CHAPTER 1

INTRODUCTION AND LITERATURE REVIEW

1.1 NEED FOR RISK ANALYSIS IN SOFTWARE DEVELOPMENT

Risk analysis is a process that involves mitigation of errors during various processes of software development. Early identification of risks should shorten the software development completion time. It will also provide efficient solutions. A complete risk analysis along with identification, monitoring and management plan is sure to ensure high quality software products.

1.1.1 Concept of Software Engineering

Software Engineering is a systematic approach to development, operation, maintenance and upgradation of software. In general, Software Engineering is a discipline which provides tools and techniques to develop quality software in an orderly fashion. It provides an engineering approach that is used to build software for end users using well defined, well managed, consistent and cost effective processes.

The Software Development Life Cycle (SDLC) is the sequence of different activities that take place during the development. Any software development consists of five phases namely Requirement Analysis, Design, Implementation, Testing and Maintenance (Pressman 2000).
The SDLC begins with the identification of the requirements of a software and ends with the formal justification for the development of software against that requirement. SDLC is the period of time that starts, when a software product is conceived and ends when the product becomes dormant. Traditionally, the models used for SDLC have been insisting on sequential development, progressing through a number of well defined phases. Spiral model (Boehm 1988) is one of the such models, which reduces the risk from medium level to low level, using strong decision support system. The spiral model encompasses the strength of the Waterfall model which includes risk analysis, risk management, risk support and management processes. Spiral model allows the development of the product to be performed using a Prototyping technique or Rapid Application Development through the use of fourth generation language and development tools. It allows the feedback from users, to reduce risks by using reusable capabilities, and other modifications to ensure an acceptable product. It reflects the underlying concept that each cycle involves a progression that addresses the same sequence of steps as that of the Waterfall process model. This is done for each portion of the product and for each level of the complexity of each individual program. Figure 1.1 shows the risk analysis that is involved in the spiral model.

The reason for choosing the spiral model in this research work is due to the fact that it includes risk analysis at each iteration of software development phase and also identifies and resolves the risk at every phase.

**Determining objective, alternatives and constraints for risk:** In the first quadrant of activities, the functional performance, ability to accommodate change, hardware or software interface and critical success factors are identified. Constraints imposed on the application of alternatives in terms of cost, schedule, interface and environmental limitations are examined. The risks are analyzed for lack of experience, obsolete technology, tight schedule and poor processes.
The alternatives need to be evaluated to identify and resolve risks in the second quadrant. Here the objective and constraints are evaluated and the identification and resolution of risks are also carried out.

The third quadrant consists of activities like design creation, review of design, development and inspection of code, testing and packaging of the product etc. The first step towards building software is to enhance the customer’s interest by presenting an attractive system. The second step is to respond to the customer’s reactions for subsequent phases and to produce a fully satisfactory system for the customer. The degree of change or
modification diminishes in each phase and eventually results in a complete operational system.

The fourth quadrant consists of typical activities like the development of different plans such as project, configuration management, testing, and development for installation.

For each iteration processes like, determining the objective, providing the alternatives and constraints, identifying and resolving the risks are performed. The evaluation of alternative and development of deliverables are carried out for every iteration and verified. If they are correct then the product is released otherwise it will be transferred to the next iteration (Boehm 1988). There is no definite number of iterations to deliver a product which depends upon the nature of the project.

A salient feature of this model is that the coding is adaptable; even it is transferred to some other model. The basic idea behind it is to achieve the minimum risks through successive refinement of user requirements. This spiral method emphasizes on the evaluation of alternatives and risk assessment. A review at the end of each phase ensures commitment to the next phase or if necessary, it identifies the need to rework a phase. The advantage of the spiral model lies in its analysis on procedures such as risk analysis, and its adaptability to different life cycle approaches.

The Requirement Analysis and Design phase could be completed using any of the SDLC model, based on the understandability of each user. But in case if implementation phase, the level of importance should be higher, when compared to other phases.

In SDLC, the implementation phase of coding is responsible for building the software. In the implementation phase, each component of the
software is coded and each unit is tested to verify whether it has implemented the detailed design or not. Testing phase ensures proper functioning of the software with the given data. A good code is one that works is bug-free, readable and maintainable. Global organizations have their own coding standards that all the developers must follow. There are some important qualities of a good code such as transparency, clarity and consistency. These make it easy for programmers to understand how a program works. Extensibility reuse and modifying the code without any risk castigation is also essential in programming.

Software maintenance is an activity that includes error corrections, enhancement of capabilities, deletion of obsolete capabilities and optimization. As changes are inevitable, mechanisms must be developed for evaluating, controlling and making modifications. Hence changes in software during its operation period may be considered after the reuse. The purpose of this phase is to preserve the value of the software over a long period of time. This value can be enhanced by expanding the customer base, meeting additional requirements, making it more users friendly and efficient software deploying newer technology. The categories of software maintenance are corrective, adaptive, perfection and preventive maintenance after reuse.

The advantages of using software engineering in SDLC are to improve the quality, reduce the cost, establish standard, improve productivity and increase reliability of software projects.

