Software reuse is the concept of developing new software by using existing software assets or components. Parts of some code that are in original form of a different project are normally identified for reuse. Code reuse is a concept in which a part or whole of the computer program can be used in other programs to save time, cost and energy. Software assets include all software products i.e. requirement specifications, designs, coding, algorithms, user manuals and test cases.

For successful reuse of code in systematic manner, organization must identify the good components, frameworks, software architectures and need at a proper time to design, implement, optimize, validate, apply, and maintain them. Developing reusable software assets need a mature construction where expert developers and architects can separate key sources of variability and common features in their application domain.

A good software reuse process results in growth of productivity, quality, reliability, performance and reduction in cost, effort, risk, implementation time etc.

1.1.1 Reusable Software Components

Reusable software components include not only the generic source code, but also other aspects of the software development life cycle including executable code, design structure, specifications and documentation.

Reusability is not a perception that is restricted to the coding phase of software development life cycle. It is necessary to discover the various life-cycle phases those could be helpful in reusability. An extensive approach for reuse should be accentuated. The various types of software components that can be reused are code, requirements, test strategies, architecture/design documentation, instruction manuals, technical specifications, design decisions, outlines i.e. templates, domain understanding, modules/data structures, i.e. models of knowledge domains, scientific laws, Environmental level information i.e. practical or experimental data or user’s responses & their feedbacks.
Chapter 1

The most common software reusable artifact is the source code. Nevertheless, it is not the only one reuse candidate; actually, other software engineering work products can also be reused. In this section, we review some of the common reusable artifacts [MIL1998] [MIL2002].

- **Executable Code:** It is usually represented in machine-readable form, and is indexed by means of its functionality.
- **Source Code:** It is the main candidate for reuse. Source code has both structural as well as functional details. It can be viewed as problem solving knowledge. It can be indexed either by its functional properties or by structural properties.
- **Requirements Specifications:** Code artifacts are executable but requirements are not. Requirements are recorded in form of natural language or in form of formal specifications or in mixed format. They represent basic functionalities of any software product. Specifications can be indexed by means of functionalities. Requirements specifications may be reused to construct either compound specifications or for variations/upgradations on the original product.
- **Designs:** Designs are generic representations of the problem solving knowledge. As compared to specification assets, designs capture structural information rather than functional information. Designs are represented by patterns. Features of the family of problems they intend to solve can index them.
- **Test Data:** For testing similar kind of products or to test the product after some maintenance operations, test data can be reused. Representation of test data is straightforward and depends upon the type of format user is following. Test data can be indexed either by description of input domain or by general indication of the functionality performed by the system.
- **Documentation:** Many reusable assets such as design, specifications etc are represented by natural language documentation. Documentation is most likely represented in natural
language and can be indexed by the asset that it documents or it can be indexed using hypertext.

- **Architectures:** Software architecture defines the structure of a software system. It is an aggregate set of components that exchange data. The architectures have a higher level of abstraction than programming language codes and are of different nature. They represent information flow, control flow or communication protocols between components. Architectures are represented by means of specialized notations and are indexed by means of their architectural features.

The important factors in reusable software are:

- **Documentation:** There must be adequate information about what each reusable software component can do.
- **Flexibility:** A truly reusable component must be usable in more than one context.
- **Visibility:** In order to reuse software, the user must be able to find it.
- **Efficiency:** A reusable component will not be used if it is not efficient enough for the job.

### 1.1.2 Software Reuse Categories

**Opportunistic Reuse:** In the beginning of any development process, the software team realizes that there are existing components that can be used. This is possible during programming when components are discovered that happened to fit a requirement [SHA2007] [BOE1999].

It can further be classified as

*Internal Reuse:* here developer team uses its own component.

*External Reuse:* here developer team may choose to buy a third party component.

**Planned or Systematic Reuse:** In planned reuse, a team strategically designs components so that they will be reusable in future projects. This requires a design process that considers how existing designs may be
reused and that explicitly organizes the design around available software components [SHA2007] [BOE1999].

**Black Box Reuse:** In Black Box reuse, the re-user sees the interface, not the implementation of the component. The interface contains public methods, user documentation, requirements and restrictions of the component. If a programmer wants to change the code of a black box component, compiling and linking the component would propagate the change to the applications that reuse the component. As the users of the component trust its interface, changes should not affect the logical behavior of the component. [SHA2007] [BOE1999].

**Glass box reuse:** In Glass Box Reuse, the inside of the box can be seen as well as the outside, but it is not possible to touch the inside. This solution has an advantage when compared to black box reuse, as the re-user can understand the box and its use better. The disadvantage is that it is possible that the re-user will rely on a particular way of implementation or other factors that are not in the contract. That can be risky when the implementation changes [SHA2007] [BOE1999].

