Literature review gives attention on software development methodology, reusability, reengineering, domain engineering, frame working, reusable component repositories and searching and storing methods of software reusable components. If we are talking about reusability, which is not a very old subject to discuss, started in 1969. Doug McIlroy, firstly introduced the thoughts of systematic reuse of software components in 1968 [MCI1968]. After that many software companies such as IBM, HP, Dell, Hitachi and many others, reported successful reuse programs globally [GRI1993]. Many reports show that reusability truly works, enhances productivity, decreases development time and reduces the cost. Reuse is an umbrella concept, encompassing a variety of approaches and situations [MOR2002]. The reusable assets or components are in several forms like component repository, various modules in a domain-specific framework, COTS (Commercial-Off-The-Shelf) components, software architectures and their components forming a complete product family.

Davis et al. [DAV1988] mentioned about a framework that can be used as a foundation for evaluating the similarities and dissimilarities among different life-cycle models; as an instrument for researchers of software engineering to define the feasible ways of a life-cycle model and as a means to support software specialists and practitioners for choosing a proper life-cycle model to employ on a specific project or in a precise development area.

Ulrich et al. [ULR1990] discussed that reengineering includes code restructuring, code splitting, code re-aggregation and language level upgrades. Reengineering means up-gradation of existing system to fit in new environment with normal maintenance. First step in reengineering is reverse engineering. In reverse engineering, software engineers moves from code to higher levels of abstractions. We can create specifications
from code. Then these specifications can be used as input for next levels in reengineering process.

*Frakes et al. [FRA1995]* have investigated 16 questions about software reuse using a survey in 29 organizations in 1991-1992. They report that most software engineers prefer to reuse rather than to build from scratch. They also did not find any evidence that use of certain programming languages, CASE tools or software repositories promote reuse. On the other hand, reuse education and a software process that promotes reuse have positive impact on reuse. They also found that the telecom industry has higher levels of reuse than some other fields.

*Karlsson [KAR1995]* given a good overview of all aspects of software reuse such as organizational aspects, metrics for measuring reuse, development for and with reuse, managing a reuse repository, the clean room adaptation, object-oriented design for reuse and documenting reuse.

*Jacobson et al. [JAC1997]* described the reuse-driven software engineering to manage business, architecture, process and organization for large-scale software reuse. They mainly focus on architecture of the software and three discrete activities of component system engineering, application system engineering, and application family engineering. Notations are UML-based, with use cases to specify both the super-ordinate and subordinate component systems. One of the master keys of components-based development's success is standardization of infrastructure for every component. Also, infrastructures need three main elements:

- First, a uniform design notation which provides a standardized way of describing software component’s properties and functions.
- Second, a common interface for components-Based Development interface is highly recommended.
- Third, repositories are needed for cataloging of available software components with their descriptive features.
Nada N. [NAD1997] postulated the consequence of reuse on software quality (number of errors) and on software development timetables by cost/productivity model. Results recommend that trade-offs can happen between the percentage of reuse and the budgets of emerging and using reusable software components.

Alessandro et al. [ALE2003] proposed an iterative reengineering model. They suggested that the reengineering work should be iteratively executed on various components in different phases. During the process there may be coexistence of existing components, reengineered components and new designed components. This model improves the reengineering process with some limitations. Frequent reengineering components may cause resource wastage and increase the complexity. Format change between existing and new system may affect the performance.

Bush D. et al. [BUS2003] presented an approach to access requirement stability using scenarios. To perform requirement stability analysis, first of all a list of scenarios and a list of goals are developed. The scenarios are then analyzed for stability against goal list.

Jovanovich and Dogsa D. [JOV2003] described a different ideology and highlight the contrast of software development models into various phases. In first phase, the presentation of development models was described, in second phase they acquainted with a practical methodology to deploy one of the software development model. Lastly, the challenge was of defining the most appropriate software development model in the case of evolving PC based software applications.

