COMPOSITORY: THE AUTOMATED COMPONENT REPOSITORY

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

Submitted By
Pankaj Vohra
(Roll No. 801131019)

Under the supervision of:
Ms. Ashima Singh
Assistant Professor
Computer Science and Engineering Department
Thapar University, Patiala

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2013

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Compository: The Automated Component Repository*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Ms. Ashima Singh* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

(Pankaj Vohra)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

15/7/13

(Ashima Singh)
Assistant Professor
CSED, TU, Patiala

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

i

Acknowledgement

First of all I would like to thank the Almighty, who has always guided me to work on the right path of the life. Foremost, I would like to express my deep and sincere gratitude to my supervisor Ms. Ashima Singh, *Assistant Professor,* Computer Science & Engineering Department. I thank my supervisor for her time, patience, discussions and valuable comments. Her enthusiasm and optimism made this experience both rewarding and enjoyable.

I am equally grateful to Dr. Maninder Singh, *Associate Professor and Head*, Computer Science & Engineering Department, for motivation and inspiration that triggered me for the thesis work. I will be failing in my duty if I don't express my gratitude to, Dr. S. K. Mohapatra, *Senior Professor and Dean of Academic Affairs,* of the University for making provisions of infrastructure such as library facilities, computer labs equipped with net facilities, immensely useful for the learners to equip themselves with the latest in the field. I am also thankful to the entire faculty and staff members of Computer Science and Engineering Department for their direct-indirect help, cooperation, love and affection, which made my stay at Thapar University memorable.

Pankaj Vohra
(801131019)

Abstract

In software industry, there is progressive increase in reusability of software components. To increase productivity and to reduce time-to-market organizations are adapting for-reuse and with-reuse approach. In order to make reusable component available in future projects there must be a system to store software components so that they can be efficiently retrieved as per user requirements. So this work is towards construction of component repository with efficient storage and retrieval of software component that best fits user's requirements. Besides, it enhances reusability; reduces efforts of software development from scratch and increases productivity.

Component Retrieval in component based software engineering is the core functionality to be performed as components generated in the software organization are maintained in a software repository. Also, component retrieval is one of the difficult tasks because it depends on the classification and categorization of components in a software repository. Software repository is the storage medium used for storing, classifying, maintaining, browsing and retrieving components. The main focus of maintaining software repository is to enhance reusability by storing reusable components and retrieving them for reuse in other projects. If a specific component is to be searched through keywords or phrase, then the potential keywords need to be stored in software repository. If in any case the keyword or the phrase framed is unable to give desired result, then the software repository is required to be enriched with more efficient retrieval technique. In this research we have proposed and materialized "Auto-Detect-Fragment-Store technique", "Auto-Generation of closely related Keywords" w.r.t. to component name, "User Priority-Based Component Retrieval technique" and "Keyword-Based Component Retrieval, using Tags" and its retrieval mechanisms. "The Compository" tool is constructed which implements all the above proposed techniques and it gives specific or more exact results, As it incorporates processing of user input and automatically generates closely related tags using most reliable search engine database Google and its dynamic pages. A fully functional automated component repository is generated.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Software Process Models

Software Processes are the lifeline of any Software Development Model. Software Processes decide the survival of a particular software development model in the market as well as in software organization. The set of processes those proved to be effective and efficient for software development in one organization may or may not be followed in another organization. That is other organization finds another approach for software development more convenient to work with. The primary function of software development process models is to "determine the order of the stages involved in software development and evolution and to establish the transition criteria for progressing from one stage to the next" [1]. In history various models were proposed. Fig. 1.1 illustrates the evolution of process models in the past decades.

