CHAPTER-2
BACKGROUND AND LITERATURE SURVEY

In the last chapter, secure by design approach and its importance, problem definition, along with overview of research challenge were covered in detail. In this chapter, background concepts and literature review related to present research work were discussed. The research work rests on design and development of secure software. This section will overview different background concepts related to present research work.

2.1 Basic Concepts:

The thesis resolves around some of the basic terms, which were already known to most of the researcher working in this area. The fundamentals concepts include quality, security and defects. These are the basic terms, which can be repeatedly used in this thesis.

**Quality:** The word “quality” is often used to signify the relative worth of something in such phrases as “good quality”, “bad quality” and “quality of life” - which means different things to every person. Software quality is the extent to which system, process or component encounter quantified necessities and customer or user needs or expectations. The concept of software quality is more complex than what common people tend to believe. However, it is very popular for both common personal and IT professionals. If we look at the definition of quality in a dictionary, it is usual to find something like the following: a set of characteristics that allows us to rank things as better as or worse than other similar ones (Jeffrey J.P and Tsai, 1988).

**Security:** The software security chiefly aims at safeguarding of the confidentiality, integrity, and availability (CIA) of the information assets and resources that the software creates, stores, processes or transmits plus the executing programs themselves.

- By preventing unauthorized disclosure, we could preserve confidentiality;
- By preventing unauthorized alteration, we could preserve integrity;
- By preventing unauthorized destruction or refutation of service or access, we could preserve availability.
The software system not only preserves these properties within its digital domain, but also contributes to other systems, the security goals, by following ways:

- By building the real world genuineness of data and users.
- By setting up the idea of accountability of users
- To gain users acceptance of security features, it is necessary to permit usability.

### 2.2 Definitions:

**Vulnerability** - Software vulnerability refers to a software flaw which is caused by a design, implementation, or configuration error which will allow any external intervention to the software to behave in unintended ways. These external interventions were normally malicious in nature. Software security is a software attribute that exemplifies the extent of software susceptibility. If the system is vulnerable, then it is open to attacks which result damages that affect confidentiality, integrity and availability. With the system becoming more vulnerable its security decreases and vice versa. Thus, as the security increases, the vulnerability decreases. Hence, software security can be viewed as an attribute which may be measured by the extent of vulnerability that exists in the software.

**Defect, fault, errors** (Andy JuAn Wang, 2005) - As software errors are the cause of poor software quality, it is important to investigate the causes of these errors in order to prevent them. A software error can be “code error”, a “procedure error”, a “design error”, or a “software data error”. It should be emphasized that the causes of all these errors are human, made by system analyst, programmer, software tester, documentation expert, manager and sometimes clients and their representatives. Even in rare cases where software errors may be caused by the development environment (interpreters, wizards, automatic software generators etc.), it is reasonable to claim that it is human error that caused the failure of the development environment tool.

**Software metric** – It is an order for counting some attribute or trait of a computer software object. One set of such methods aims to determine the quality of a computer program. Software metric measures certain property of a piece of software or its stipulations. Since measurable quantities were indispensable in all disciplines, there is an incessant endeavor by computer science specialist to get along comparable methods to software development. The aim is about achieving our goals, reproducible
and assessable quantities, which might have abundant treasured solicitations in budget and schedule planning, quality assurance testing, cost estimation, software debugging, software performance optimization, and optimal personnel task assignments.

**Software Model** - A description (textual or visual) of any aspect of a software system such as requirements, architecture, behaviour, deployment: the development process model, a requirements model and concurrency model.

### 2.3 Background Terminology:

**Object oriented methodology** - By 1990’s, various methodologies were developed in the software industry for designing software products; but all of them displayed several limitations. So, object oriented methods were developed to overcome them. Object oriented methods which were designed for software development in particular intended at viewing, modeling and implementing the system as an anthology of interacting objects, using the dedicated modeling techniques, languages and activities required to address the particular problems of the object oriented instance. In the evolution of the object oriented archetypes, numerous object oriented software’s were developed.

**UML** - The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas and reusable software components.

**Class Diagram** - A class diagram is used to describe the classes within a model. Classes have attributes (member variables), operations (member functions) and the relationships with other classes, in an object oriented application. A class diagram not only documents, envisages and describes various aspects of a system but also builds executable code of the software application. The class diagram showcases a collection of classes, associations, interfaces, constraints and collaborations. It is also called as structural diagram. The class diagram is used to analyze and design the static view of an application.

**UML sec** – UML sec is an extension to the Unified Modeling Language for integrating security related information in UML specifications. This information can be used for
model based security engineering. Most security information is added using stereotypes and cover many security properties including secure information flow, confidentiality and access control.