**White Box Reuse:** In White Box Reuse, it is possible to see and change the inside of the box as well as its interface. A white box can share its internal structure or implementation with another box through inheritance or delegation. The new box can retain the reused box as such or it can change it. It is necessary to test anything new that is created or changed [SHA2007] [BOE1999].

### 1.1.3 Approaches and Levels of Reuse

There are mainly two approaches required to develop software:

- Software component development with reuse
- Software component development for reuse

**Software component development with reuse:** This approach tries to make best use of the existing software assets. An advantage of this approach is that the cost of development decreased drastically. Decrease
of cost is only one major benefit of software component reuse. Systematic reuse put forward various advantages such as enhanced system reliability, reduced risk, personified organizational standards, and saving of huge time during software development.

**Software component development for reuse:** When any software developed for reuse then its components must be general in name as well as in operation. They should not be specific to any application. The reusable component must be general in all respects.

- **Name generalization:** The name of the component should be tailored for its name neutralization with other application otherwise, a direct reflection of some existing application will be exposed.
- **Exception generalization:** It verifies each component to observe the exceptions that includes these components & passes through them in the component interface.
- **Operation generalization:** An operation is to be added or removed to the components that are very definite to some operation domain.

Levels of reuse can be performed at any level in the system. Reuse can be performed with whole application or by using a portion i.e. module or subsystem of the application. Major levels of reuse are:

- Application system reuse
- Module or object reuse
- Subsystem reuse
- Function reuse

### 1.1.4 Merits of Reuse

We can achieve the following goals if reuse is correctly implemented:

- **Increase productivity:** Reuse of existing software components help projects to decrease their development cost and maintenance after having some initial investment that results increase in productivity.
- **Shorten time to market:** Organizations reported that time to
market is significantly reduced by using reuse software, also shorten the decisive path in delivering a product.

- **Enhance software quality:** It is very well known & practically proved that software that has been deployed & tested many times, will give less error than newly developed software components.

- **Provide interoperability and consistency across products:** Commonly used components with similar or standard interfaces across products assist the ease of employ and interoperability.

- **Ensure product conformance with user requirements through prototyping:** Availability of reusable components assist prototyping; user’s needs may be more easily authenticated. This prototyping will facilitate discovery and resolution of errors at early stages in the software life cycle.

- **Reduce risk:** Risk is automatically reduced while developing new software components by employing reusable components, because reusable components encompasses the desired functionality and standard interfaces for proper integration.

- **Improve functionality and/or performance:** Performance & functionality improves by using reusable components with given good time investment. This investment of time is more cost-effective and justified than the case where they would only be for single product.

In practice, radical gains in productivity and quality cannot be accomplished due to some presumptions made by developers.

### 1.1.5 Demerits of Reuse

Reuse involves some demerits too, these are:

- For reusing code and design, effectively there is no formal training.

- Adopted software development methodology does not support software reuse. Useful reusable artifacts are not supported on the preferred development platform.
- Reusing code is quite boring, as compared with development of entirely new system.

- There are no standard tools to support developers to discover reusable artifacts. General tools are tough to practice & learn, but dedicated tools are less likely to meet user's needs. In a giant library of tools, sophisticated tools are required to search for the accurate tools.

- In most of the cases, developed programs are too specialized for reuse and the reuse process is very slow.

- Most important the NIH factor i.e. Not Invented Here factor (Locally developed code is better than that developed somewhere else).

### 1.1.6 Limitations of Software Reuse System

Two main technical problems currently limit the practice of software reuse [MIL1998] [DAV1998].

- A lack of mechanism to produce robust adaptive reusable software components.

- A lack of mechanisms to retrieve, adapt and compose software components effectively according to user requirements.

### 1.2 Software Life Cycle Models

A lot of research has been reported on the evolution of software development life cycle (SDLC) models. Research has been reported regarding the suitability, reliability, flexibility, efficiency of different SDLC models and comparison of different SDLC models. Some research has been reported on the relationship between project categories and SDLC models (SDLC) [LAU2009] [SHA2009]. However, no empirical study has been reported regarding the importance of the factors affecting the choice of SDLC models in the software industry. However, there is no universal SDLC model, which is considered adequate in all situations in the software development environment [MAH2012] [TAY2011].

Software development plan driven approaches like the waterfall model assume that requirements are static. Other iterative methods like spiral model and evolutionary model count on change. Agile methodologies
consider software development as an empirical process and that people play the most important role in it [LEO2008]. The fundamental principle of new software development engineering is to design and implement software products and modules that minimize the intellectual distance between the problem and solution. Today methodical approaches to software design and development have evolved and design notations have proliferated [MAG2010] [PRE2004] [JAL2005].