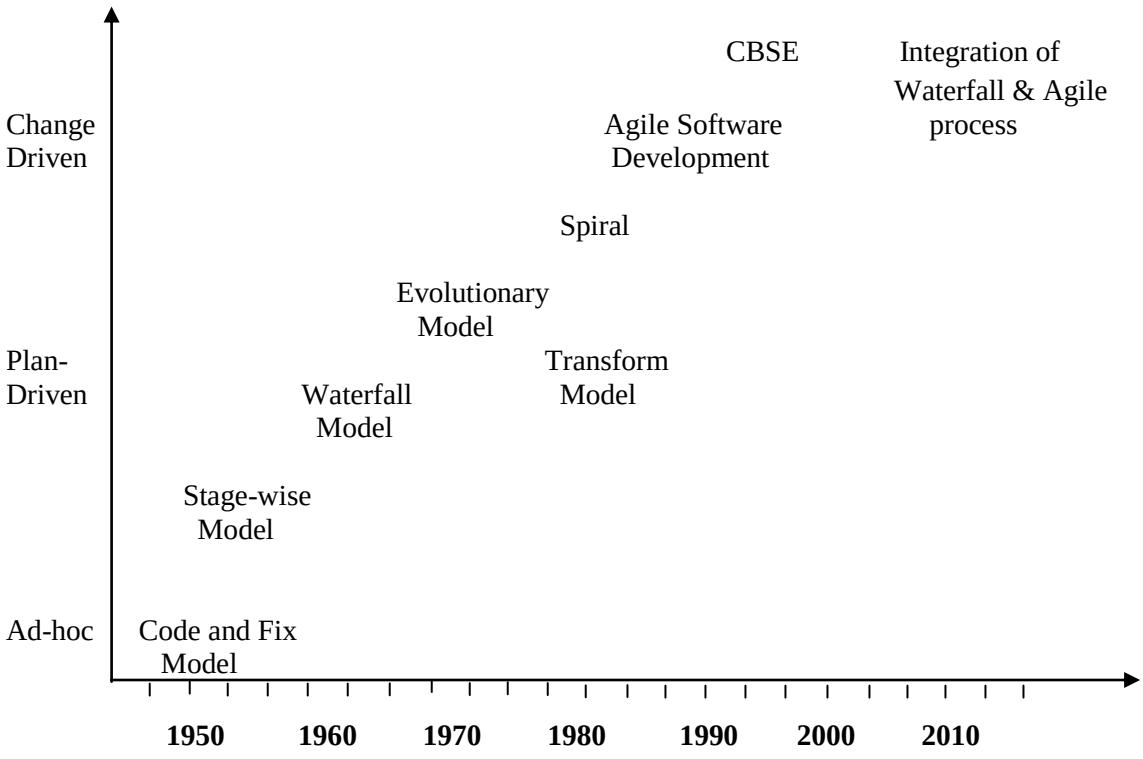

Fig. 1.1: Progression of Process Models [2]

It has also been suggested that the evolution of software development models originates from the problems of ad hoc programming that, at first, led towards traditional plan-driven models and towards iterative change-driven models of software development. The original meaning of the Latin term “ad hoc”, refers to a methodology that has been designed for a special purpose (ad hoc = for the purpose of). However in this context, as often in software engineering literature [2], the term "ad hoc" is used to refer to the low degree of methodological discipline.

1.2 Plan-Driven Models for Software Development

The plan-driven approaches of software development have been defined as document-driven, code-driven, and traditional process models [1]. As the names suggest, a common feature for the plan-driven process models is their emphasis on defining the scope, schedule, and costs of the project upfront including, for example, an early fixing stage and extensive documentation of the end product requirements. One common characteristic could also be the recurrence of the software development phases only once during the development process i.e. with only hints of iterations [3]. The two-step process model of code-and-fix, used in the early days of software development, resulted in difficulties that necessitated explicit sequencing of the phases of software development [1]. In particular, the need to design prior to coding, to define requirements prior to design, and the need for early preparation for testing and modification were identified [1]. One of the first models to rise to that challenge was the stage-wise model as early as in the middle of the 1950s [4]. This model evolved from the problems caused by the increasing size of software programs, which could not be handled by a single programmer [4]. In 1968, the NATO Science Committee held a software engineering conference in Garmisch, Germany, where the software crisis, or software gap, was discussed. A standardization of the software development process with an emphasis on quality, costs, and development practices was the key recommendation of the conference [6].