2.4 Literature review:
This section reviews the work carried out in the area of secure software engineering by the researchers from both the industry and the research institutes worldwide. The associated work, published in the standard international and national journals, proceedings of international and national conferences, various articles related to the international and national symposiums and workshops were referred for this work. The work, which is also available in standard books and titles, group newsletters, websites, etc, has been presented with a bird’s eye-view. Based on the research work, limitations of the existing methodologies related to software security have been traced. This has lead to the motivation for the present investigation that has been concluded from the literature survey.

- **Testing for Security during development: Why We Should Scrap Penetrate-and-Patch:** In this paper author has mentioned disadvantages of Penetrate and Patch testing and mentioned the need to consider the security in former stages of software development life cycle. Security is assessed by attempting to break into an installed system by exploiting well known vulnerabilities. If a break-in attempt is successful, the vulnerability that permitted the security Penetrate and patch, and the tools that help automate it, will always have a place in the security tool box. Thus, making it prominent that significant portion of computer security violations occur because of errors in software design and coding (Gary McGraw, 1998).

- **Prioritization and Selection of Software Security Activities:** Author has mentioned that security must be incorporated in software development by introducing software related activities. Software security process (S3P) has been proposed by the author who helps the developer to estimate cost of security related activities and how to select the security related activities that best suits their needs. In this paper author has presented the process component definition step for S3P, which consists of three parts: assigning costs to security-related activities, selecting the best set of activities that meet a given security goal, and expressing the resulting selection in terms that suit the target process (David Byers...
• **Software Security: A Vulnerability-Activity Revisit**: Author proposed a system which attempts to provide security by preventing already known vulnerabilities. The implementation and mitigation method of these vulnerabilities is discussed in design or implementation phase. It provides understanding of the vulnerability in initial steps. According to this mapping the most important decision is selecting a programming language and using least privileges. It describes about twenty three most common vulnerabilities with their effects on software security. In contrast with a like study labors, this paper, not only essence on the utmost imperative susceptibilities, characterized security events into and implementation phase nineteen activities and design phase thirteen activities. (M.A. Hadavi et al, 2008).

• **Information Security Models and Metrics**: Author has proposed a system by two alternate approaches for security assessment, which include analytical modeling and metrics. Author has presented a formal model for analyzing security attributes scientifically and accurately over experiment. It includes a collection of different mathematical formulas for quantitative metrics instead of qualitative measures. Author has proposed an RCC (Relative Complete Coverage), which includes a multi-level modeling for the vulnerability model using various techniques and mechanism of risk management, for reducing and maintaining the level of risk (Andy Ju An 2005).

• **Design of a Process for Software Security**: This paper is based on prevention of vulnerabilities by applying the testing activities at an early stage by vulnerability cause graph. It is a formal approach based on modeling of the causes of vulnerabilities and trying to prevent them. Author mentions that the given process is straight forward, simple, flexible and suitable for business requirements (David Byers and Nahid Shahmehri, 2007).

• **A Metrics Framework to Drive Application Security Improvement**: In this paper, the author explains the need of integrating security in software development life cycle. Early involvement in the life cycle ensures less security impact as the application is deployed. The key to success for the security team’s participation in the development life cycle is to understand what elements exist that can be leveraged for security purposes and where there are gaps to be filled with
additional security-centric artefact and activities (John Steven and Gunnar Peterson, 2007).

- **Towards Measuring Framework for Security Properties of Software**: This paper highlights the fundamental questions that need to be answered in order to bridge the gap and proposes an initial approach. Based on security principles and security properties, new metrics were proposed to measure them in this work. Author focuses on architecture and design phase for metric calculations. For this, the author has presented framework extract with relevant security properties which can be measured by security principles and practice (Riccardo Scandariato, 2006).

- **Software Security Metric Identification Framework (SSM)**: This paper introduces quantitative assessment of product security by reducing error rates. Proposed modeling techniques include threat and vulnerability modeling to moderate security. High Level Diagram (HLD) and Low Level Diagram (LLD) can be used as an input to the process. Proposed estimation process is based on five different phases. Goal of Software Estimation Framework (SEF) is to provide high-level protection to the software and contribute to the mitigation of security failures (S Chandra and R Khan, 2009).

- **A Knowledge Management Approach to Support a Secure Software Development**: Author has proposed PSSS (Process to Support Software Security) to help in secure software development by integrating security activities in software development life cycle. PSSS gives the support of a knowledge management environment based system, specially, on security inspections of the artifacts generated during the execution of the processes. In addition to that, author has also proposed a checklist to security inspections on the software requirement (Francisco Jose and Barreto Nunes, 2009).