Life cycle models for software development provide the basic guidelines for developing software using engineering technique. The first task of a software life cycle model is to determine the sequence of stages in software development and to establish the transition criteria for progression from one stage to the next. There are several life cycle models and many companies adopt their own models, but all have very similar patterns.

All steps taken together in the successful development and deployment of computer software are referred to as the software life cycle. From the IEEE Standard Glossary of Software Engineering Terminology, 1983, software life cycle is defined as follows:

“The time duration that begins when a software product is implemented and ends when it is no longer present for use. The software life-cycle typically includes a requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase and sometimes, retirement phase”.

There are several software development life cycle models and many companies adopt their own models, but all have very similar patterns. Earlier approaches to software development were adhoc or they were programming oriented, which considered development of software from only programming point of view. Such approaches resulted in degradation of software quality and hence reliability of software decreases.

The most primitive model of software development is Waterfall model. After that, many SDLC models and their detailed phases have been
developed that are proven to be more robust [MAH2012] [TAY2011]. The main phases of SDLC are mentioned below:

1.2.1 Phases of SDLC

- Requirement analysis
- Specification
- Software design
- Implementation
- Testing
- Deployment
- Maintenance

The adherence to particular life cycle model helps to produce good quality software. This results software development in a systematic and disciplined manner.

1.2.2 Various SDLC Models

- Waterfall Model
- Iterative Waterfall Model
- V-Shaped Model
- Prototype Model
- Spiral Model
- Agile Model
- RAD Model
- Incremental Model

1.2.3 Explanation & Comparison of Various Models

**Waterfall Model:** This model is also known as classic life cycle model. It is sequential & systematic approach to software development that begins with customer specification and leads to planning, modeling, construction and deployment [SHA2011] [TAY2011].

In this model, phases are organized in linear order, which has some benefits i.e. it clearly identifies the end of one phase and beginning of
next. It is simplest model & due to its simplicity, it divides a big task into cleanly divided phases [MAH2012] [BAL2012].

![Waterfall Model Diagram](image)

**Figure 1.1: Waterfall Model [MAH2012a]**

The Classical Waterfall Model was popularized in 1970 and is the backbone of many other software life cycle models. This process model is structured as a cascade of phases, where output of one-phase acts as the input to the next phase. The classical waterfall model is an unrealistic one since there is no provision of detecting and rectifying the error at any stage of the life cycle. However, in practical developments, there is always chance of errors, due to various reasons, in almost every phase of the life cycle. Therefore, in any practical software development work, it is not possible to strictly follow the classical waterfall model [ROY1987] [MOH2010].

**Advantages of Waterfall Model:**
- It is easy to understand.
- After each stage, a product is delivered.
- There is no overlapping among the various phases.
- Documentation is available at the end of each phase.
Disadvantages of Waterfall Model:

- This model assumes all the requirements to be defined early in the project, which is always not possible. [TAY2011].

- Changes are needed to be made in the project but while using waterfall model changes cannot be accommodated. We cannot move in backward direction as it strictly follows sequential order [SHA2011].

- This model cannot be used in large projects. Because in making large projects requirements may change, new technology may be developed which better suites our project so making all such changes is not possible in waterfall model.

- Most of the time customer does not know what he wants in the software, how the project should behave so any such changes mentioned by him in the later stages will cause many problems, even small changes are difficult to make later on.

- Until the final stage is not complete, working software cannot be delivered to the customer.

Due to all these issues in waterfall model, it should be used only in applications where the requirements are well understood.

Iterative Waterfall Model: Iterative waterfall model was developed to overcome the disadvantage of classical waterfall model. It consists of the same phases as the waterfall model but it is less restricted. With this model, it is possible to move in backward direction if the situation demands. A useable product is released at the end of each cycle and with each release, extra functionality is added. The big project is divided into small projects, each small-developed part is given to the system users, and their valuable feedback is obtained [MAH2012] [LAU2009].
The main change to waterfall model is the feedback path from every phase to the preceding phase. Therefore, when during testing, errors are identified it can be removed by moving through these feedback path. Even the design can be changed, if it is identified in the later stages that it is not according to the specification of the customer [SHA2009] [MAG2010].

**Basic Principles of Iterative Waterfall Model:**

Iterative waterfall model suggests feedback paths in the classical waterfall model from every phase to its preceding phases. It allow for the correction of the errors committed during a phase that are detected in later phases. After detecting the error in later phases, it would be necessary not only to rework the design, but also to appropriately redo the coding and the system testing, thus incurring higher cost [MAH2012] [MOH2010] [GHE2002].

- The problems with the Waterfall Model created a demand for a new method of developing systems, which could provide faster results, require less up-front information and offer greater flexibility [MAH2012a].