Soon after this, as refinement of the stepwise model, the waterfall model was introduced. The early version of the waterfall model was introduced in 1970 [5] and it has since evolved into a concept consisting of the sequential phases of requirements analysis,

design, and development [4]. The waterfall model provided two main advances over the stepwise model: it introduced prototyping to parallel the stages of requirements analysis and design, and provided feedback loops between the sequential stages [1]. It should also be noted that, already in the early waterfall model [5], it had been realized that it might be necessary to first build a pilot model of the system, i.e., to conduct two cycles of development and to obtain feedback to adjust the model. Thus, hints of iterations in the model can be seen yet .this iterative feedback-based step has been lost in most descriptions of this model, although it is clearly not classic IID [7]. Today, the waterfall model has been adopted for most software acquisition standards in government and industry. While the waterfall model has solved various core problems in software development, it also includes features not appropriate for every software development context. One central problem of the waterfall model has been identified as its emphasis on fully elaborated documents as completion criteria for early requirements and design phases [1].

It can be argued that the plan-driven models of software development can and should be applied in a dynamic way by repeating the phases or even the entire process, if necessary. However, the original purpose of these process models was not to welcome changes during the development, but rather to try to fix factors, such as scope, time and money, up-front in order to eliminate change which was considered a risk factor.

1.3 Iterative Change-Driven Models for Software Development

The software development models, developed after the waterfall model, seem to have the common aim of enabling, at least to some degree, the evolution of product requirements during the process of software development. This contributed one main modification to the earlier software development models: the adoption of the iterative and incremental approach. Iterative development refers to the overall lifecycle model in which the software is built in several iterations in sequence [8]. Every iteration can be considered as a mini project in which the activities of requirements analysis, design, implementation and testing are conducted in order to produce a subset of the final system, often resulting in internal iteration release. An iteration release has been defined as “a stable, integrated and tested partially complete system”. A development approach where the system is

developed in several iterations is called iterative and incremental development (IID), yet it is often referred to as iterative development [8].

Even though agile software development has recently brought the IID approach of developing software into the spotlight, the history of these approaches is, in fact, considerably longer [7]. Among the first models that focused on increasing the possibility of determining product improvements throughout the development process, was the evolutionary development (Evo) model. This concept was first introduced in 1981 [9] and has been expanded by Gilb [10][11].

The spiral model of the late 1980s [1] typically consists of four iteratively repeatable steps: 1) determining the objectives, alternatives, and constraints, 2) evaluating alternatives, and identifying and resolving risks, 3) development and verification, and 4) planning the next phase. [1] Defined the spiral model as a risk-driven approach for software development.

Agile software development, which emerged in the mid-1990s, can also be classified as an iterative and change-driven software development approach. It could be argued that at present there is no common agile process model with specified phases, but there is rather a set of fundamentals common to the methods claiming to be agile. However, Extreme Programming (XP) [12], which is probably the best-known among the first agile methodologies, contains an underlying process model for agile software development that has been adopted and adapted by its successors.

