Secure Software Development. *International Journal of Intelligent Information Technology Application*, 2009, 2(3):136-143

SINDRE, GUTTORM, AND ANDREAS L. OPDAHL**.** Eliciting security requirements with misuse cases. *Requirements Engineering* 10.1 (2005): 34-44.

STUART EDWARD SCHECHTER. *Computer Security Strength & Risk: A Quantitative Approach.* PhD thesis, Harvard University, 2004. 7.2.

SSE-CMM: Systems Security Engineering Capability Maturity Model. *International Systems Security Engineering Association (ISSEA),* Retrieved on August 4, 2011, http://www.ssecmm.org/metric/metric.asp.

SWANSON. M. Security self-assessment guide for information technology systems. *No. NIST-SP-800-26. BOOZ-ALLEN AND HAMILTON INC MCLEAN VA*, 2001.

SWANSON. M, BARTOL. N, SABATO. J, HASH. J, GRAFFO. L. Security Metrics Guide for Information Technology Systems. *National Institute of Standards and Technology Special Publication 800-55*, Jul., 2003.

SWANSON, M., et al. NIST Special Publication 800-55: Security Metrics Guide for Information Technology Systems. *Technical Report. National Institute of Standards and Technology (NIST), Gaithersburg, Maryland*, 2003.

SZYPERSKI, CLEMENS, DOMINIK GRUNTZ, AND STEPHAN MURER. *Component software: beyond object-oriented programming.* Addison-Wesley, 2002.

SZYPERSKI, CLEMENS, DOMINIK GRUNTZ, AND STEPHAN MURER. *Component software: beyond object-oriented programming.* Addison-Wesley, 2002.

THE FRACTAL PROJECt at, *http://fractal.ow2.org/* 2012.

TYAN, HUNG-YING. Design, realization and evaluation of a component-based compositional software architecture for network simulation. *Diss. School of The Ohio State UniverSity By Hung-ying Tyan,***** *The Ohio State UniverSity*, 2002.

UML, O. M. G. 2.0 Superstructure Specification." *OMG, Needham* (2004).

VAN OMMERING, ROB, et al. The Koala component model for consumer electronics software. *Computer* 33.3 (2000): 78-85.

VAUGHN JR, RAYFORD B., RONDA HENNING, AND AMBAREEN SIRAJ. Information assurance measures and metrics-state of practice and proposed taxonomy. *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on.* IEEE, 2003.

VIEGA, J., MCGRAW, G.: *Building Secure Software: How to Avoid Security Problems the Right Way.* 1st ed. Addison-Wesley. 2001 .

VOAS, JE, et al. Defining an adaptive software security metric from a dynamic software failure tolerance measure. *Computer Assurance, 1996. COMPASS'96,'Systems Integrity. Software Safety. Process Security'. Proceedings of the Eleventh Annual Conference on*. IEEE, 1996.

WANG, HUAIQING, AND CHEN WANG. Taxonomy of security considerations and software quality. *Communications of the ACM* 46.6 (2003): 75-78.

WEYUKER E.J. Evaluating software complexity measures. *IEEE Transactions on Software Engineering*, 14(9):1357–1365, 1988. 5.1

WIEGERS, KARL E. Software requirements. *Microsoft press*, 2009.

WILLIAMS, LLOYD G., AND CONNIE U. SMITH. Performance evaluation of software architectures. *Proceedings of the 1st international workshop on Software and performance*. ACM, 1998.

WILLIAMS, LLOYD G., AND CONNIE U. SMITH. Performance evaluation of software architectures. *Proceedings of the 1st international workshop on Software and performance.* ACM, 1998.

WILLIAM THOMPSON. Popular lectures and addresses. 1891-1894.

WIGLEY, ANDY, et al. Microsoft®. NET Compact Framework *(Core Reference). Microsoft Press*, 2009.

WU, YE, MEI-HWA CHEN, AND JEFF OFFUTT. UML-based integration testing for component-based software. *COTS-Based Software Systems* (2003): 251-260.

XIE, MIN. Software reliability modeling. *World Scientific Publishing Company Incorporated*, vol. 1 1991

YAMADA, S., OHTERA, H. AND NORIHISA, H., "Software reliability growth model with testing-effort", IEEE Transactions on Reliability, Vol. R-35, no. 1, 1986 pp.19-23.

YAMADA, S., HISHITANI J. AND OSAKI, S."Software reliability growth model with Weibull testing-effort: a model and application", IEEE Transactions on Reliability, Vol. R-42, 1993. pp. 100-105.

ZIMMERMANN, OLAF, PAL KROGDAHL, AND CLIVE GEE. Elements of service-oriented analysis and design. *IBM developerworks* (2004).