- Iterative model, the project is divided into small parts. This allows the development team to demonstrate results earlier in the process and obtain valuable feedback from system users [MAH2012a].
• Each iteration is actually a mini-Waterfall process with the feedback from one phase providing vital information for the design of the next phase.

**Advantages of Iterative Waterfall Model:**

• It can be used only for the projects where requirements are not well understood in the earlier stages.
• An operational quality product is produced at each release of a part.
• Uses “divide and conquer” breakdown of tasks.

**Disadvantages of Iterative Waterfall Model:**

• There is no clear separation between the phases.
• It is quite difficult to handle this model.

**V-Shape Model:** The v-shape model is a verification and validation model. It does not follow a sequential approach instead the stages moves upward when the coding is complete. It verifies from the previous step before moving to the next step. It involves verification and validation at every step [TAY2011] [BAL2012]. Testing is not done at once as in waterfall model, in this the developer and tester works in parallel. Like the waterfall model, the V-Shaped model is linear in nature. Each phase must be completed before the next phase starts. However, emphasis on testing in this model is more than that in the waterfall model [MAH2012].

![Figure 1.3: V-shape Model [MOH2010]](image-url)
Advantages of V-Shape Model:
- Simple and easy to use.
- Each phase has specific deliverables [MOH2010].
- Requirements can be changed anytime in this model.
- Milestones can track progress.

Disadvantages of V-Shape Model:
- It is not easy to handle concurrent events.
- It does not consider risk analysis activities, as it requires highly specific expertise.
- Does not handle iterations or phases.
- It is not suitable for smaller projects.

**Prototyping Model:** The prototyping model produces a working prototype before the actual software is built. Prototyping is a process that is part of the analysis phase of the cycle. A quick design is made and a prototype is built. This prototype is given to the customer and his feedback is obtained, based on the feedback of the customer, changes are made to the software, again the software is given for taking feedback [MOH2010]. This process is continued until the customer is satisfied. When the customer is satisfied, the actual system is developed using either waterfall model or the exploratory model. Prototyping works well for complex and large systems. [MAH2012a] [TAY2011].

![Figure 1.4: Prototyping Model](source: www.qastation.files.wordpress.com)
**Throwaway prototyping model:** Brooks advocated it. It is useful in situations where requirements and user's needs are unclear or poorly specified. The approach is to construct a quick and dirty partial implementation of the system during or before the requirements phase [BRO1995] [JAL2005] [MAH2012].

**Evolutionary prototyping model:** This is a mix of Waterfall model and prototyping. It is suitable in projects where the main problem is user interface requirements, but internal architecture is relatively well established and static [JAL2005] [MAH2012].

**Advantages of Prototyping Model:**
- Prototypes developed give an idea to the user how the actual system will look like, which helps user in understanding deeply about the software.
- Reduces development time development costs.
- Developers receive quantifiable user feedback.
- User is involved throughout the development that increases the chances of acceptance of the product.
- Customer is highly satisfied using this model.
- Developers get an idea how the future enhancements can be made.
- This approach decreases the cost, time and allows for more iteration.

**Disadvantages of Prototyping Model:**
- It sometimes may lead to incomplete documentation.
- Sometimes the users expectation about the performance of the end system to be the exactly as the prototype which in real is not possible.
- It is possible that systems are left unfinished and implemented before they are ready.
- It is generally not suitable for large systems

**Spiral Model:** Boehm proposed the Spiral model in 1988 [BOE1988]. It involves repetition of the same set of life-cycle phases such as plan, develop, build and evaluate until development is complete. The main
emphasis is given on risk analysis. It encounters almost all the different types of risks such as cost overruns, change in requirements, loss of intelligent project personnel, unavailability of necessary hardware, competition from other software developers, technological drawbacks which obsolete the project and many more [BOE1998] [BOE2001] [MAH2012].

In Spiral model work is done incrementally therefore; it is somewhat similar to incremental model with a difference that, in this model, more emphasis is on the risk analysis. In the risk analysis phase, a process is applied to evaluate the risk and alternate solutions [MAH2012a] [TAY2011].

The spiral model consists of four phases:

- Determine objectives, alternatives, constraints.
- Evaluate alternatives, identify/resolve risks.
- Develop/verify next-level product.
- Plan next phase.

The spiral model is a combination of top down & bottom up approach. The project is divided into several smaller modules and risk is evaluated after a module cycle is complete. Every module cycle consists of the same four phases. The cycle can also be referred to as a trip. Development
takes place in the third phase. Customer involvement is there in the spiral model in evaluation phase [MOH2010] [MAH2012].