- **Software Security Checklist for the Software Life Cycle**: In this paper, the author focuses on the development of a Software Security Checklist (SSC) for the life cycle. It includes all phases of software development life cycle. Integrating security in the software life cycle should be processed right from the beginning with a security risk analysis and requirements gathering, through design and development, testing and integration. The author has stated corporate policies and
the requirements on security as well as guidelines which might be complicated. (David P. Gilliam, Thomas L. Wolfe, Josef S. Sherif, 2003)

- **Security Estimation Framework: Design Phase Perspective:** This paper provides a methodology to quantify a software security in early stage of development with an approach that uses a sequential method for estimation and mitigation vulnerability in development phase. It analyzes non functional properties at early stages. It helps to reduce, control and improve development time, efforts and cost (S Chandra et al, 2009).

- **Quantifying Software Performance, Reliability and Security: An Architecture-Based Approach:** In this paper, the author has proposed an architecture-based on unified hierarchical model. It analyses the various properties like software security, performance, cache behavior prediction and reliability. Author uses discrete time Markov chains (DTMCs) to model software. It is used for the prediction of the behavior of the system including architecture and individual component characteristics and also the bottlenecks. Author has proposed a hierarchical method for prediction of a software system attributes on the basis of software architecture and individual components attributes. It also allows the change in individual components without any loss. Correction is followed by testing simultaneously. Author has proposed a formal sensitivity analysis and brute force approach (Sharma V.S, 2007).

- **Statistical Analysis for Object Oriented Design Software Security Metrics:** To improve any process we have to measure that process. Based on this fact, in this paper present evaluation of CK metrics suits’. The definition of six different metrics is presented in this work. These metrics are validated by projects that use object oriented language, and are useful to get quick feedback for software designers and managers. (D. Sravan Kumar and M. Upendra Kuma, 2010).

- **Detection and Correction of Design Defects in Object-Oriented Architectures:** The recognition and rectification of design flaws are problematic due to absence of exact stipulations of faults and mechanisms. Author has provided a systematic technique to postulate design faults exactly and to create recognition and alteration algorithms expending refactoring from their stipulations semi-automatically (Naouel Moha, 2007).
• **Detection and Correction of Design Defects in Object-Oriented Designs**: In this paper, the author has proposed a method to detect design defects precisely and to generate automatic detection and correction algorithms from their specifications. The proposed algorithms were based on metrics, semantics and structural properties. The correction algorithms were based on refactoring rules. Author has validated these algorithms on commercial open-source projects implemented in object-oriented methodology. The algorithm allows a systematic specification with precise detection as well as suitable correction of design defects. (Naouel Moha, 2007).

• **Hierarchical Model for Object-Oriented Design Quality Assessment**: In this paper, the author has proposed hierarchical model for quality assessment of object-oriented designs. In this model, structural and behavioral design properties like classes, objects, and their relationships were examined using object-oriented design metrics. This model tries to find relationship between design properties and quality attributes. Design properties like encapsulation, modularity, coupling, and cohesion were considered and quality attributes like reusability, flexibility and complexity were considered (Jagdish Bansia, 2002).

• **An Algorithm to Measure Attribute Vulnerability Ratio of an Object Oriented Design**: To minimize vulnerabilities and achieve target level security, quantification of security is necessary. Based on this idea, an effort has been made by the authors to introduce a methodology to find out the impact of inheritance on vulnerability propagation in object oriented design. In this work, an algorithm to measure the Attribute Vulnerability Ratio (AVR) is present for object oriented design. The implementation of such vulnerability is done using a case study of Automated Teller Machine (ATM) which illustrates the applicability of the approach. AVR calculates how a single vulnerable class can manifest itself in class hierarchy and design whole software becomes vulnerable. (A. Agrawal and R.A. Khan, 2009).

• **Security Metrics for Object-Oriented Class Designs**: In this paper, the author has presented new metric for object oriented class design. It also allows the designer to compare different alternative designs from security point of view. Author has proposed seven security metrics to determine encapsulation and
cohesion properties of given object-oriented class. Metrics based on security principles like least privilege, reduce attack surface, and were defined on the grounds of probable information stream properties inside a specified class in the complete class diagram. The metrics have been arranged to fit in between the range 0 to 1. A low value is desired for each (Bandar Alshammeri, 2009).

**Is Complexity Really the Enemy of Software Security?** Authors have performed statistical analysis on nine code complexity metrics from the Java script engine in the Mozilla application framework to investigate whether this hypothesis is true. Initial results show that the nine complexity measures have weak correlation with security problems for Mozilla Java script engine. The study needs to be replicated on more products with design and code level metrics. It may be necessary to create new complexity metrics to embody the type of complexity that leads to security problems (Yonghee Shin and Laurie Williams, 2008).