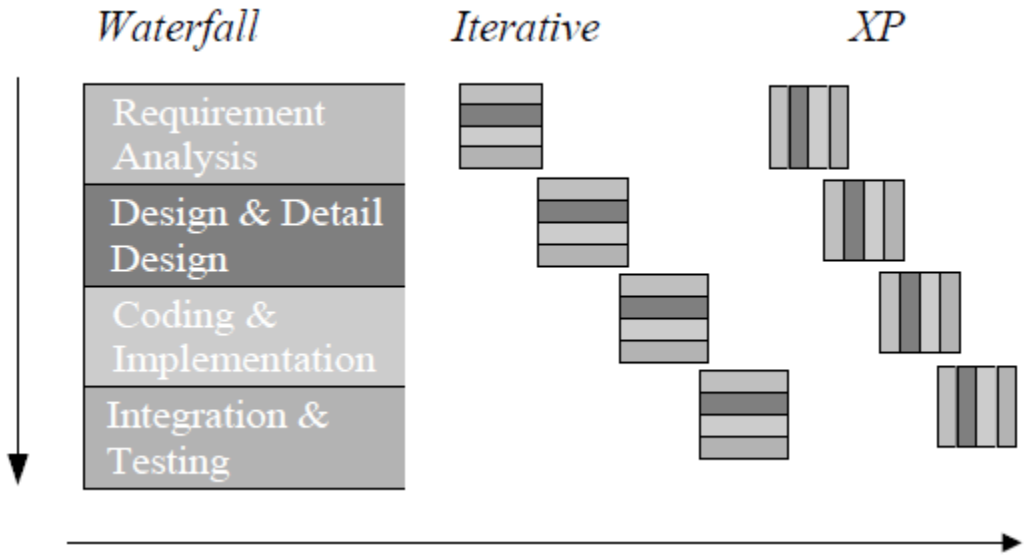

Fig. 1.2: A Contrast: Waterfall, Iterative, Agility and XP[12]

Fig. 1.2 illustrates how Beck [1] has compared the agile development model of XP with the waterfall model and with the iterative processes. XP aims at combining the activities of analysis, design, implementation and testing, a little at a time, throughout the entire software development process. The common feature of agile methods is the recognition that software development cannot be considered to be a defined process, but rather an empirical (or nonlinear) one due to the constant changes that are welcomed during the development of the software product [13].

1.4 Component-Based Software Engineering

CBSE (Component Based Software Engineering) Component-based software engineering emerged in the late 1990s as an approach to software systems development based on reusing software components. CBSE is a process that emphasizes the design and construction of computer-based systems using reusable software components [14].

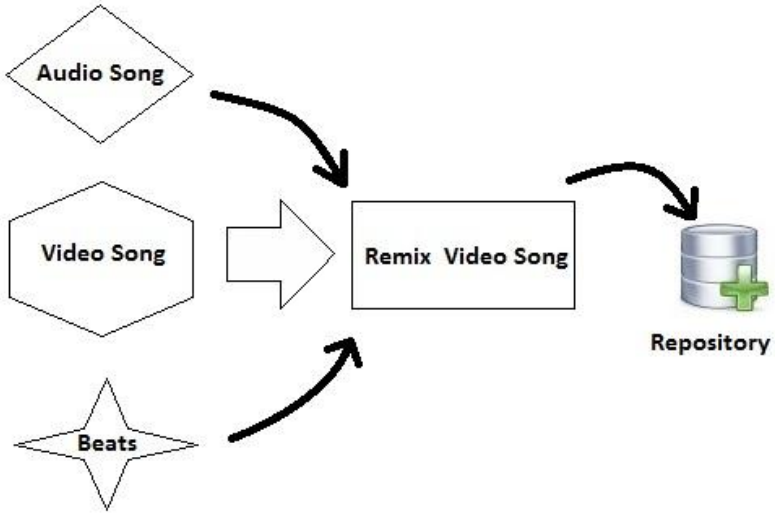

Fig. 1.3: CBSE Process.

Fig. 1.3 describes various reusable components i.e. audio song and video song that are integrated with beats to form Complete System and stored in repository. Component Based Software Engineering confirms the increase in productivity and quality of system and enhances reusability.

1.4.1 Issues with Traditional Software Models

1. Lack of Reusability: In traditional software models development is based on

specific requirements. Product is developed for specific purpose not for general purpose.