Haddad et al. [HAD2006] mentioned that software organizations have to invest large amount of money to start successful reuse methodology and it is a barrier for them. Also core of reuse is source code. The authors estimated that domain specific components represent up to 65% of the application size. One effective approach to reuse practices focuses on domain specific components.
Chapter 2

Zhang Sen and Yao Zheng [ZHA2007] studied on the relation of CMM and SDLC models from integrity aspect, and the comparison between these models. The understanding of the relations between CMM and SDLC models are the basis on which a software organization makes macro-decisions regarding its software developing activities. So, the relations between CMM and SDLC models will be investigated in the following three aspects: (1), Relations between the two in terms of the history of development; (2), Comparisons of their focuses in the software engineering field; and (3), Mutual influences on their contents.

Mnkandla E. [MNK2009] focused on various software engineering frameworks and methodologies. His aim was to improve understanding of frameworks and methodologies for improving the ways of design and development of new software information system. Different available frameworks and methodologies are analyzed revealing their similarities and differences. A framework gives the guidelines about the way by which things should be done. Methodologies give more specific information about the way how activities are arranged in various phases. Conclusion is that better understanding of frameworks and methodologies results in development of better system.

Laura C. Rodríguez et al. [LAU2009] focused on various lifecycles frameworks models like Spiral, Waterfall, RAD etc. and comprehensive SDLC process like unified process, MBASE, TSP etc. This study concludes that despite of the criticism on the Waterfall model, which is the bases for several subsequent models and is the first to suggest the iteration and feedback issues; it is also possible to elaborate a systematic description/comparison of PM-SDLCs by using a meta-model.

Muzaffar Iqbal and Muhammad Rizwan [MUZ2009] proposed that there is constant need to enhance software quality which tends the commitment of software project managers all over the software development life cycle, so it is essential to provide them a list of predefined activities as per Waterfall Model, which should be made more efficient for senior
developers to get the productivity (output) up to 80% very easily. After applying the idea of 80/20 rule (Pareto Principle, 1935) the results of their study is offering the software professionals to adopt their approach by focusing on the list of critical activities of the Waterfall Model, which are giving the 80 percent output (productivity) and performance of the software process, as the effort is reduced and performance has been increased. So, finally software process has been improved.

*Sharma B. and Sharma N. [SHA2009]* had given the comparative analysis & investigation of software process enhancement models. Software process enhancement is renowned as a significant slice of the software development life cycle. They analyze each model along with their importance and drawbacks, and brought the simulation of the existent models like Capability Maturity Model, ISO etc.

*Maglyas A. et al. [MAG2010]* the objective of their research was to relate two existing models of triumph estimate i.e. the Standish Group and McConnell models and to evaluate their métiers and demerits. The Standish Group has leaning to overemphasize the challenges in any project. McConnell envisages the successful projects pretty well but undervalues the percentage of unproductive projects. Also, they mentioned that the size and complication of software development projects are rising, at the same time proportion of prosperous projects is quiet low.

*Nabil Mohammed Ali Munassar et al. [MOH2010]* in their work compared the five model of software development designed between 1970 to 1999. These five models are Waterfall model, Spiral model, Iteration, V-shape and Extreme programming. They compared each model with advantages and disadvantages of it. In their work they determined that there are many existing models used for developing software components of different size projects. Each model has the tendencies to abolish the weakness & drawbacks of previous developed models. Among all of them Spiral model and Waterfall model are commonly used in application development. Therefore, to ensure model’s suitability and documentation
during software development they recommended for designing a model by simulating the advantages which are found in different models into various project in past.

Niranjan P. et al. [NIR2010] in their research mentioned that software reuse efficiency can be enhanced with less investment. Software reuse expenses are further bargained when reusable components are easy to discover, acclimate and combined with newly developed applications. For increasing software quality in the software development, reuse is the key factor. Also, they focused on the software tool implementation with a new integrated classification scheme to develop new build of the software component, which results improvement in effective software reuse repositories to smooth retrieval or recovery of software components as per user necessities.