**Advantages of Spiral Model:**

- Large amount of risk assessment is there, as module is evaluated after every cycle.
- Suitable for large projects.
- It makes use of existing modules hence reusability is there which results in better productivity.
- A quality product results due to proper control of time, cost and manpower requirements.
- Bugs or errors are eliminated in the early stages.
- Guarantee of success is there.

**Disadvantages of Spiral Model:**

- Difficult to do risk evaluation, risk assessment and risk management.
- Risk analysis requires a team of experts.
- Difficult to do time and cost estimations.
- Not suitable for smaller projects.
- Complexity is there.

**Agile Model:** Agile model is a kind of incremental model. The software is produced in smaller releases and each new release depends on the previous release. Agile model is used in time critical applications as releases are done quickly. Each release is properly tested in order to maintain the quality of the software. One of the advance feature of the Agile model is that changes can be made even late in the development. New features can be easily added and unwanted can be removed based on the feedback of the customer [LEO2008] [BAL2012].

In 2001, Kent Beck and 16 other noted software developers proposed an agile view of process. Agile software engineering combines a philosophy and a set of development guidelines. The philosophy encourages
customer satisfaction and early incremental delivery of software, small, highly motivated project teams, informal methods, minimal software engineering work products and overall development simplicity. The development guidelines stress on delivery over analysis and design and active and continuous communication between developers and customers [BEC2000] [MAH2012] [LEO2008].

The term ‘agile’ refers to a philosophy of software development. Extreme Programming, Scrum, Crystal, Adaptive Software Development (ASD) are agile methodologies [BOE2002] [COC2001] [HIG2002].

![Figure 1.6: Agile Model](source: www.tutorialspoint.com)

**Advantages of Agile Model:**
- Customer satisfaction is there by rapid, continuous delivery of software.
- Suitable for smaller projects.
- Changes can be made anytime in the project.
- Able to respond to changing environment.
- Customer is involved so there is no concept of misunderstanding.
- Very little planning is required.
• Easy to manage and flexible.

**Disadvantages of Agile Model:**
• Cannot be used for larger projects as time and efforts required are difficult to judge.
• Requires highly expertise for the development.
• Training required for the new programmers.
• Not suitable to handle compels dependencies.
• Highly depend on the customer interaction, if the customer is not clear, the project can be driven in the wrong direction.

**RAD Model:** RAD model is Rapid Application Development model. It uses component-based construction and therefore results in a rapid development. Once the requirements are well understood, a fully functional model can be developed in short times. In this model, components or functions are developed in parallel and then combined [SHA2011] [TAY2011]. The phases in the rapid application development are business modeling, data modeling, process modeling, application development, testing, and turnover. IBM proposed Rapid Application Development model in 1980. This model is based on an evolving prototype that is not thrown away. Rapid Application Development model is the first model, which emphasizes a short development cycle e.g., 60 to 90 days. It is a “high-speed” adaptation of the waterfall model, in which rapid development is achieved by using component based construction approach [MAR1991] [BUT1994] [MAH2012].

*Business Modeling:* The business model is designed in terms of information flow and distribution of information in various business channels.

*Data Modeling:* The data gathered during the business modeling is reviewed and analyzed and set of data objects are created.

*Process Modeling:* The data objects are transformed. Processing descriptions are created for addition, deletion or retrieval of data object.
• Application Generation: It makes use of existing components or software functionality by the use of automated tools.

• Testing and Turnover: testing is done in the end.

Advantages of RAD Model:

• Takes less development time.
• Makes reuse of existing components.
• Involves customer feedback.
• Progress can be measured.
• Productivity with fewer people.
• Changing requirements can be accommodated.

![Figure 1.7: RAD Model](softwareprojectmanager.com)

Disadvantages of RAD Model:

• System that can be divided into modules can be developed through RAD.
• Requires skilled developers.
• More complexity is there.
• Inappropriate for small and economical projects.

Incremental Model: In Incremental model the whole project is divided into various modules and modules are developed in parallel. At one time, various sequential cycles are taking place. Each cycle produces a deliverable module. Each increment consists of requirements analysis,
coding, design and test phases. Incremental model is iterative in nature. Increments can also be planned to manage technical risks. The life cycle is also referred to as the successive versions or evolutionary model [JAL2005] [MCC1996].

Incremental model is an intuitive approach to the waterfall model. Multiple development cycles take place here, making the life cycle a “multi waterfall” cycle. Cycles are divided up into smaller, more easily managed iterations. A working version of software is produced during the first iteration, so we have working software early on during the software life cycle. Subsequent iterations build on the initial software produced during the first iteration [JAL2005] [PRE2004] [MAH2012].

Advantages of Incremental Model:

- Working software is produced in early stages.
- Testing and debugging is an easy task as a build is small.
- Feedback can be taken after each build.
- Changes can be made at later stages.
- Initial delivery cost is smaller.