**Can Complexity, Coupling, and Cohesion Metrics be Used as Early Indicators of Vulnerabilities:** In this paper, author proposes coupling, complexity and cohesion (CCC) related software metrics that can be measured during the preliminary stages of software development. The author has deduced that if experimental associations can be exposed flanked by CCC metrics and susceptibilities, then these given metrics could be obliging for software developers to take hands on activities in contradiction of prospective susceptibilities in the software (Istehad Chowdhury and Mohammad Zulkernine, 2011).

**An Efficient Measurement of Object Oriented Design Vulnerability:** Author has proposed a procedure for calculating Vulnerability Propagation Factor (VPF) in this paper. This can determines number of susceptible classes in some design classes. The suggested algorithm investigates whether inheritance subsidizes to promulgation of susceptibilities from one class to another or not. It is founded on the point that a single susceptible class can establish itself in class hierarchy, as well as in the designing of whole software susceptible. The VP calculates number of classes influenced due to a susceptible class in an inheritance hierarchy of an object oriented design. On these grounds of VP, Vulnerability Propagation Factor (VPF) for the design was considered (Alka Agrawal et al, 2009).
• **A Framework to Detect and Analyze Software Vulnerabilities - Development Phase Perspective:** Author has proposed a software vulnerability detection and analysis framework (SVDAF) which covers all phases in software development life cycle. For each phase security checklist has been prepared which is verified after completion of appropriate phase that ensures that the output of particular phase fulfils prerequisites for security of the phase. If it is found to satisfy, then it declares that phase as secure. If not, the output is forwarded to vulnerability life cycle, where it is properly analyzed for the detection of the vulnerabilities. Finally, documented suggestions, in the form of feedback were sent as input to the corresponding phase of SDLC from where the vulnerable output was received as input (A. Agrawal and R.A. Khan, 2009).

• **Confidentiality checking an object-oriented class hierarchy:** Author mention that the class diagrams is right place to check confidentiality, we can observe, analyze and test, how and where the information is glided. In the confidentiality inspection process, the sensitivity of both method and class must be verified. They shouldn’t be hindered. Reusability is a vital piece of object oriented model which is responsible for multiple uses of classes and methods in diagram, but it could also cause further cases in which admittance license is mandatory. For these purpose confidentiality verifications of a class hierarchy algorithm is proposed in this work. It is essential to confirm any sensitive data were not public (S. Chandra and R.A Khan, 2010).

• **Software security: A Quantitative Approach:** Literature lacks about any efficient method or tool to estimate security in early stages of software development life cycle. In order to improve security of the software from the beginning, security quantification was required. Author has introduced primary concepts of security that may be used for quantitative estimation, like facts that may help to understand the software architecture and to improve software security. The proposed method was broken-down into its multiple phases (S. Chandra and R A Khan, 2010).

• **Object oriented design security quantification:** The proposed study talks about a systematic approach to quantify security based on complexity factors which are having impact on security attributes. This paper guides researchers and software
practitioners to evaluate, and preferably, measure software security in design phase. A security assessment through complexity framework (SVDF) has been suggested in order to develop quality products by including security features. It may then be used as a measure to test severity of software products (Suhel Ahmad Khan and Raees Ahmad Khan, 2011).

- **Integrity Quantification Model for Object Oriented Design:** Integrity Quantification Model (IQM) was proposed by the author for object oriented design. The proposed model was based on the facts that the identification of complexity factors that influence integrity at design phase, Identification of objects oriented design characteristics and establishment of correlation between these two. The relative significance of individual factors were weighed proportionally, based upon the relationship of the integrity security factors and complexity factors (Suhel Ahmad Khan and Raees Ahmad Khan, 2012).

- **A Framework to Quantify Security: Complexity Perspective:** Complexity is enemy of software security, stated in many research work. Based on this fact, an effort was made to measure impact of complexity on object oriented design parameters to quantify security. A Security Assessment through Complexity (SAC) framework is proposed in this work and relationship from design parameters to complexity and security was defined with its expected influence and importance (Suhel Ahmad Khan and Raees Ahmad Khan, 2012).