Sandhu Parvinder S. et al. [SAN2010] performed a survey on software reusability. In their study they discussed the various models available for evaluation of software reusability of a component. According to this research work for categorizing and identifying the quality of reusable components, there are few metrics but the function that makes the uses of these metrics to find reusability of software components is still not clear. They suggested to define these metrics in early phases of the project so that it helps in reducing the rework by improving quality of reuse of the components which further improve the production & efficiency.

Cortellessa V. et al. [COR2011] coupled the performance analysis with software development life cycle. In the beginning, they explained all the phases of a software development life cycle in reference to waterfall model. After that they presented their views on software performance analysis integrated within a software development process (i.e. the Q-Model).

Panwar D. et al. [PAN2011] proposed a new method to find the maximum number of faults by analyzing reliability and reusability in component-based software. Among many factors two important ones are reliability and reusability that are analyzed in component based software
engineering. This analysis is done to reduce the cost and risk in Component Based Software Development process. A Numbers of faults are calculated using two methods. One is existing called as Halsted method and the other new proposed method. In both, the formulas are applied to existing components and numbers of faults are calculated. And then component is modified according to the requirements of the customer, again both the formulas are applied to find faults. It is found that new method finds more number of faults and hence early detection of errors are done which helps to make quality software.

Fazal-e-Amin et al. [FAZ2011] presented a mixed method for identifying factors affecting software reusability in a reuse intensive software development environment. In their research paper they illustrated the results of a mixed method study. In first part of the study, research is based on meetings & interviews with professionals & domain experts. In second part of study, a survey is conducted to evaluate the significance or importance of the various aspects. In third part of the study, all the standard procedures were followed and results are presented of both research activities. Software is reused to gain benefits of cost, effort and time. Software reuse results in better quality, productivity and resource efficiency. They have identified 9 factors affecting reusability. These factors are Understandability, Flexibility, Scope Coverage, Portability, Maintainability, Variability, Stability, Documentation and Usage history. Although only first 6 factors are included in survey. Last three factors are not included in the survey due to their subjective nature.

Kardile Vilas Vasantrao [KAR2011] in their research work mentioned that the software design methods are crucial because of today’s dynamic environment of software development organizations. WWW or Internet has given a stand for organizations to interconnect and communicate directly with their consumers and buddies. Their outcome is that a systematic look towards the impact of uncertainty at module level and pattern of software development approaches will be useful to describe and express configuration in a way that respective specific allocated
process reduces the causes of software failure. It is therefore crucial that the uncertainty is introduced in the introductory phase and tracked throughout the model study and its identification, characterization of all uncertainty sources are performed jointly by the modeler. Problem framing and identification of the objectives of the modeling study performed by project manager and stakeholders helps the developer in choosing the development approaches as per level of uncertainty.

Niranjan P. et al. [NIR2011] proposed an intelligent technique for component storage and retrieval. Reuse means the use of domain & engineering knowledge from present software component to develop new & latest systems with more quality & less cost. The most common reuse product is the source code. Design, documentation, architecture, test data, tool and requirement specification can also be reused. The biggest problem of software reusability is the ability to locate and retrieve the existing software components. To overcome this problem, they proposed a genetic algorithm based technique where 9 characteristics of a component are stored in the form of 0’s and 1’s in the form of a 36 bit string. The 9 attributes of the component to be searched are given as input to search algorithm. Most appropriate match is searched as output of the intelligent technique.

Taya Sanjana and Gupta Shaveta [TAY2011] their research study specified that different software development models have their specific merits & demerits. Timing is very crucial in software development. The requirement of systematizing the various activities improves with the growing operations of an organization. Therefore, they described the various fashions of increasing technical complexity of the systems joined with the need for repeatable and predictable process methodologies to found system development models. So, it is sensed that some standard approach is to be introduced in the industry so that transition from manual to computerized or automated system became easier and also, there should be a tradeoff between the development time & quality of the software product.
Kumar Sunil et al. [KUM2011] deals with the extraction of software components from the software design phase. Method is based on UML in different perspective from the existing one. According to paper “MDL is an unstructured representation of the information contained in the various UML diagrams like use case, class, sequence etc in textual format. Extracted information is mapped into database tables to form clusters. The process of clustering is to facilitate the retrieval of components so as to gain high precision and recall.” In present study they worked on use cases only which may further also be extended in context of other diagrams.