Disadvantages of Incremental Model:
- Needs proper understanding of the requirements.
- Needs good planning and design.
1.3 The Unified Modeling Language

The Unified Modeling Language [BOO2005] is a language used for visualization, constructing, specifying and documenting the facts of a software system. UML is a modeling language that is a standard for software blueprints. Its various rules and vocabulary focus on conceptual and physical demonstration of a software system. It addresses a variety of views of architecture of a system as it evolves all across the SDLC. We used UML for building models that are complete, instantly recognizable and precise too. UML address the specification of every design, analysis, and implementation that must be made in developing a software-intensive system. Another feature of modeling is an explicit model facilitates communication in software development process, as a developer can understand the code written by another if it is modeled graphically.

In UML notation, behind each symbol there is a well-defined semantics. Hence, if one developer designs a model, any other developer or tool can interpret that model very easily. Another feature of UML is forward and reverse engineering. Generation of code from a UML model into a programming language is known as forward engineering, where as reverse engineering is reconstructing back UML model from the code. Thus, it enables round trip engineering i.e. ability to work in a textual view or graphical view. In a software development process, there are various objects in addition to raw code. This includes requirement, architecture, design, source code, project plan, test prototypes and releases at various stages of development. In addition to this, UML provides a language platform for activity modeling of project planning and release management.

1.4 Principles of Modeling

A model is simplification of reality. Modeling is a central activity that is used in almost all engineering disciplines. We build models to converse the desired formation and behavior of system. Also, to visualize and
control the system’s architecture to manage risk at early stages. Complex System's model is needed to be build because we cannot realize such a system without its physical presence in front of us. Building of a model may be structural, emphasizing the dynamics of the system, accentuate the organization of the system, or it may be behavioral. So, we build models to understand the system in a better way in which we are developing. There are few more advantages of building a model as below:

- Models help us to document our decisions which we can use as a proof in future.
- Models visualize a system as it is or as we want it to be.
- Models allow us to specify the behavior or structure of a system.
- Model gives us a template that guides us in constructing a system.

More, there are five principles of modeling as described below:

- Every model is expressed at different levels of system.
- In practical life, no single model is sufficient for any system development. Every system is best approached by a small set of independent models.
- The choice of what a model is needed to create has a profound influence on how a problem is attacked and how a solution is shaped.
- UML is a modeling language that is used for specifying, visualizing, constructing and documenting the object-oriented systems.
- The best models are associated to realism.

The main goals of inventing MDL file format was to have a practical and easy way of storing or organizing models that is fast to read & write, easy to use in programming and should be space (storage) efficient. Another reason was to have a readable format for humans who would also be suitable for small-small files, and easy for programs that did not understand the data into a file. The primary application of the MDL file format is placing models for practical representation, as well as difficult
materials and huge polygon meshes. It is a low-tech system, which shows syntax for placing simple data in text and binary files. This further set into nested “chunks”. It also describes numerous types of data chunks. UML models are stored in MDL file format.

1.5 Terminology for Component Storage and Retrieval

Software Library /Repository [BUR1987] [HEN1997] is a set of software assets that are maintained by an organization for possible browsing and/or retrieval. It consists of different reusable assets/components. These assets/components are the result of one development process and same can be used in other/new development processes.

**Browsing and Retrieval:** There are mainly two purposes of maintaining a software library, one is browsing and second is retrieval. Both are explained below in more details:

*Browsing* is inspecting assets for possible extraction, without predefined criterion.

*Retrieval* is identifying and extracting assets that satisfy a predefined matching criterion. Retrieval depends on matching a candidate asset against a user query.

In addition of the above, there are few more techniques come under component storage and retrieval as given below:

**Navigation and Matching:** There are two aspects in the activity of asset retrieval, i.e. navigation and matching.

*Navigation:* determines which assets are visited / inspected and in what order.

*Matching:* determines the condition under which an asset is selected. In testing it consists a condition that involves a given asset to determine whether the asset is selected or not.
**Relevance criterion and matching condition:** There are two aspects of conditions.

*Relevance criterion:* It is a condition under which an asset is considered to be relevant with respect to a query.

*Matching condition:* It is a condition under which an asset is selected.

**Objectives of asset retrieval operation:** There are two main objectives of asset retrieval operation.

- Exact Retrieval
- Approximate Retrieval

**Exact Retrieval:**

The concept of exact retrieval is to identify library assets that satisfy the submitted query with respect to all the following aspects like functional correctness, architectural assumptions, interaction protocols, resource requirements etc. Exact retrieval fits in the life cycle of black box reuse. The phases of black box reuse are:

- *Exact Retrieval:* This includes identification and retrieval of correct assets.
- *Integration:* In integration, assets are integrated with host system and then system is subjected to some form of integration testing.
- *Assessment:* In assessment, assets that have been retrieved are evaluated with respect to the query; the asset that provides the best fit is selected.
- *Instantiation:* Here the selected asset is duly specialized to fit the purpose of query at hand.