- **An Empirical Validation of Integrity Risk Factor Metric: An Object-Oriented Design Perspective:** Based on earlier research work author has concluded that the empirical relationship could exist between security metrics and security problems, and this may help to improve overall product security. In this work they focus on empirical validation of the method using nine different class hierarchies of software. To develop the quantitative assessment of integrity risk, two security metrics and integrity state transition model has been proposed in this work. Based on existing loopholes in class diagram, the proposed model measures and ranks the security of software (Shalini Chandra Raees and Ahmad Khan, 2013).
• **Authorization Estimation Model: An Object Oriented Design Complexity Perspective:** Author has developed metric for authorization based on the structural and behavioral design properties. For this, the author was assumed complexity as key factor classes and their relationship information is used to develop metric for authorization. The metric was developed using multiple linear regression technique on object oriented design constructs. Author has used statistical analysis which concludes its statistical significance remarked that calculated data was highly acceptable (Suhel Ahmad Khan and Raees Ahmad Khan, 2014).

• **Security Improvement of Object Oriented Design using Refactoring Rules:** In this work, a case study was adopted based on refactoring rules mentioned by Fowler to implement the Security Improvement Guidelines (SIG). The developed Security Quantification Model (SQMOODC) was calculated the quantified value of security at each step of SDLC. SIG is helpful to stop the security flaws. Each refactored steps, which are done on case study, match the prediction of the impact for refactoring rules on security (Suhel Ahmad Khan and Raees Ahmad Khan, 2015).

• **Modeling and Quantification of Security Attributes of Software Systems:** Author has focused various issues for enumerating the security characteristic of an intrusion tolerant system (SITAR). By expending arbitrary sculpting method to seize the attacker’s conduct and the system’s retort to a security invasion. Suggested prototype scrutinizes and calculates the security characteristics of the system. At first, the consistent behavior directing to procedures like steady state accessibility is used to quantification analysis. To study and enumerate the security characteristics of such a system, author considered not only the system’s response to a security attack, but also the actions taken by an attacker to cause such an attack (Bharat B. Madan et al, 2002).

• **Quantifying the Quality of Object Oriented Design: the Factor-Strategy Model:** A Factor Strategy model was proposed in this paper, which was different from traditional quality models. It relates quality of a design to its conformance with a set of essential principles, rules and heuristics. This model was based on detection strategy that raises the abstraction level in dealing with metrics, by
allowing to formulate good design rules and heuristics in a quantifiable manner and to detect automatically deviations from these rules. This model provides an easier construction and understanding of the quality which was based on design principles rather than “raw numbers” and a direct classification of the causes of quality flaws (Radu Marinescu and Daniel Ratiu, 2014).

• **Software Security: Integrating Secure Software Engineering in Graduate Computer Science Curriculum:** For effective consideration of security in earlier stages we need to educate and train peoples with security issues. Software industry had already identified this need as Microsoft has propelled its reliable totaling ingenuity, and IBM has taken under way its SPADE (Security and Privacy Aware Development Environment) project. Software engineers ought to know the information regarding how to trail the finest schemes of supplying protected software over numerous training and educational suites. But since the software security is a precise complicated idea, it must be concerning security prerequisite engineering, design ethics and strategies, execution risks, security testing and investigation methods. With appropriate progression and test centre material, universities must permit students to comprehend the schemes and methods in addition to use operative apparatus for development of secure software (Stephen S. Yau and Zhaoji Chen, 2006).

• **Security Requirements for the Rest of Us: A Survey** to identify threats, assets and security objectives, a survey had been conducted by the author, which determined most commonly suggested security requirement tasks. Aim is to identify a set of security requirement techniques well situated for average software developers. It is hard to identify all security requirements but can focus on the most important. After iterated process, developers will be able to modify and add further requirements at later stages. Practically because of lack of a common definition of a security requirement, the developers need concrete examples for describing and documenting such requirements (Inger Anne Tondel et al, 2008).

• **On the secure software development process: CLASP, SDL and Touch points compared:** Software processes provide common knowledge and normal practice in most development organizations. Unfortunately, these processes offer little support to meet security requirements. In this paper, author has compared three
processes (OWASP’s CLASP, Microsoft’s SDL and McGraw’s Touch points) for the development of secure software. The paper identifies the commonalities, discusses the specificity of each approach and proposes suggestions for improvement (Bart De Win, et al).

- **Dependability and Security Models:** In this paper, author states a new categorization of security models and dependability. In this model, the author has firstly discussed the classification of threats and mitigations in systems and networks, and then several individual model types. These includes availability, confidentiality, integrity, performance, reliability, survivability, safety and maintainability have been presented. Finally, the author has validated that each model type can be united and symbolized by one of the model representation techniques, namely: combinatorial (such as reliability block diagrams (RBD), reliability graphs, fault trees, attack trees), state-space (continuous time Markov chains, stochastic Petri nets, fluid stochastic Petri nets, etc) and hierarchical (e.g., fault trees in the upper level and Markov chains in the lower level). Case studies for each individual model types as well as composite model types have been studied in the present work (Kishor S. Trivedi et al, 2009).