2. Lack of standardized component interface between components: Components interfaces are designed for a specific project. There is no consistent mechanism for supporting component interactions
3. Lack of Customization: Customization is not possible in Traditional software development.
4. Lack of Component Interoperability: In traditional software models component interoperability is not possible, due to lack of consistent data exchange mechanism between components and lack of consistent interaction mechanism between components.
5. On following traditional software models for development size and complexity of product increases rapidly.
6. Generally, Cost of development of product crosses our predictions.
7. Due to lack of reusability in traditional software models, product consumes more time and efforts for development from scratch.
8. In traditional software models product is upgraded mostly after development, because requirements must be fixed before initializing its development

1.4.2 Component-Based Development

All the above issues can be resolved by Component-based development. As, component based development provides the idea: to build Software system form pre-existing components. Example – building furniture from existing components, for building components that can be reused in different applications. In CBSE maintenance is done by replacing of component and introducing new components into system. It focuses on reusing and adapting existing components. CBD (Component-based Development) provides idea to build component that can be reused in future project development. CBD is an approach to find, select, adapt, compose and replace components. Fig. 1.4 describes various reusable components i.e. Component 1 and component 2 that are integrated with main component to form Complete System.

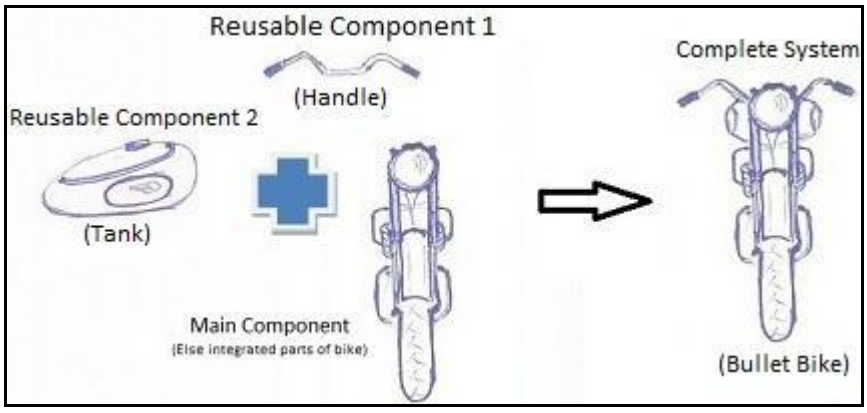

Fig. 1.4: Component Based Development Process

Component Based Development ensures the increase in productivity and quality of system and also improves time-to-market in software development. It also provides maintenance by replacing component with another component. CBD is the process of identifying & selecting reusable components, qualifying, adapting and compose to form a new reusable component.

1.4.3 CBSE vs Traditional Software Engineering

1. CBSE life cycle is shorter as compare to waterfall model

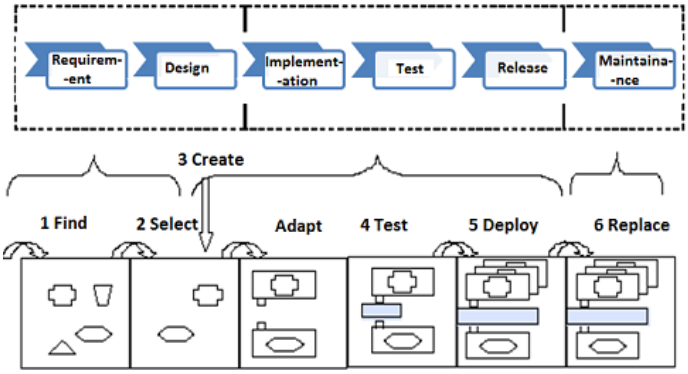

Fig. 1.5: Waterfall vs CBSE Development Cycle [15]

2. CBSE develops architecture.
3. CBSE is less expensive because in this high quality & certified components are reused to form system, which also reduces risk of failure.
4. In CBSE maintenance is easily done by replacing components and introducing new component to system.
5. Time-to-market is less in Component based development.