**Approximate Retrieval:**

The concept of approximate retrieval [JIL1997] is to identify library assets that can be modified with minimal efforts to satisfy a submitted query.
Approximate retrieval fits in the life cycle of white box reuse. The phases of white box reuse are:

- **Approximate Retrieval**: This includes identification and retrieval of approximate assets.
- **Integration**: Here the instantiated asset is duly integrated into the host system and duly tested.
- **Modification**: Here the selected asset is duly modified to fit the purpose of the query at hand.
- **Assessment**: Here the retrieved assets are evaluated with respect to the query to be selected, which further minimizes the expected modification effort.

**Assessment Criteria**: A set of criteria is being used for accessing and comparing storage and retrieval methods [JIL1997]. These assessment criteria’s can be categorized into three main categories mentioned below:

- Technical Criteria
- Managerial Criteria
- Human Criteria

**Technical Criteria**: There are mainly six technical evaluation criteria; which are described below:

- **Precision**: The precision of retrieval algorithm can be defined as the ratio of relevant retrieved assets over the total number of retrieved assets.
- **Recall**: The recall of retrieval algorithm can be defined as the ratio of relevant retrieved assets over the total number of relevant assets in the library.
- **Time Complexity per match**: Time complexity of a retrieval algorithm can be defined as number of computation steps that are required to
match the query against a library asset. This is represented by \( O(N) \) notation where \( N \) is some measure of size of the query.

- **Logical Complexity per match**: It is used to measure the performance of a retrieval algorithm. In this performance is considered against the complexity of performing a match. Under this method more complex the match, the better results will be yielded.

- **Automation/Automation potential**: This is being used for storage and retrieval automation.

- **Coverage Ratio**: The coverage of a retrieval algorithm can be defined as average number of assets that are visited over the total size of the library.

**Managerial Criteria**: There are four major managerial evaluation criteria described below:

- **Investment Cost Investment**: cost is the cost that is incurred in setting up a software library.

- **Operating Cost**: Operating cost measures the yearly cost of operating a software library.

- **Pervasiveness**: This measures the extent to which the proposed method is used in research and development.

- **State of Development**: It reflects the state of the development of the method in hand.

**Human Criteria**: There are two human evaluation criteria described below:

- **Difficulty of use**: It reflects the difficulty of use of the proposed method i.e. degree of complexity.

- **Transparency**: It reflects, up to what extent the operation of a asset library depends on an understanding of how the retrieval algorithm works i.e. operation should be transparent to the user to work on system.
1.5.1 Features for Characterizing Software Retrieval Method

The features that are being used for characterizing software retrieval methods are as follows [MIL1998].

- **Scope of Library:** This includes the scope in which the library is to be used. That is whether the library is used within a project, across a program, across a product line, across multiple product lines, worldwide etc.

- **Nature of Assets:** This includes the nature of asset that is stored in the library. Assets include source code, executable code, requirements specification, design description, test data, documentation, and proof etc.

- **Asset representation:** This includes how the assets are represented in the library. It dictates that what form of user queries should take. Assets can be represented as functional specification, signature specification, source code, executable code, requirements specification, documentation, and set of keywords.

- **Storage Structure:** This includes how the assets are stored in library. Storage structures can be hypertext links, flat structure, refinement ordering, ordering by generality.

- **Query representation:** This includes the form of the query submitted to the library. Queries can be submitted in various forms as functional specification, signature specification, keyword list, design pattern, behavioral sample etc.

- **Navigation Scheme:** This includes how the assets are visited or navigated. Various navigation schemes include Exhaustive linear scan, navigation hypertexts links, and navigating refinement relations.

- **Matching Condition:** This includes the condition that is chosen to check between the submitted query and a candidate library asset. Various factors for matching condition include correctness formula,
signature identity, signature refinement, equality of keywords, natural language analysis and pattern recognition.

- **Retrieval Goal:** This includes, finding out assets that are correct with respect to a given query. Various retrieval goals include correctness, functional proximity and structural proximity.

### 1.6 Problem Outline

The main utility of numerous software life cycle models is to identify and arrange the phases involved in software development. One of the main short coming of all these models is that none of them explicitly encourages reusability in all their phases. Therefore, a software life cycle model that emphasizes the importance of component reuse during software development with tools is still in demand.