1.1.2 The Purpose of Risk Analysis and Management in Software Project

Software comes under various categories such as system software, development software and application software. In general, all these software are designed to assist in performing specific tasks. With technological
advancement, every field depends on the software product for its requirements and lot of new software are being developed and marketed every day. All these software require testing during their development stages and prior to delivery. The main reason for software risk analysis is to remove the errors, flaws, threats and vulnerabilities, if any, during execution of the software before releasing (Fairley 1994).

One of the aims of software development is to create artifacts of quality software. The quality of a software product has many aspects, one of which is correctness, which refers to the absence of defects. Software risk analysis checks whether the product will work for the designed situations or not. Risk analysis may incur costs lesser than in the phase of testing in software development. These errors can be identified and removed amicably (Charette1989). The most damaging errors are those, which are not identified during the development phase and remain when the system becomes active. The purposes of risk analysis are to identify the initial defects, reduce the development time, provide better customer service, build better applications and check whether the user requirements are satisfied or not. Risk analysis is also meant for making desired modifications and enhancements for later versions. This is done by identifying reusable modules and components for adaptability. Risk analysis of software project is also aimed at evaluating various attributes and capabilities of a program to ensure that it produces the required results without risk factors.

1.2 NEED FOR RISK ANALYSIS IN SOFTWARE PROJECT

Occurrences of errors during the execution of a program need to be predicted before implementation. This is generally done through Risk analysis. For instance, classifying data types with variables would reduce the risk of errors. Literatures have pointed out the requirements of risk analysis for these types of problems.
A risk is a potential problem which may or may not occur. Though software development is a very difficult process that determines risks and taking practical measures to avoid them, risk analysis becomes mandatory. In addition, to manage the risks, in order to reduce the complexity and the efforts taken for it, also becomes mandatory (Boehm 1991).

**Software Risks:**

The risk has two characteristics

1. Uncertainty: The risk may or may not occur.
2. Loss: Unwanted consequences or losses will occur if risk is present.

To quantify the levels of uncertainty and the degree of loss associated with the risk, the type of risks should be categorized as follows:

Risks which are external to program coding have been delimited to the study for this research. Four of these risks namely i) ‘People Risk’ would lead to reduction of efficiency, ii) ‘Project Risk’ is sensitive to project cost iii) ‘Technical Risk’ is related to the time of implementation and iv) ‘Business Risks’ influence the software viability. Hence these issues are considered for the research study.

Other categories of risks namely 1) Known risk: Risk which can be uncovered after careful valuation, ii) Predictable risk: Risk which can be felt from past experience and iii) Unpredictable risk: Risk which is difficult to identify in advance. All these are important.
1.2.1 Risk Analysis Process

Risk analysis is a process for identifying, estimating, refining, mitigating, monitoring and maintaining software risks. The process schema is shown in Figure 1.2.

![Risk Analysis Process](image)

Figure 1.2 Risk Analysis Process

**Risk Identification:** Risk identification is a systematic approach which specifies the threats to project plan. Risk identification is carried out in two phases, namely

- Generic risks- Potential threats to every Software Project.
- Product specific risks – Risks that can be identified by skilled persons.

Risk identification is considered to be of prime importance in the process of Risk analysis. This process is depicted in Figure 1.3.

![Risk Identification Process](image)

Figure 1.3 Risk Identification Process
**Risk Estimation (Projection):** Risk estimation activity attempts to rate the risks. The estimation may be done in two ways namely i) ‘Probability of risk’ and ii) ‘Consequences of problem associated with risk’. The risk estimation process has four different activities as shown in Figure 1.4.

![Risk Estimation Process Diagram](image)

**Figure 1.4 Risk Estimation Process**

The steps of Risk estimation process are to be considered for prioritization. The team can allocate resources where they will have the most impact of that prioritized risk.

**Risk Refinement:** Risk refinement is a task that state general risks into a set of more detailed risks in order to make the Mitigation, Monitoring and Management of Risk easier.

**Risk Mitigation, Monitoring and Management (RMMM):** RMMM plays a vital role in the proposed model of Risk analysis. It involves three components that are: Risk Refinement, Risk Mitigation, Risk Monitoring and Risk Management. These three components have been considered in the proposed risk analysis model.

**Risk Mitigation:** Risk mitigation attempts to avoid any risk before it occurs by developing efficient strategies and planning to predict the risk and taking necessary actions to avoid them.
**Risk Monitoring:** This activity attempts to monitor factors that might provide an indication to whether the risk would become more or less probable. Risk monitoring activity has three primary objectives: (i) assessing whether the predicted risks occur (ii) ensuring risk avoidance (iii) collecting information that can be used for future risk analysis.

**Risk Management:** Risk management assumes that mitigation efforts fail and the risks would become reality. In RMMM plan, all the works performed form a part of risk analysis and are used by the project manager as a part of the overall project plan. RMMM is a form of Risk Information Sheet (RIS). Once RMMM has been documented, risk mitigation and monitoring steps are commenced.