The timely adopted and adapted changes in software development approaches slowly changed the face of software development. Different flavors of software development originating from classical to agile and component based models showcased distinct ways of software development. As Ad hoc software development approach gave birth to software crisis, middle-aged plan driven models introduced process structure. Now the era of change driven development that is moving towards implementing change in requirement, design and code at any point of time. One of the Agile development technique is SCRUM that supports implementing frequent requirement change. This characteristic of SCRUM made it first choice of software developers. But SCRUM approach compromised on software reusability, whereas CBSE delivers reusable components which basically save the efforts of development for scratch. So in order to save these efforts we need to store the developed components so that it can be reused in another project. . Especially, Component Repository is the tool which provides and store reusable software components that are desired by user.

Chapter 2
Literature Survey

2.1 Introduction

Repository Definition: It is a collection of software components or assets that are maintained by an organization for storage, browsing and/or retrieval of reusable assets directed toward facilitating software component reuse to meet specific cost-effectiveness and productivity goals.

2.1.1 Accessing Criteria

Accessing criteria can be classified in to two categories:

1. Browsing: Browsing a component means exploring the content of repository.
    a. Done before product design, requirements are not clear that what is needed.
    b. When designer have no precise definition of what they need, but do have some idea.
2. Retrieval: navigating the repository to find assets that meets our requirements is called retrieval
    a. Performed after product design.(designer has well specified requirements)

2.1.2 Reusable Assets

Reusable assets are the components which can be stored in repository and can be reused in future projects [19].

1. Executable Code, Source Code
2. Requirement Specifications
3. Designs
4. Test Data,
5. Documentation,
6. Architectures

2.1.3 Component Categories

Software components can be categorized as follows:

1. In-house: Components that are developed inside organization
2. Legacy Systems: Assets that are extracted from legacy systems
3. COTS: Third party components that are not present in repository (Vendor responsibility)

The reason for the existence of a reuse library is to provide ready access to reusable components by the staff of development and maintenance organizations, The number of cases in which library systems are successfully being used to maintain code and other reusable software life cycle components continues to increase. It is essential that the library system support developers and other users in the process of locating, retrieving, and maintaining reusable software components so that they can be reused in future projects.

2.1.4 Repository Features

Usually, Reuse library capabilities include the following:

1. Automation: Automated library system with a Graphical User Interface, for browsing, searching and retrieval of assets;
2. Standard component framework: to include purpose, functional description, certification level, key environmental constraints, historical results of usage and legal restrictions;
3. Classification scheme: Effective classification scheme for each domain;
4. Documentation: Complete system and component documentation (essential feature).

Each library system must be designed to provide as much automated support as possible to users in identification, comparison, evaluation, and retrieval of similar reusable components. Whatever tool is used, the library must have a way to classify Reusable Software Components so that a user can quickly find what is wanted without frustration and delay. Sophisticated, expert system, knowledge-based approaches and new technologies for high-speed text search are the subjects of current research efforts. Generally speaking, software reusable component retrieval methods include browsing, keyword searching, facet approach, syntactic matching, and semantic matching [19].

Effective classification schemes are essential to assist the user in locating and comparing library components, and to speed the process of identifying appropriate components for the task at hand. Finally, system and component documentation complete the cycle of evaluation, and enable the re-user to determine which components have reuse potential with regard to specific requirements, and to fully comprehend the process of obtaining components for reuse in a new application.