- **An Attack Surface Metric:** In this paper, the author suggests that the attack surface measurement of a software system acts as a pointer of the security of the system. We familiarize an attack surface metric to amount the attack surface in a systematic fashion. The method of calculation used by author was unconvinced to the implementation language of a software system and is applicable to systems of all sizes. They elucidate technique by gauging the attack surfaces of small desktop applications and hefty enterprise systems realized in C and Java. Author carried out three analytical empirical studies to authenticate the said method. By computing and plummeting the software’s attack surfaces, the software designers can alleviate their software’s security risk. The software industry’s outdated code quality enhancement method for security risk mitigation accompaniment mention attack surface reduction style and is obliging in several phases of the software development lifecycle. Author’s collaboration with SAP demonstrates the use of mention metric in the software development process (Pratyusa K. Manadhata, and Jeannette M. Wing, 2011).
• **Towards the Application of Security Metrics at Different Stages of Information Systems:** The intimidations in the functioning environment amplified the consideration towards integrating security. By means of such incorporation of security schemes, the question that arises is what could be the level of security in the system and how much protected. The response to this query was assumed by the application of security metrics to examine the fallbacks. Security metrics have a vital part at every single phase of information systems development as well as in functioning environment. In this paper, author has concentrated on the validity of such security metrics at diverse phases of information systems life cycle and recognizing some metrics structure existing for every phase of the system (Irshad Ahmad Mir et al, 2011).

• **A System for Identification and Assessment of Secure Design using Dynamic Security Metrics:** In this work author develops a tool which applies on static and dynamic metrics on design. And also applies reverse engineering to validate the design. Firstly, the input for the system will be UML diagram which was then applied to the refactoring algorithm to get multiple design of the same diagram. Secondly, author has applied on all the designs, the design level static and dynamic metrics to get secure design which was a result of design level metrics. Finally, in third step, implementation of that secure design was done and then it is reverse engineering concept for the validation of secure design was applied (Devendra Singh Thakore and Akhilesh R. Upadhyay, 2013).

• **Development of Object-Oriented Analysis and Design Methodology for Secure Web Applications:** For developing web applications using object-oriented methodology Java EE technologies were used. But security method by Java EE mechanism was implemented at the last step, since it is difficult to apply constant security during the whole software development process. Therefore, author has presented an object-oriented analysis and design methodology emphasized in the security for secure web application systems from the requirement analysis to implementation. By adopting UML sec notations in modeling language one could highlight on security for the requirements (Kyung-Soo Joo and Jung Woong Woo, 2014).
• **A Roadmap for Security:** Object oriented methodology was used to represent the problem domain to clear, develop and understand the design. Author suggests the complexity as a key factor of security. By proper adjustment of design characteristics and relation between them, complexity can be restricted for acquiring secure software. The security of software can be achieved by controlling the complexity. By controlling the design characteristics and their relation it is possible to develop a product which is more reliable and secure, (Raees Ahmad Khan, Suhel Ahmad Khan, 2010).

• **Attack Surface Metrics and Automated Compile-Time OS Kernel Tailoring:** Author has evaluated the security benefits by measuring and comparing the attack surface of the kernels which was obtained. The notion of attack surface was formally defined and evaluated in a very generic security model, as well as a security model taking specifically into account the threats posed by a local unprivileged attacker. The resulting configuration leads to a Linux kernel in which unnecessary functionality was removed at compile time and thus, inaccessible to attackers. Author has evaluated this reduction using a number of different metrics, including SLOC, the cyclomatic complexity and previously reported vulnerability reports, resulting in a reduction of the attack surface between about 50% and 85%. The evaluations also indicate that this approach reduces the attack surface of the kernel against local attacker’s significantly more than previous work on kernel extension isolation for Linux (Anil Kurmus and Reinhard Tartler, 2014).

• **Properties for Security Measures of Software Products:** Authors have identified number of internal software attributes that could be related to a array of security qualities. Since theoretical validation was an important step in the development of any metrics program, they focused on studying the measurement properties associated with these internal attributes. The property, based on popular security design principles was used in security engineering processes, which may guide the search of software security metrics. They studied the feasibility of our theoretical framework by presenting case studies based on metrics derived from existing security measurement frameworks, namely the attack surface metrics system and the privilege graph paradigm. They have defined and formalized a collection of properties characterizing security related internal software attributes.
The properties were based on security design principles widely accepted in the security engineering community. This represents an important contribution in the field of software security metrics, as the field was still in immaturity. It was important to note that the internal attributes suggested in this paper cover only a limited view of software security (Yanguo Michael Liu and Issa Traore, 2007).