Kaur Vaneeta and Goel Shivani [KAU2011] discussed various assets of the component repository like component searching mechanisms and classifications such as enumerated, free text, faceted classifications, and attribute value. The retrieval system of free text classification is typically based upon a keyword search. In this type of searching technique, a user input the keywords to search and as a result a ranked list of documents is returned. Enumerated classification is a single dimension classification which uses a set of mutually exclusive classes. An example of this is the Dewey decimal system used to classify books in a library. Attribute value classification scheme uses a set of attributes to classify a component. Faceted classification relies on facets which are extracted by experts to describe features about components.

Nassif A. B. et al. [NAS2012] proposed a software effort estimation model in the early stages of the software life cycle using Cascade Correlation Neural Network (CCNN). As effort estimation is a crucial task for the successful completion of a software project; their model calculates the cost in the beginning based on use case diagrams. Several existing techniques of cost estimation such as COCOMO, FP etc are acknowledged in the study. The Cascade Correlation Neural Network model has three inputs. These inputs include software size, team productivity and project complexity. Software size is computed based on the use case point model. Team productivity factor is also calculated based on the use case point model with some modifications. Project
complexity, is calculated using a new method. To evaluate the proposed model, a multiple linear regression model was developed. The multiple linear regression and the CCNN models were trained using 168 projects and evaluated using 72 projects. They concluded that the proposed model can be used as an alternate method for effort estimation.

Bjørner D. [BJO2012] presented a survey on domain engineering. According to him “Software Engineering consists of three major phases: domain engineering in which a domain description is developed; requirement engineering in which a requirement prescription is developed related to the domain description; and software design in which a software specification is related to the former.” He also discussed various issues relating to domain engineering in context of implementing reusability of software systems.

Fazal-e-Amin et al. [FAZ2012] presented an evolutionary study of reusability in open source software jasmine and pbeans. Jasmin is an open source java assembler which converts ASCII description to their respective binary java class files. Pbeans is an open source that facilitates automatic relational mapping of java objects to database tables. Pbeans package has evolved from version 1.0 to version 2.0.2(10 versions). The number of classes evolved from 28 to 49, the number of methods increase from 161 to 341. Jasmin has evolved from version 1.0 to 2.4(6 versions). The number of classes evolved from 99 to 118, the number of methods increase from 618 to 792. The reusability attribute model is applied to several versions of jasmine and pbeans in order to find the reusability factor. The analysis aids in understanding the reusability and its attributes in the context of software evolution.

Hashimi Hussein et al. [HAS2012] proposed a new model that helps in assessing and managing the risks and the processes needed in each phase of the development life cycle. The success of software development projects that associated with the completion of the project is not the estimated time and cost but it mainly depends on how to manage the
risk. Risk management is defined as “How we can identify, assess, and prioritize risks and then deal with these risks in a proper way to avoid or mitigate them.” (ISO31000). The proposed wheel model is a novel view of both traditional V-Model and advanced V-Model to make efficient and effective improvements in software development. It includes all software development lifecycle phases, along with their associated testing phases found in traditional V-model, and maintenance phases found in advanced V-model. The proposed model is more flexible and streamlined than the V-Model and the advanced V-model. Here the main goal of wheel model was to know how the likelihood risk can be assessed through it, and be aware with risks that influence software development life cycle.

*Mumin Hicdurmaz et al. [MUM2012]* proposed a fuzzy multi criteria decision making approach for the selection of appropriate SDLC model. The study provides a view on considering various factors related with people, process and technology for selecting a particular SDLC model. There are several models available and there is no rule that a particular model is best suitable for all types of projects. They discussed five models in their work with advantages and disadvantages. They introduced a fuzzy method of considering various factors for the selection of a SDLC model.