Companies that are concerned in growth of quality and productivity use the software reuse as a key factor. Although without appropriate life cycle model, it is very hard to achieve the desired benefits. Therefore, this study proposes a practically efficient life cycle model. The model is composed of guidelines and a well-defined set of activities. This study proposes that the reuse of existing components will lead to more cost-effective, more reliable, and higher quality software products.

The K-Model has been proposed to target software reusability and reengineering practices during component-based software development. The representation of the K-Model considers overlapping and iteration where appropriate. The emphasis is on reusability and reengineering. Reusability leads to use of composition methods during software creation; this is achieved initially by choosing reusable components and joining them, or by adjusting the software to a point where it become easy to select components from a reusable library. Reusability within this life cycle is efficient and more cost effective than within the traditional models because it integrates at its core for reuse and the mechanisms to achieve it.
1.7 Research Objectives

Following five research objectives are identified and planned in order to achieve enhanced user experience in terms of software engineering while developing any SDLC.

i) **To identify and analyze various challenges in Software development using existing SDLC models**

This objective deals with the identification of various challenges prevailing in the area of software development using existing SDLC models. Challenges include choice of a particular model in different situation. Many researchers have purposed various Models. But no SDLC can be termed as the benchmark for software development and almost all SDLCs are application specific rather than generic.

ii) **To design reusability based SDLC model for efficient and cost effective software development**

This objective deals with the identification of methods and techniques for software component reuse during software development life cycle. The challenge is to design a model that is equipped with the concept of Reengineering, Domain Engineering, Iterative development and benefits of feedback to previous phases.

iii) **To link the proposed model with SCSRS**

The goal is to propose a software component storage and retrieval system (SCSRS). The first step concerned with the construction and storage of components in the reusable component repository (RCR). Such reusable components could be collected from the recent projects in the same domain. The next step focuses on the retrieval aspect of the components. The developers try to find the required component from the RCR. After minor adaptations on the retrieval result, the component can be reused.
iv) Integrated tool support for optimization of search results to reduce manual work

The goal is to optimize the search result from the repository. Manually it is very difficult to select the most appropriate software asset retrieved from the repository from a large number of matched components. The goal is to select a component that is most suitable for a particular application development.

v) Applying proposed model for software development

Here the goal is to compare the proposed model with some existing models on certain attributes to find the suitability of the proposed model for any particular scenario. The goal of the present study is to implement practically the model in software companies and collect their feedback on the proposed model.

1.8 Scope of Study

The areas covered in the present study are as follows:

- Perform projects based analysis of the proposed model as compared to existing models in order to evaluate the applicability of various software engineering paradigms to develop a complex, dynamic and open software system.

- Compare and evaluate the improvements on various paradigms in the proposed model with the existing ones.

1.9 Organization of the Thesis

The thesis is organized into eight chapters as discussed briefly below:

Chapter 1: Introduction

This chapter contains introduction of software component reusability, existing software life cycle models, Unified Modeling Language,
component storage and retrieval, problem outline, research objectives, scope of the study and organization of thesis.

**Chapter 2: Literature Survey**

This chapter provides the study and review of literature based on proposed research objectives.

**Chapter 3: Research Methodology**

This chapter describes the concepts underlying research methodology and research background. This chapter describes the methodology that is taken during the research work.

**Chapter 4: K-Model**

In this chapter, we have described the proposed K-model along with the description of all the SDLC phases i.e. Domain Engineering, Framewrok, Reusable components repository, System requirement analysis, Partial design, Assembly, System design, Implementation and Integration, Testing, Reengineering and Maintenance. Here the workflow in K-Model is also discussed along with the scenario where this model is most suitable.

**Chapter 5: Reusable Components Repository**

This chapter poses the design and implementation of reusable software components repository. This chapter explains system overview, developing component repository and algorithm for updating reusable software components repository. This chapter also describes comparative analysis of existing reusable component retrieval system and proposed system. It also explains the technique for storage of data and retrieval techniques based on MDL file format and semantic keywords.

**Chapter 6: Optimization in Repository**

This chapter introduces a novice mechanism to provide automatic rankings to the search results. The rankings are used for optimization of
search results. This chapter explains the design and implementation of feed forward back propagation Neural Network. It also describes result analysis of optimization process. Here the Comparative Analysis of existing reusable software component retrieval systems and proposed system with optimization process are also discussed.

Chapter 7: Application of K Model in Software Development

This chapter contains the statements and parameters used for the analysis of K-Model. Here the detailed information about various projects that are designed and developed using K-Model and their comparison with similar projects already developed using any SDLC model are given and the result of case studies are discussed. A critical review of proposed model by experts has also been explained.

Chapter 8: Conclusions and Future Scope

This chapter contains the findings of the work and also explains the future scope.