- **A Vulnerability Metric for the Design Phase of Object Oriented Software:** Unlike quality, quantitative estimation of security at design phase of object oriented software was basically not covered. The work examines that coupling was one of the object oriented design feature accountable for propagation of vulnerabilities in the design of software. A metric was proposed to determine whether the design of one version of a software system was more vulnerable than another with respect to propagation of vulnerability. Unlike, counting bugs at the code level or counting vulnerability reports at system level, the proposed metric measures the overall propagation of vulnerabilities in an object oriented design. A weak foundation not at all tends to create vulnerability resistant software. In absence of any tool or approach, it was hard to expect the security of an object oriented design. A metric, CIVPF, has been proposed and the algorithms for automatic calculation of the metric were designed. The metric calculates comparative vulnerability of two object oriented designs of identical software as well as designs of different object oriented software (A. Agrawal and R. A. Khan, 2014).

- **Availability State Transition Model:** An idea to model availability states an Availability State Transition Model (ASTM) has been proposed in this article. In ASTM methodology, only design level details were required which can be easily retrieved from the software’s design. It was well accepted agreement that reusability is very strong attribute of object oriented programming but this feature leads to make classes dangerous and creates loopholes which may be exploited without difficulty. ASTM may help to find out which class needs to secure or where sharing of methods need to decrease. Author has identified various significance of ASTM like it is easy to find out class’s state, easy to trace dynamic behavior of class and able to find out the reason that why class become risky. It has also important to take decision, what action may require for making it safe and
secure. In ASTM methodology, only design level details was required which can be easily retrieved from the software’s design (Shalini Chandra & Raees Ahmad Khan, 2011).

- **Assessing Impact of Cohesion on Security - An Object Oriented Design Perspective:** This research paper not only analyses the role of cohesion for object oriented design security but also proposes security metrics measuring the impact of cohesion on security vulnerability. Reducing vulnerabilities of an object oriented design with the help of its constructs such as cohesion, encapsulation, inheritance and coupling was almost missing in the literature. The proposed metrics were the part of the work which aims to develop a security metric suite. With the help of existing literature on cohesion, it was established that cohesion of vulnerable attributes with the methods goes against the security design principles. Hence it could be concluded that cohesion of vulnerable attributes must be minimized. Method level, class level and design level cohesion metric vulnerable association (VA) was developed to measure the effect of cohesion on security of object oriented design. An algorithm was proposed to compute the metric VA. The metric was analyzed to reveal the fact that greater the Vulnerable Association (VA), greater the vulnerability and hence less security in the design (A. Agrawal and R. A. Khan, 2014).

- **Securing Object Oriented Design: A Complexity Perspective:** Complexity is often hypothesized to be the enemy of software security. If this hypothesis is true, complexity metrics may be used to predict the conditions of security problems and can be used to prioritize inspection and testing efforts. Our goal is to investigate the effect of software complexity to security problems and to identify metrics that represent the complexity that lead to security problems. Author has used statistical analysis and machine learning to build models to capture the relationship between complexity and security problems using open source and industrial projects. The initial case study on the Java Script Engine in the Mozilla application framework shows that our initial modeling and complexity metrics need to be improved, but our approach is a feasible means of identifying vulnerable areas of code (Suhel Ahmad Khan and Raees Ahmad Khan, 2014).
2.5 Limitation of existing methodology:

There were three main methodologies to develop secure software. The first method to do so is called as Penetrate and Patch. It refers to applying patches to recover susceptible applications. This approach is most commonly used for securing software applications. But using this method, it becomes more tedious and uneconomical to detect and fix a bug in an already released software product which further shoots the expenses to hundred times of the cost required in fixing the same bug in the development phases of that software. And still, the patches don’t guarantee you 100% assurance of removal of vulnerabilities from the software. The second approach bull's eye the safety of functioning environment. It is dependable on expedients which were external for a software system, for instance, the intrusion detection systems and firewalls. Even though these defense mechanisms can deliver security to some extent but they remain inefficient in sustaining resistance of the application against external software design or execution. The other downside of these methods is that they can be brought in to work only when the software is in functioning mode. The third approach for building software security is called as secure software engineering. In this method, considering security of the system in mind, the emphasis remains on implementing well structured processes centered on the investigation of necessity done through application and design.