*Khan P.M. et al. [KHA2012]* designed a decision support matrix for selecting SDLC. Their paper recognizes the various risk associated with wrong selection of SDLC for any particular software development and proposed a selection matrix for selecting a particular SDLC model among traditional and agile methods that is most suitable according to specific situation. Evaluation criteria are based on various factors such as business criticality, customer involvement through SDLC, requirements clarity, requirements volatility and availability of business users. Their study is a valuable contribution and prescriptive guidance for business critical software projects.

*Kumar Sumit et al. [KUM2012]* proposed a tool for calculating the reusability of an object oriented program. Software reuse increases the
productivity, quality and reduces the cost of the software development. It is necessary to calculate the reusability of the module in order to implement the reuse of module effectively. Many measuring reusability methods have been proposed for estimating the reusability of module. They concentrate on the problem of finding more suitable module in old software. Module can be reused either without modification or after modifying it. Reusability reduces the development efforts, cost and improves the quality. The work presented by them can be used to calculate the reusability of any object oriented software module.

Subedha V. et al. [SUB2012] in their paper proposed a process model for context specific reusability. Context-specific reuse is a way to increase the value of reuse. They suggested process model which creates the quality assessment of the reusable software component for better reusability and helps to adopt a technique to define the reusability factor. As the quality of a reuse system is exceedingly reliant on the quality of the component reused due to that quality analysis becomes very critical. The process model offered by them associate three different quality aspects into reusability metrics. The main aim of this work is to reduce the effort spent by the developer to identify the component candidates for quality reusability.

Wang Chengjion [WAN2012] proposed pattern oriented reusable software component development approach that results in a component model. Reusability results in reduced development cost and improved quality. Software reuse has been promoted by component-based development method. According to him “Traceability is a key factor for reusability of software components. Patterns are used to describe the development expertise. The traceability between different models at different abstract levels is created as the side effect of development process of the software components.” The authors concluded that in traditional development traceability is important only during maintenance phase but in reuse oriented development it is important to link every low level component to the higher level architecture, so that developers can trace the most suitable component.
Benitti Fabiane Barreto Vavassori et al. [BEN2013] highlighted the reuse of requirement specifications, leading to greater reuse of other artifacts such as design, code and tests. Their paper presents an approach to the requirements reuse, supported by a tool. The efficiency and effectiveness of the approach were evaluated using a quasi experiment in a university. They conducted a quantitative evaluation of the approach on an assessment of participant’s perceptions regarding the use of proposed approach and the computational tool. The results of the quasi-experiment indicate that the approach presented makes the activities of requirement elicitation and specification about 40% more efficient and effective in terms of the way they were previously conducted.

The approach is based on three pillars “(i) requirement writing patterns for structuring knowledge in order to assist reuse; (ii) Patterns catalog providing a mechanism to facilitate the selection of a pattern; and (iii) Traceability to identify new requirements from a requirement reused.” The results of this quasi-experiment indicate that the efficiency and effectiveness of the approach is higher.”

Annaiahshetty Kalashankar et al. [ANN2013] attempted to design and develop an expert system to assist the software developers in the complete software development life cycle with multiple domain experts such as Telecom, Banking, Insurance, Logistics, Healthcare, Satellite and many more knowledge acquisition. The goal of their research include knowledge acquisition specific to the problems of using multiple domain experts, design and development of a prototype expert system for software development, and validation of the prototype expert system.

**Software Component Classification and Retrieval Techniques:** Component classification methods are generally used to categorize the software components into various domains. At the time of storing a component in the repository, it’s preferred to store it under appropriate category, this help a lot later at the time of retrieval. But many times the same general-purpose functionalities also stored in more than one
cluster or domain. Retrieval techniques are used to find the appropriate component from the repository for existing requirements i.e. one to one matching of component specifications as per requirement. For efficient and convenient retrieval, appropriate component classification and precise component specifications are very useful. It is one of the major activities in software reuse. This also includes even repository management techniques. The existing retrieval and classification techniques are also discussed in this section. There are various classification and retrieval methods & techniques based on formal specifications, intelligent systems and natural keywords [MIL1994] [SUS1987] [TAN2003].