### 1.3 THE CONCEPT OF THREATS, VULNERABILITY AND SECURITY RISK

A general problem solving activity is used to find out an acceptable solution using various software engineering methodologies, experimentation and reuses pattern solutions.

When risks are analyzed, it is important to quantify the levels of uncertainty and the degree of loss associated with it. This thesis mainly deals with technical risks. Technical risks threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or even impossible (Whitman 2003)

Risk effective level is always in the form of Threat and Vulnerability and it is associated with risk analysis process i.e., Risk = Threats x Vulnerability.
Risk classification of effective level is shown in Figure 1.5. Risk analysis is the sequence of activities that help to protect the software system from threats. The overflow in a program leads to flaw or weakness in an information asset and it will breach the security. So overflow is also one of the causes for threat and vulnerability. A threat is an object or person or other entity that represents constant danger to an asset. Software kills itself as well as it paves a way to a hacker to attack the software. Strong supporting system model is thus required for avoiding the risk factors in a software project. This is the rational behind this research.

**Unsecure Model:** As long as Vulnerability and Threat loop hole exist in Software, they would become much benefitable to the attacker. In addition, the efficiency of software is decreased as well. Figure 1.6 shown below depicts an unsecured model.

---

**Figure 1.5 Risk Classification**

**Figure 1.6 Vulnerability and Threats in Unsecured Model**
**Secured Model:** Each and every phase poses a certain level of vulnerability and threat. Minimum threats and vulnerability ensures a secured product. Figure 1.7 shown below depicts a secured model.

![Diagram]

**Figure 1.7 Vulnerability and Threats in Secured Model**

**Generic Procedure to transform unsecured model into secured model:** If any Software (created) exhibits Vulnerabilities, it requires some sort of remedy that mitigates the level of risk. The programmers from different communities of Software development must work together to address all levels of risk, that ranges from disaster to smaller mistakes made by the developer. Figure 1.8 shows a flowchart, which depicts how an organization makes a secure model for the developer to avoid risk factors.
A development team for information security creates the ranked vulnerability and threats to choose one of the following four basic strategies to control the risk (Whitman and Mattord 2009).

Avoidance: Avoidance is the risk control methods to prevent the exploitation of vulnerability and threats. It also makes access limited to assert and add protective safe guards.
**Transference:** Transference is a control approach that attempts to shift the risk to other assets or other organizations.

**Acceptance:** Varying mitigation, acceptance of risk is the choice of protecting from the vulnerability and to accept the outcome of its exploitation.

**Mitigation:** Mitigation is the control strategy that attempts to reduce impact caused by the exploitation of vulnerability through planning and preparation.

### 1.4 CONCEPT OF REUSABILITY

A system is said to be reusable, if it has the capability of using its components in an orderly way. In short, a part or whole of one system can be incorporated to assemble a new system. As far as software is concerned, it makes the reusability simpler and competent to frame a new module which is free from bugs and risks (William Frakes and Carol Terry 1996). Table 1.1 depicts the merits, demerits and supported area involved in the concept of reusability.

<table>
<thead>
<tr>
<th>Merits of reusability</th>
<th>Increases adaptability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reduces process risk</td>
</tr>
<tr>
<td></td>
<td>Reduces time and cost</td>
</tr>
<tr>
<td>Demerits of reusability</td>
<td>Lacks in tool support</td>
</tr>
<tr>
<td></td>
<td>Creating and maintaining a component library which creates overhead</td>
</tr>
<tr>
<td></td>
<td>Adaptability in reusability is cumbersome</td>
</tr>
<tr>
<td>Supported area of reusability</td>
<td>Design pattern</td>
</tr>
<tr>
<td></td>
<td>Component based development</td>
</tr>
<tr>
<td></td>
<td>Application product lines</td>
</tr>
<tr>
<td></td>
<td>Application frame work</td>
</tr>
<tr>
<td></td>
<td>Server oriented system</td>
</tr>
<tr>
<td></td>
<td>Program libraries</td>
</tr>
<tr>
<td></td>
<td>Object oriented software development</td>
</tr>
</tbody>
</table>
1.5 LITERATURE REVIEW

Traditional, Procedure Oriented Program and Object Oriented Program are different from each other (Firesmith 1993 and Ryder et al 2005). Procedure oriented languages are algorithm-centric in which a program is driven by an algorithm that traces its execution from beginning to end. Data is an external entity that is operated upon by the algorithm. Fundamentally, these classes of programming languages are characterized by data that are considered and separated from the operations or program part. Algorithm being the driver, with data being subsidiary to the algorithm, the procedure oriented program breaks up a larger job into a number of subtasks and subprograms independently as functions and subroutines. These functions and subroutines are then combined to form a program. The general notion is to simplify the debugging process of a program and to increase the number of reusable procedures in programs. The importance of program reuse is thus realized by software professionals. By reusing, the cost of building software would be considerably reduced. This has led to the emergence of Object Oriented Language. Khan Al-A’ali and Girgis (1995) have described the Object oriented programming language to be data or Object centric.