Also, as per CERT/CC report, it has been proved that more than 90% of the susceptibilities were leaked out during the development of the system software. These are nothing but the outcome of overlooking the susceptibilities found in the other systems (Nancy R. Mead, Gary McGraw, 2005). The same report also validates that there are ten world wide-known susceptibilities which account for 75% of software security breaks in the present day applications. Thus, if the developers were well acquainted with them, then about 75% of the susceptibilities can be precluded. Based on these facts many researchers had suggested frameworks for software development life cycle. Those frameworks were mention in literature review. But they were still not providing the practical solution to the security problem.

In case of huge object-oriented programs, it turns out to be luxurious to maintain them due to the architectural drift and bad design practices used (Perry, D.E. and Wolf, A.L, 1992), which creates adding, debugging, and evolving attributes more tedious.
Such poor design practices form the root causes of design defects. In object-oriented architectures, design defects were poor solutions to chronic design problems which further present difficulties in the design and have negated consequences on maintenance. They consist of matters with numerous granularities, from architectural (global) glitches, such as anti patterns, to low-level (local) hitches, such as code smells (M. Fowler, 1999) (such as long methods, large classes, or long lists of parameters). It is easier to comprehend and maintain good software architecture without design defects. Still, the detection and correction of defects remains a manual activity in case of large architectures due to lack of tools. These happen so due to the fact that a flaw affects diverse classes and methods and are fundamentally founded on documented explanations subject to dissimilar clarifications.

Software security is a dynamic property as software that was protected in a particular environment within a particular threat landscape may no longer be secured if that environment or threat landscape changes or if the software itself changes. Security must be understood in terms of a simple question: Secure against what and from whom? Understanding security is best understood by thinking about goals. What is it we are trying to protect? From whom are we protecting it? How can we get what we want? (O.H.Alhazmi et al, 2007), the answer of these questions is secure software which is in and of itself robust against attack.

Software security consists of internal weakness and external attacks. The external danger every so often breaks a software system by exploiting its internal weakness or susceptibilities. As a result, they can mark the violation and breakdown of security. Software architecture must be designed to gratify the requirements of product security goals, and sensitive information contained within. Hence, in the suggested work the emphasis was on software weakness (design flaws) which will later develop into susceptibilities. In this scenario, scholars have suggested procedures and methods to identify and precise design flaws with different algorithm, metrics and refactoring techniques. Still, prevailing procedures and methods have several inadequacies. Processes were normally based on manual inspections and henceforth, they were not being scaled to large programs easily. Also, these recognition techniques mostly use software metrics, which cannot reflect the all properties of design. In particular, the
inaccuracy of the documented explanations hinder with the comprehending of the design flaws and the execution of recognition and amendment algorithms.

2.6 Motivation:

Number of security loopholes and vulnerabilities exists due to the defects of security architecture and security mechanism. Hackers and attackers do not create security loopholes; rather they target the weaknesses in the software and exploit them. In order to maintain the software security during the software developmental stages, hacking should be made too difficult (J.Viega and G. McGraw, 2001). The object oriented approach naturally lends itself to an early assessment and evaluation. Object oriented methodologies require significant effort early in the development cycle to identify object and classes, attributes and operations, and relationships. Encapsulation, inheritance, and polymorphism require designer to carefully structure the design and consider the interaction between objects. The result of this early analysis and design process was blueprint for implementation. Therefore, the approaches provide the information needed to assess the security of the design’s classes, structure, and relationships before they were committed to an implementation (S. P. Kadam et al, 2011). Analysis of the structure of relationships depicts the robustness of design decisions. Hence, there appears a need to develop a mechanism to quantify object oriented software security early in the development life cycle (S Chandra and R A Khan, 2008).

In this research work, the aim is find the weakness that will turn out to be vulnerabilities that become basic indicators for the level of trustworthiness of the software. In order to achieve this goal quantification of security at design stage is necessary. Consider security in design phase, quantitative estimation of security is required. Quantitative estimation of quality had been mention in many studies (Jagdish Bansiya, 2002). Security is one of the major factors of quality therefore; there is a possibility to quantify security based on quality quantification methodologies for object oriented software. To overcome these barriers in this research work an attempt was made to investigated the impact of all design property on the security and found a relationship between design properties and security attributes. Develop the security model for object oriented design (SMOOD) by weighting the relationships. In this way, it is possible to check the security in software
architecture level. It allows the designer to do the necessary action to deliver the software that is secure by design. The advantages of proposed work were to detect and remove defects at earlier stage, which in turn reduce development time and cost.

2.7 Discussion:
The primary objective of the process of software development was to dispense products with high quality in addition to them being accurate, steady and reliable. This aim has been motivating the researchers to explore many methodologies supporting the powerful design and analysis for systems at the software architecture design level. This helps in early detection and correction of defects in the software which further reduces the development time as well as the cost of the system.