**Faceted Classifications:** For cataloging components in software repositories Prieto-Diaz proposed faceted classification scheme [PRI1987]. In this scheme, some attribute called facets which are used as descriptors of a software component. These descriptors mainly concentrate on the action that component performs and manipulate by the component on objects. The Faceted scheme mainly consists of four facets - Object manipulated by the function, function performed by the component, system to which the function belongs and data structure where the function takes place.

The facet classification has the capability to explore domain understanding with the use of terms extracted from fix number of mutually exclusive facets. Each faceted scheme may have several facets, which also consist of numerous terms. The similarities between the components are considered using the conceptual distance between the facets. While retrieving a component, in query the value of each facet is given and it’s matched to the facets of stored component. After that, the components having the threshold value of similarity are retrieved. Facet categorization is beneficial in domain specific applications. The major disadvantage of facet categorization is that it requires manual indexing which makes it very expensive [KOT2003].
The method of faceted classification and retrieval is most extensive [HAI2004] [YUA2002]. A term is used into stated language context and is classified by specific angle of view (is called facet) which reflect essential characteristic of a reusable software component in faceted classification [GIB2000] [YUA2002], a facet is a basic characteristic which is described in a domain. A reusable software component is classified by each facet from different profiles, a software component can be described by many facets and many terms in a facet, different facet can describe a component from different angle of views. There are a set of terms in a facet, structured term space is formed by common and special relation. The value of a term can be only attained from given facet. It is helpful to understand correlative domain for the reused that travel in term space, the term space can be evolved.

The method of faceted classification is most accurate to express information of a reusable software component and can be easily understood by users in various methods of software component retrieval, therefore, if the method of faceted classification can be provided in some software component meta-data and component repositories which include many methods of software component retrieval, then it will achieve the best effect that the method of faceted classification is used [YUA2002]. But the type of software components and the requirement of organizations and user are different; the models of faceted classification are different too.

In other words, the condition of retrieval for target software component is quite other, a user wish search appropriate software components from a software component repository, the model of faceted classification must be understood and the condition of retrieval must be constructed, these manmade and subjective factors lead to the retrieval precision is low, when the main information of software component retrieval is provided, a user must make the most of generic terms or accepted terms. The metadata repository integrates expert knowledge of correlative domains and generalizes crucial concepts and relations among concepts in these
domains [LIN2007] [ZIP2007]. These query terms which are formed in virtue of metadata knowledge can improve the software component retrieval precision.

**Automatic Indexing:** Classification of software components is done by indexing the software component according to keywords extracted from the natural language documentation of software components. Free text indexing systems extract keywords from user queries in natural language and these keywords are used to locate documents. Similarly, software components are classified in a software repository by indexing them according to keywords extracted from the natural language. The system extracts lexical, syntactic and semantic information from the documents and this information is permanently stored as an index. Based upon the matching frequency of the keywords, mostly keywords are integrated into the index. In retrieval process, the keywords extracted from query should match with the index keywords. In information retrieval automatic indexing methods are of two types - Linguistic and statistical. [AND2006] [BOL1998] [SHA2005].

**Formal Specifications:** Formal specification languages are good for modeling. These are used for specifying the component, their states, invariant and operations. Mainly usage of formal specification languages are Z, Petri Net, Larch and Promela. The formal specification languages are used to represent the component and the query. The main disadvantage of this technique is that user must know the complex representations, as these formal specifications are highly syntax sensitive [SHA1995] [VAR1990]. Also formal specification matching techniques have chances to miss maximum nearly matched components [LAM2000][DIL1997] [GER1997] [YON1997].