2.1.5 UML Specification

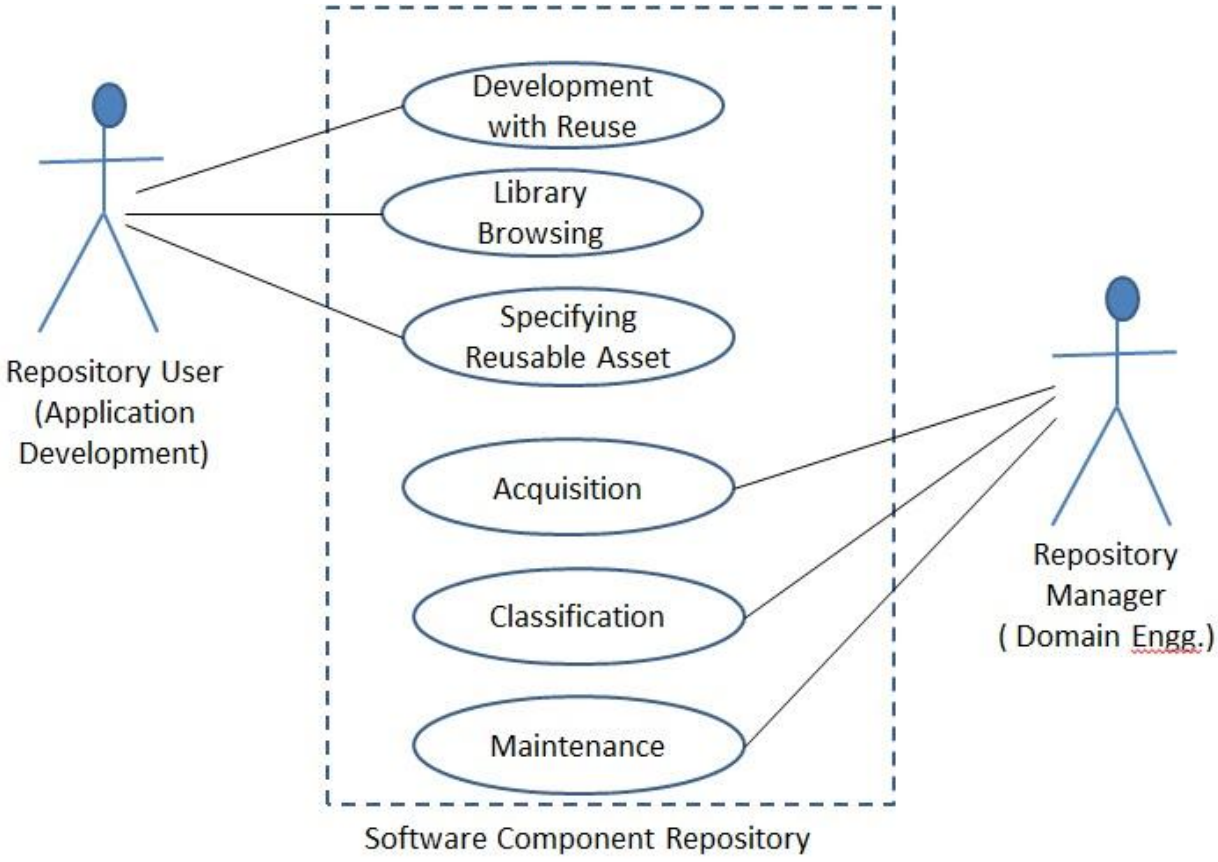

Fig. 2.1: Use Case diagram for Repository System

2.2 Various Component Repository

There are different Communities/Organizations for which a repository is necessary, and each community has somewhat different repository requirements.

2.2.1 Commercial Repositories

1. +1Reuse Repository

The +1Reuse system was developed by +1 Software Engineering Co. in California [16]. It is now running on Sun Workstation platforms. Operating system is Solaris. GUI is based on OpenWindows, Motif, and Common Desktop Environment. Selective reuse significantly improves a user's ability to reuse all source code and documentation from all previous projects (To the best of our knowledge, they are the only company to support this feature). The +1Reuse system supports reuse of: design, documentation, source code, header files, test cases, test shell scripts, expected test results, and modeling information. Since a programmer's productivity can be increased by reusing existing code and documentation, +1Reuse helps to make all source code, documentation, header files, and test files reusable.

2. Software Asset Library Management System (SALMS)

SALMS is a system for classifying, describing, and querying reusable assets [17]. Reuse of software assets at all phases of the software engineering life-cycle is recognized as being one of the major enablers for productivity and quality improvements.