Object Oriented Programs promote objects that help to identify the relationship between data and object that improves creativity in reusability. When common data and objects are used in different programs, it will reduce the development time for programmers and increase the efficiency of the program. Reusable components are chosen in such a way that they will be suitable for the activities in the implementation phase, to reduce risk. The implementation of code reusable components is mainly focused in this thesis.
Nunamaker et al (1989) suggested approaches to understand software development processes and improve software productivity including the using and designing of automated software development tools. Studying human factors in software development and applying software productivity measurement and evaluation technique are other processes. An environment to facilitate the reuse of software components is described. Such an environment supports the identification, creation, categorization, storage, selection, usage, and maintenance of reusable components. A meta system environment that allow users to define functionalities, structures, and constraints of various software components is discussed. Information about these components is used by a knowledge-based system to support the selection, configuration, and distribution of reusable components.

Ramamoorthy et al (1993) stated that the overview of risk management for reusability of software development is based on two knowledge tools namely risk assessment and consistency management. Risk assessment will help knowledge based systems that are involved in analysing all the metric data to high risk and low risk components. Influence Diagram Based Expect System (IDES) estimates the expected loss of each risk item. The influence of problem and decision on it, rules and inference engine are solved by the system. The conflict resolution strategy is also provided by it. Consistency management system ensures proper operation of the system after correcting and measuring the relationship between the components in each phase. This approach is called Assumption–Based Truth Maintenance System (ATMS) which is used by any team for dependency relationship between the software object and data throughout the life cycle of the system development. This tool can help to prevent disaster and also to reduce the impact of the risks.
The method of Domain Analysis is to find similar patterns of software from development experiences to satisfy the current customer needs. Past experience guides to reuse the existing software products and improves the quality of the delivered product. The system analyst has to analyze the current problems of the customer and find the similarity of requirements from any completed projected requirement. Domains of products have to be identified for comparing the current problem domain to minimize the efforts for developing the system. This domain analysis technique was declared by Basili et al (1994).

Object Oriented Metrics are dependent on programs and are estimated through empirical study. Empirical study focuses on process improvement which increases necessity of using software measures and metrics. The need for such metrics was particularly notable when an organization was adopted for further enhancement in programs. An automated data collection tool was developed and implemented to collect an empirical sample of these metrics at two field sites in order to demonstrate their feasibility and suggested ways in which project managers may use these metrics for process improvement by the teams of Chidamber and Kemerer (1994).

Paul Raymond et al (1996) have discussed in their paper, the use of an object oriented data modeling approach for quality and risk requirements pertaining to software projects. The proposed approach provides an environment for users to develop individual views and allows the system to process semantically heterogeneous queries. Here a temporal modeling framework is used which allows users to capture temporal semantics of information associated with software metrics. Formalism for temporal modeling of metrics data using a set of temporal relation is also created. Further, an object oriented query model to receive semantics associated with
software quality and risk is obtained for modeling temporal semantics of software data. To express queries, the notion of scenario was also introduced where one scenario is identified through raw values or processed data. Temporal modeling is used to express complex scenario spanning various metrics and to help in identifying risk and also for evaluating the quality of the software project at various levels of abstraction.

The studies on examining how organizations actually exploit reuse technologies and evaluates the reuse factors are reported. These factors affect the rate of reuse in an organization so as to attempt to enhance the measurement of the rate of reuse. The effectiveness of reuse by establishing conceptual foundations in the literature for reuse and conducting an empirical investigation of organizations using Ada technology is reported. This study differentiated software reuse criteria such as domain, humans, tool, organization, software metrics, and environment. Nam Yong Lee and Litecky (1997) have shown that the rate of reuse significantly depends upon reuse capability, software development effort, object-oriented design capability, repository development effort, Ada technology capability, and domain capability.

David Kane et al (1997) and his team have discussed how to manage the changes in reusable software. Organization which undertake software reuse need to manage changes in component which grow due to expanded usage. Often, this is an expanded life cycle of software asset. In this report, six patterns to support reusable software are suggested. They are (i) develop a shared platform (ii) maintain the reuse platform identity (iii) integrate reuse and tie to the bottom line (iv) reuse code (v) treat reusable components like product and (vi) merge clones. Each of this pattern must be described with their problem statement and with their solution of reusable software.
Charette et al (1997) explained that United State Navy Software Maintenance Organization management has created a process improvement activity for upgrading the overall quality of internal software maintenance. It provides a simple consensus method of risk assessment and mitigation. This approach focuses on addressing the real risk rather than perceived ones. They state that there is a significant difference between maintenance oriented risk management and development oriented risk management. They have suggested specific risk management information as input to specific risk management process.

Gemmer Art (1997) has discussed on the need for monitoring the function behavior in risk management beyond process. The author has suggested coaching the functional behavior to developer in order to manage potential risks before program begins and to reduce communication risks. The feature is obtained by construction of an effective preprogram structure that will be able to integrate itself with earlier programs as a framework.