**Signature Matching:** Selection of appropriate signature for multidimensional functionality in itself is a big problem [AMY1993]. In signature matching, type transformation and type matching are mainly used. Components are represented by a multi-set feature signature.
Signature matching approach assumes that the set of allowable signatures in the system is known. In functional matching the type of input parameters, the type of function, type of output parameters and local variables are matched. Feature of functional matching can be the following - constructed type, simple type, and user-defined type. The major difficulty in signature matching is extraction of signatures based upon functionality of the component.

**Behavior-Based Retrieval:** In this approach the execution of software components i.e. dynamic behavior is used to classify components in the repository. The components are classified in the order of their behavior. Behavior retrieval develops the executable of software components. Software programs are executed using components and their responses are stored.

Retrieval is achieved by selecting those components whose responses are closest to the pre-determined set of desired responses. Component behavior is testing the components under different constraints with various arguments yielding the dynamic responses. In behavioral retrieval technique dynamic behavior of the component is considered to be most important while creating the application. It's hard to locate those components which behave in the similar manner as the query components [HEN2001] [ROB1993] [YOE1991].

**Browsing:** In this approach the starting point is determined from the user query, to retrieve the most wanted component. The user navigates from the initial point. Browsing is viewed as a user navigating with help of graph where nodes represent library items and the arcs represents relationships. Browsing is considered to search where the goal is not well defined in advance; the goal is eventually dependent on what is discovered during the search. The main drawback of browsing is that it need user to derive the navigation process which becomes huge burden when component repository is large [SAL1989].
**Hypertext:** Components are prearranged as interconnected similar graphs in hypertext mechanism. In a graph, nodes are represented as components. The similar components have a connected edge between them. The user refers the graph to reach at the desired node which further represents the desired component. The semantic related with each node helps the user to navigate to the desired component. The main problem of this technique is that developing and maintaining software need for big efforts [CUT1999] [MOD2000].

**Fuzzy Logic Based Retrieval:** The software components are explained using the descriptors in fuzzy based retrieval methods. Software descriptor is a list of features describing the key characteristics of a software component. These descriptors are composed from term pairs, reflecting the functionality of component and are constructed from the code and accompanied documentation [ANS1998]. These features are in form of list and expressed by a weight i.e. a number between 0 and 1. The classification of software components is also done using a descriptor in fuzzy based retrieval methods.

The weights assigned to features are viewed as the fuzzy sets and, fuzzy techniques are used for retrieval. These features are extracted automatically. A feature weighing function is used to assign the weights to descriptors. Fuzzy matching between the software descriptors is used for computing the conceptual distance between the components. A preset threshold value is compared with compatibility value; if the value is greater than the threshold value then the component is retrieved. The similar value depends upon the weight of the common features, number and their importance [QIA2000] [RAO2003] [SHA1997].

**Genetic Algorithm Based Organization:** The basic idea of genetic algorithms (GA) is considered to simulate the processes in natural system for development, specifically which follow the ideology, first laid down by Charles Darwin of fittest survival. Genetic algorithms are good because
these consider large search space for optimal grouping and results which we might not find.

To solve a problem GA represent an intelligent development of an arbitrary search within a definite search space. The major benefit of the GA technique is the ease of using it with which it can handle arbitrary objectives and constraints. All such things can be taken care as weighted components of the fitness purpose, making it easy to adapt the GA scheduler for the particular requirements of a very wide range.

**Neural Network Based Organization:** Neural network are used to build the software library. The library is semantically organized i.e. similar to functionality of the stored component. Similar behavior components should be stored near to each other. The input data represents the features of the component to be stored. This input data is mapped to a grid of neurons. The output unit allocates the weight vector of n-dimensional. During the learning process the input vectors are frequently accessible to all output units. The outputs produced by these components are measured in terms of their similarity. Detailed functionality is described in a query for retrieving the component. This query in turn can easily be transformed into a feature vector, which is fired on to the grid, and the best-matching stored components are searched for. After overall operation, top matching unit and its adjacent components are retrieved [PRO2000].