Judith Barnard(1998) has proposed new reusability metrics which may tend to eliminate the future risks of the developed object oriented software. The author puts forward the importance of the completion and specification of the requirement in a product. The author considers the coverage of the methods and classes, the access specifies of the classes and the maximum level of genericity of the component and its adaptability in other environments as the reusability metrics for object oriented software development. The experiments proved their suggested by involving the analysis of a reusable components from widespread classes of well known and accepted programming libraries. This selection and estimation of its reusability will reduce the maintenance effort and cost for error corrections during the latter phase.
Mili et al (1998) have proposed a concept for analyzing and finding the libraries on component development software system. The criteria for reusing a library must be satisfied for improving the system performance that will reduce the development risks. The existing solutions can be used for reducing the storage complexity and performance statistics. The proposed methods are for retrieval of reused libraries to reduce the development time as well as improving the quality of the product.

Sen Tarng Lai and Chein Chiao Yang (1998) have suggested software metrics that can measure the quality characteristics of a Reusable Software Component (RSC). However, individual software metric cannot measure the overall quality characteristic of the RSC. Therefore, the software metrics shall be combined so that the conflict situations in metric combination might be reduced. The approach is a rule-based metric combination model. This is based on dynamically weighted linear combinations and conflicts reducing production rules. Applying this model, a highly flexible and practical metric combination approach can be created. The conflict situations in metric combination can also be reduced. An RSC extraction tool can thus be generated.

Fenton Norman and Neil Martin (1999) have suggested decision support which has risk analysis potential. The major focus of their work is to apply software metrics for defect modeling and resource modeling predictions. This explicit model performs on ignorance as well as cause and effect relationship. It explicitly places those assumptions that were previously hidden and provides visibility and audit ability to the decision making process. The Bayesian belief nets models are metrics based risk management decision support tools that build on the relatively simple metrics. These tools combine different aspects of software development and testing to many kinds of predictions, assessments and trade offs during one software life cycle.
Tockey (1999) gave recommendations that a set of skills and knowledge that could serve to differentiate proficiencies for software industry. Computer science and software engineering are related subjects through their definitions. This paper has described and recommended several skill and knowledge as quoted below 1) Computer science and Discrete Mathematics provide the relevant theory of computing like Automata theory, Turing machine theory, Set theory, Predicate logic etc.,. Software practice would recommend code optimization, human computer interaction, debugging techniques, reuse techniques etc. 2)Software product deployment has recommended for system conversion techniques, customer support techniques, general technology transfer issues etc., Software engineering management has recommended that the risk assessment and risk management, alternative software life cycle, change of management, effective communication skill etc are needed. Any engineering discipline points to the economic concept for industry. The author has pointed many issues like profit-loss, time versus the funds, depreciation and decision making etc, which are linked to risk and uncertainty. The author has also suggested a standardized curriculum for software engineering degree which should include his pin pointed issues. This curriculum that produces graduates who would be highly valued with such skill and knowledge would be able to deliver better and should have substantial responsibility in software organization.

Souter and Pollock (1999) have discussed static analysis of object-oriented software. They have focused on addressing the new features of classes, inheritance, polymorphism and dynamic binding. According to them, the use of Object-Oriented Program design principles has created a new set of problems for software tool developers and compiler writers. Besides having to represent and analyze programs that exploit the new features of classes, inheritance, polymorphism and dynamic binding, the style used to develop
software applications and the units of analysis need to be changed. Object-Oriented program design promotes the reuse of code not only through inheritance and polymorphism, but also through building server classes which could be used by many different client classes. This paper demonstrates how exploiting the nature of object-oriented design principles can influence development of scalable static analyses. The algorithm presented describes computing definition-use information for a server class. This information may be useful for data flow testing and debugging. Static definition-use information was not only useful to optimize and parallelize compilers, but also for debuggers, software testing, editors, program integration and software maintenance.

Another work by Bowjarwah et al (2000) analyses and tests, the integration aspects of software components, particularly the object-oriented software. Most of the software was developed using object-based designs and object-oriented languages. The task of integrating the components becomes more crucial to the success of the software. Data flow analysis has been applied for testing procedural and object-oriented programs especially in Java programs.

Class libraries are generally designed with an emphasis on flexibility and extensibility. Applications used a library and exercised only part of the library's functionality. As a result, objects created by the application may contain unused members. The presented algorithm is specialized in a class hierarchy with respect to its usage in a program. The algorithm analyzed the member access patterns for variables, and created distinct classes for variables that access different members. This class hierarchy specialization algorithm reduced object size and execution time through reduced object creation/destruction time (Frank Tip and Sweeney 2000).
Ropponen Janne and Lyytinen Kalle (2000) in their paper have discussed software development risks, mitigation of risk components, requirement management and resources usages. The authors have suggested an analytical tool for risk management. They have stressed the need for risk management analysis. They have also studied Personal Management risks, enumerated their findings and suggested a process for managing the risks based on the experiences of individuals involved.

Object-Oriented measures predict classes having faults. Object-Oriented measures are used to find program complexity. A cognitive theory proposes a threshold that effects many object-oriented measures. If Object-Oriented classes complexity is below a threshold, easy understandability increases. It explains how to design Object-Oriented programs. This paper tested two C++ telecommunication systems (Benlarbi et al 2000).

The planned reuse of software artifacts can improve productivity and quality in software development. However, a planned reuse effort require substantial investments in the process and in the software repository. In case of failure of a reuse effort, the initial expenses may not be recovered. The likelihood of success may vary in the reuse strategy employed; hence, potential reuse adopters must be able to understand the reuse strategy alternatives and their implications. Rothenberger et al (2002) described the characteristics of systematic software reuse strategies. They evaluated how they contributed to a successful reuse program using survey data collected from seventy-one software development groups. They have reported on ‘Object patterns’ for the early reuse programs that recognized the importance of the technical and also the organizational aspects of reuse. They argue that software reuse can only succeed if non-technical issues are also considered. The results of the survey from reuse experts indicates that additional work is required with respect to the non-technical reuse issues.
Wang (2002) has reported from his analysis on Component-based software engineering. He has characterized two major activities, namely development for reuse, and development with reuse. The reusability metrics should be defined differently for these two. In “development for reuse”, the reusability is maximized to make the component of the product as reusable as possible in various contexts and purposes. In “development with reuse”, on the other hand, the components with lower cost and lower irrelevant reusability should be selected in order to minimize the cost and complexity of the system. He has provided static metrics for measuring reusability of components in the development, based on the internal structure and services provided by the components. For development with reuse, he has provided dynamic metrics for cost-benefit analysis to pick up the best components in terms of their reusability.

Janusz Gorki and Jakub Miler (2002) have reported on the concept of risk management in distributed software development. They have stressed on effective, continuous and open communication. The authors have assumed an infinite memory communication channel to record and store all information and prioritized the risks, based on their perception during project history. It can be divided risk assessment into three hierarchical layer as reporting analysis, identification and present a process of continuous risk assessment.

The component-based software engineering is to decrease development time and development costs of software systems, by reusing prefabricated building blocks. Merijn de Jonge (2003) has focused on software reuse within the implementation of such component-based applications and on the corresponding software development process. As it turns out, achieving effective reuse practice between the components of a
single application and between the components of multiple applications which has serious effects on the complexity of the software development process.

More success and failure factors in software reuse have been reported by Tim Menzies and Di Stefano (2003). These researchers have reported more from expect opinion and empirical data discrepancies and have commented on their methodology implication. It relates to the use of World Wide Web for simplifying this process. The important feature highlighted in this article is its simplicity to find unusual patterns quickly and confirm the automatically detected design or refute patterns.

Gillian et al (2004) have stated that a software security checklist and assessment tools should be incorporated into the life cycle process and integrated with a security risk assessment and mitigation tool. This tool is used for to planning. The design for security begins with elicitation and specification of security requirements. The prevention tool, explicitly represents risks. Mitigations are useful for risk reduction. Additionally, the mitigation can include security checklist for good direction in determining the scope and environment of security. This type of integration tool can reduce the overall risk of an organization and the security must be addressed in the life cycle to reduce vulnerabilities and unwanted exposure in the software system.

Roy (2004) has given an overview of the new risk management framework as pro-risk. His work has a given a framework that can be applied to either a small scale or quite a complex project. With management levels of data requirement, its framework focuses attention on primary project component. The business domain in which the project is created and the operational domain when the project is actually carried out start with some activities. Stakeholder identification and risk factor identification are used to minimize the factors through the clusters. It is a open system, allowing the
user to develop their own specialized models. To calibrate the cost perspective and schedule, the perspective of operating the organization, the domain of the project must be suitable. This allows the risk analysis to run in parallel with the conventional project management activities.

Kwak and Stoddard (2004) have discussed the lesson learnt from implementing the practices of project risk management during the process of software development. It is found out that 15 to 35 percent of the software projects were cancelled due to schedule slippages, cost overrunning and failure to meet the project goals. The main objective of ‘Team risk management’ is to share the risk responsibility and burden effectively to lower the risk of all entities involved. The major problem of risk management processes is that one or more process steps may be missed, weakening the implementation and indirect benefits of integration.

Jakub Miller and Janusz Gorski (2004) have presented a systematic approach to software risk identification, based on risk patterns. They have assumed an explicit modeling for the process and demonstrated it by using RUP (Rational Unified Process) as the reference model. They have also provided a framework for risk identification at various abstraction levels which provided the risk analyst a possibility to adjust and control the resource spent on risk identification. This approach is helpful to both methods of risk identification using the predefined risk checklist or brainstorming risks during controlling risk identification.

Software collaboration graphs for two Open Source software projects written in Java programming language and Kowari Metastore respectively were analyzed for fifteen-month periods of development by David Hyland-Wood et al (2005). Collaboration graphs were produced for the package, class and method levels. The collaboration graphs were found form networks which exhibited approximately scale-free properties at all the three
levels during each period analyzed. This finding supports the claims made by others, that software class collaboration graphs provide approximate scale-free networks and also give new insights that seem to retain the scale-free properties across different levels of granularity. This is done over the course of their development life cycle. Significant differences between the magnitudes of power law exponents identified and those found in C and C++ applications by previous researchers have been cited and discussed.