**Probabilistic Retrieval:** In the earlier described retrieval models, a document is retrieved when it is in some way similar to a query. Probabilistic retrieval model intends to retrieve documents according to their probability of relevance to the query. More specifically, an individual can say that a document representation has a certain probability of relevance to the query. When the similarity between the document representation and query increases, the possibility of relevance also increases [FAB2003].
Reuse Libraries: In order to make efficient use of a repository or reuse library, an individual must have complete information of its contents. Any organization can achieve the actual benefits of software reuse if all components are stored in the repository in an organized and unified way [DAN2002] [VIT2003]. Reuse library contents can be represented in following two ways as described below.

**Horizontal Library:** Horizontal libraries consist of generic components. Generic components give general-purpose programming support on top of the programming language [CON1999]. Such components contain menu drivers, graphics utilities, abstract data types, and mathematical routines. The benefit of such library is that it is very useful across the application domains. In horizontal libraries, components serve to various application domains so their numbers are very large. It is very difficult to manage large repositories.

**Vertical Library:** It contains domain specific components [HAD2003][VIT2003]. The benefit of such library is that reusing its contents can reduce the amount of code. The downside of such library is that it can be used only for specific application domain. Vertical libraries are particularly useful for organizations that develop and keep large scale and long-lasting objects.

As per the literature survey, software reuse has been recognized as one of the most realistic and promising ways to improve software productivity, quality and reliability, reduce maintenance costs and the time required to release software for client. To maximize the benefits of reuse it is necessary to understand its importance and working properly. One more important point to be taken care of is the component you are going to reuse must address the user’s problem successfully. If it does not address the user’s problem then it is not reusable. Another constraint is that the component you want to reuse, if it cannot interface to existing systems, then it is not reusable. As Systems have to interface to an operating environment. Presently most systems interface to some subset
of graphics, networking and database subsystems. Therefore, the component you want to reuse must interface to the existing system. For reusing a component, the above-mentioned constraints must be properly taken care of, so that the benefits of reuse can be attained.

These constraints of software component reuse would pose many challenging problems for software reuse practitioners and researchers that do not exist otherwise in traditional software engineering. Therefore, software reuse requires careful planning, realistic expectations and a long-term perspective. Reusable assets can be executable code, source code, requirements documents, designs, test data, documentation etc. We need large software component repositories to support software component reuse. There is a need to come up with new effective mechanisms to manage software component repositories. An early contribution needed to begin a software reuse process but this contribution pays very soon for itself after a few reuses. In short, the advancement in a reuse technique and archive result are the bases of knowledge that improves in quality after each reuse, and thus require less efforts for coming projects, thus reducing the risk in coming projects that are based on archive knowledge.

As per literature survey, Software reuse library is a set of software components that use specialized methods for reusable components classification, storage, retrieval and delivering. Software components library played a significant role in recent software and system development. While libraries of Unified Modeling Language (UML) diagrams and source codes exist, one of the challenges that remain is to find appropriate designs and source codes, and acclimatize them to meet up the definite needs of the software developer. Solutions on how to represent, search, store and retrieve components and tools for all these activities are still being under development. Although reuse is a very good technique with many benefits but here are different reuse constraints.
These constraints must be handled properly to maximize the benefits of reuse. Like for the reuse purpose the software libraries i.e. code and domain libraries are needed to be maintained. Fact is that the libraries decay and they must be maintained properly. This decay causes due to changes in the version and configuration space. There are versions of the library, versions of the tools that compile the library regularly and versions of the environment. There are configurations of the library, configurations of the tools that support the library, and configurations of the environment. A new compiler and configuration might cause serious damage to an existing code library base. A solution to this problem is that a platform must be used that can handle the issue of code stability. Another major constraint is that before reusing something it is important to understand its functionality i.e. what its do and how exactly. The history of computing is driven by optimization. There is always desire for more to be done with current computing resources and this will always be the case in future as well. Reuse-oriented software development is a kind of software reuse approaches that makes the use of a software reuse library from which reusable components may be extracted. Software components library play a significant role in recent software developments.