Software reuse in a product family approach is commonly thought to lead to fewer product problems. It also offers greater productivity and easier maintenance. However, little empirical data has been found to support this assumption recently. Nancy Bazilchuk and Parastoo Mohagheghi (2005) have done analysis on more than 13000 problem reports collected by the mobile phone company Ericsson in Grimstad and Norway which has shown that software reuse does result in significantly fewer problems and has better stability. The analysis showed that reused components had a lower defect density than non-reused ones. This defect density is calculated by dividing the number of defects by the number of lines of code. At the same time, however, the reused components had more defects with the highest level of severity than the total distribution. But after delivery, less of these defects can be found.

Misra et al (2005) have presented risk management activities that are conducted at the project level, process and product level. The findings will help software project managers to quickly grasp the fundamentals and methodology that they should use in their organization for specific projects. Focus should be on comparing approaches with respect to predefined set of evaluation criteria.

Paul Adams et al (2005) have elaborated on open source content management system, designed to run in conjunction with zone application
server for web-based system. Software components have been included for reuse of open source software, based on python programming language. The work concentrates on an adaptive reuse software system that is reused through adaptation through customization with modification parameters. It serves different purposes and may involve more extensive change and possible extensions for development support environments. The development action was for Libber software engineering and British computer society’s open source for collaborative activities. The work suggested problem based analysis and on experience phon plug-in system associated with the project and can be adapted for the benefit of reuse, more accessible to other developers. The approach was for reuse support, relevant to improvement of open source software engineering practices.

Anil Kumar Tripathi and Manjari Gupta (2006) have argued that a new approach of risk analysis in reuse oriented software development has its own assumptions that cause the possibility of risk. It is related to reuse oriented software developments that are creating reusable software items. It may also be related to developing new software by reusing these already existing reusable software items. Both these activities are imposed with the possibility of risk. Software development projects normally are based on various assumptions and decisions concerning different possibilities of reuse such as design pattern, framework, component-based development and creating reuse repository separately. Reuse oriented software development helps the software developers to identify risks with reuse repository. They can address them and control their occurrence even at an early stage of the project to ensure the successful completion of projects.

Philippe Lahire and Laurent Quintain (2006) have jointly introduced a new concept which explains a new approach to improve reusability in object oriented languages. The author is implemented along with the application
where the users want to extent their usage, to provide necessary guidance and controls to the user. This adapter is designed on the functionalities of the application called as concerns. In their proposed example, the author have demonstrated that their model has the capability of supporting five categories of adaptations. They are 1) Implementation of new interface.2) Fusion of methods. 3) Insertion or redefinition of methods.4) Insertion of new instance or class variable into classes.5) Interception of the accesses to the instance or class variables.

Sarala and Valli (2006) have presented algorithms for defect detection in object oriented program. They explain various defects that might occur in these types of programs, like logical error and also operator overloading. Through static analysis and compiler analysis on the algorithms automatically, the common defects are formulated. These perform the function of identifying the error and reducing the burden of error detection.

Robert Schaefer (2006) has discussed critical programming search for professionals for validity, trust and expertise of a senior programmer. He has also provided the necessary requirements that must be possessed when programming and the need to develop others with the same required skills.

Andrea Capiluppi et al (2007) have suggested coding standards in software developments in order to produce high quality software artifacts. It has long been recognized as the best practice in traditional software engineering in a distributed heterogeneous development environment, such as that are found in the Open Source paradigms, coding standards are informally shared and adhered by communities of loosely coupled developers.

Han Van Loon et al (2007) in their paper, explain the various process assessment standards for assessing the capability of due contractor process. They explain the Capability Maturity Model (CMM) used for
benchmarking standard and also describe various requirements and processes
to reach the CMM levels. The team states a new method called practical
process profile methodology through which the author presents a method for
specifying and improving business process efficiency, reducing risk and
increasing safety integrity.

Chin-En Lin et al (2007) have discussed an agent oriented (based)
system which is focused to investigate both objected oriented and agent
oriented techniques. Through this paper, they have found well tested object
oriented methods that can make the agent software process more adaptable to
a majority of systems. However, some differences exist between object and
agent autonomous. It should be self contained and should act to achieve goal
without external influence or initiation. This drawback may be overcome
through the frame work for evaluating the methodologies, comparing
strength, weakness and identification and to improve through a particular
methodology. Five Conditions on the methodology are presented: 1. Concept
properties form base building blocks of agents 2. Notations and modeling
techniques are the key to represent elements and activities in a system 3.
Process serves as a detailed guideline for all actives throughout the
subsequent phases 4. Dramatics provide real use scenarios as developers
apply methodology in building agents based system. 5. Final observation is a
good mechanism to support an agent autonomy, adaptation and collaboration
in partial conceptual models.

The software firms generally implement program reuse, in their
Information System developments in order to save development costs and
gain enhancement of knowledge and improvement in quality products. The
contracts generate incentives, career promises and reuse policies for
motivating developers. Research is scant in the areas of component reuse in
open source software developments where the reuse of programs is generally
absent. Using the theory on knowledge transfer, empirical research on software reuse in open source software development has been done. Developing a lightweight model of component reuse in open source software or substituting elements of software reuse programs in Information Security development for improving the adaptability of reuse component were established. This model was suggested by Sebastian Spaeth et al (2008).

Analysis and measure of software reusability of objects oriented design using the class diagram has been reported by Pradeep Kumar Bhatia and Rajbeer Mann(2008). They have investigated static analysis for finding the reusability of object oriented class diagram, based on the CK metrics, such as Weighted Methods per Class (WMC). The weights are: Number of Children (NOC), Depth of Inheritance Tree of a Class (DIT), Coupling Between Objects (CBO), Response For a Class (RFC), and Lack Of Cohesion in Methods (LOCM). The report presents five object oriented programs and calculates the values of CK metrics for each program to find out the measure reusability of various class diagram.

From the literature review presented above, it is clearly demonstrated that there is no model currently available for risk adaptable analysis of reusability transformation in object oriented programs. Hence a model has been proposed in this thesis.

1.6 PROBLEM IDENTIFICATION

Software must be designed and tested properly before it is delivered to customer without any risk and error. Risk analysis is a testing technique that is used to identify problem in the sequence of actions and recovers through reusability and also reduce the risk factors. This is done by the suggested Adaptive Reusability Risk Analysis (ARRA) model. The ARRA model mainly consists of stages such as Assay of reusability transformation,
Adaptability Risk analysis, Risk identification, Impact level of enervative and destructive risks. Risk level planning methodology involves, risk reduction and Recognition of reusability transformation. This may be implemented as a tool in C++ and JAVA programs. The role of this proposed model is clearly demonstrated through Figures 1.9 and 1.10, one for the C++ functions (Figure 1.9) and the other for the package of JAVA (Figure 1.10).

**Figure 1.9 Analysis for adaptability of C++ Functions**
**Figure 1.10 Analysis for Adaptability of JAVA Package**

There are two possibilities of adaptation. i) The existing for reuse C++ functions or JAVA package may be adaptable for reuse to the new function or package, as the changes required on them may be only a few. ii) The existing C++ functions or JAVA package may not be adaptable for reuse to the new functions or package, as the changes required on them may be large. The quantum of changes required for making them reusable depends upon several factors. These factors belong to two classes. One class belongs to the internal program coding. The other class belongs to the external software organizational set up. These factors are known as risk factors, as possibility for adaptability is more, if risks are less and vice versa. The present research has focused on the external risk factors only. It is evident from the vast literature that there is no such model currently existing for adaptive reusability risk analysis for reusability transformations in objects oriented programs and hence this innovative model has been proposed in this thesis. Any practical usability of this model for analysis need to be based on certain benchmark values. The research work also provides these benchmark values that have been derived through social study.
1.7 ORGANIZATION OF THE THESIS

The contents of different chapters in the thesis are briefly described below.

Chapter 1 presents the research objectives and the literature review. The need for risk analysis in software project and the concept of reusability is projected. The emergence of this research, on Adaptive Reusability of Risk Analysis (ARRA) is established. Besides, the organization of the thesis work in the subsequent chapters is briefly outlined.

In chapter 2, the source code of object oriented programs for finding number of lines of coding, program volume, number of errors, risk probability and the dictatorial risk factor for software project coding, human programming skill and involvement of interest, identifying the suitable technical people involved in the project code of object oriented programming through Venn diagram concept of set relation are analyzed.

In chapter 3, the various phases involved in the software development process, highlighting the concepts of reusability and the importance of reusability in any software development are analyzed. The need for reusability in risk reduction process is also emphasized and how it helps to have an error free system is explained. Major risk factor is directly proportional to the complexity of a system requirement, and it is inversely proportional to the number of reusable components in a software project.

Chapter 4 explains the innovative ARRA model that is developed for reusability checking of risk factor in Assay reusability of transformation, Adaptability of risk analysis, Risk identification, Analyzing the impact level of enervative and destructive risks, Risk reduction checking and Recognition of transformation. This chapter also elaborates how the sequences of the
ARRA model are proved through set theory analysis for transformation of reusable elements. This model can be used to analyze object oriented programs based on the sequence of processes. The practical applicability is based on certain benchmark values. This chapter also presents the design values of these benchmark.

In chapter 5, the proposed ARRA model has been developed for adaptability test, related to risks of object oriented program. This is to ensure that the extracted member functions from one module are free from risks to develop a new module. This is done with the basic idea of Inside-Out and Outside-In adaptable analysis using set theory. The relations of One-To-One and On-To function representation are explained.

In chapter 6, the ARRA model has been implemented in JAVA package to prove that the adaptability of a package from one module is free from risk to develop a new module. In JAVA, the domain and size of package, integration and dependency relationship of package, measuring the coupling and cohesion in risk metric for development and implementation of reusability package without any risk factor are performed.

In chapter 7, the conclusions derived out of the research work are presented along with suggestions for future work. The important contribution of the research is also highlighted.