SALMS (Software Asset Library Management System) is a software product which provides mechanisms for classification and storage of software assets, along with techniques for efficiently retrieving them. It fills the gap between development for reuse activities (building, acquiring, or re-engineering of reusable assets) and the development with-reuse activities (using reusable assets in the creation of new software products). It plays a central role in the implementation of a company's reuse program.

In addition, SALMS also provides features:

1. For the requirement management activity, and for the creation and management of a company's technical library.
2. Can be distributed over customer's network of PCs or UNIX workstations and thus be accessible by all developers within a software organization.
3. The user interface is based on WEB Technology.

In SALMS, an asset can be viewed as a collection of artifacts produced throughout the life-cycle, such as requirements, architecture models, design specifications, source code, or test scripts.

3. Automated Software Reuse repository (ASRR)

The Automated Software Reuse Repository (ASRR) tool provides users with a searchable repository of reuse information [18]. It consists of two main parts, the administration tool and the reuse repository.

1. The administration portion of the tool performs user administrative functionality including: the ability to add, delete, or change users and their attributes. The attributes include the following: security levels, group and security permissions to add, edit and delete modules.
2. The reuse repository allows the user to upload artifacts and store them in a searchable repository.

The ASRR provides the following functions [18]:

1. Program Control: Provides complete login control for the ASRR.
2. Protection: The ASRR can limit a user's edit, delete, view, add, upload and download artifacts permissions through the administration portion of the tool.
3. Security: The ASRR tool provides extra security for inactive users by logging them out of the ASRR after a 30-minute period of inactivity.
4. Easy Access to Reuse Items: The ASRR tool allows registered users flexibility in searching for reuse items in the reuse repository by allowing the users to search for strings of words using “not”, “or”, or “and” in searching.
5. Reuse Information Readily Available for Users: Specific information is available for reuse module items including the platforms utilized, ease of reuse and any additional information obtained from users.

4. The Universal Repository

The Universal Repository was developed by Unisys [19]. It is designed to help customers move forward into a repository-based development environment. The Universal

Repository, which is based on object-oriented principles, can function as the backbone of a flexible workgroup or enterprise development environment. At the core of this repository is the Repository Services Model (RSM) - which can encompass representations of all tools, database management systems (DBMSs), programming languages, business rules, and data.

Customers can extend the Universal Repository by adding their own models based on the structures provided in the RSM. The summation of all models defined in a repository is called the information model. Each part of customers' development environment becomes an integrated piece of the whole when customers use the models encompassed within the information model [19]. This unified view enables both developers and customers to achieve inter-tool integration.

Support and training are available to help customers quickly adopt this new technology. By providing a shared catalogue of all software components, a repository promotes reuse.

5. Reuse Library Toolset (RLT)

EVB Software Engineering, Inc. announced the commercial release of the Reuse Library Toolset (RLT) in 1994 [20]. RLT is a system for creating and managing collections of reusable assets independent of programming language, design method, or development process. To represent all life-cycle assets RLT employs the Extended Faceted Classification System, controlled keyword; attribute value (frames), and asset interdependencies.

Experience has shown that the cost of producing software is significantly reduced when reuse is an integral part of the process. RLT supports all reuse oriented tasks, from library management through domain analysis to asset search and retrieval [20]. With its intuitive graphical user interface, RLT is easy for beginners to learn, yet provides powerful functionality for advanced users with complex needs.

RLT provides reuse and library metrics, client-server architecture, and ability to exchange library information across multiple platforms and databases.

These include: DEC Alpha OSF1, HP/UX, SGI, SunOS, Solaris, Informix, Oracle, and Sybase. Additional platforms have been supported in 1995 include: Windows 3.1/NT and OS/2.

6. HSTX Reuse Repository

The HSTX Reuse Repository was developed by Hughes STX Corporation [21]. The mechanisms are designed so users can search/browse the contents of the Reuse Repository for what they need and submit contributions to the Reuse Repository librarian through WWW pages.

2.2.2 Government Repositories

1. Defense Software Repository System (DSRS)

The DSRS is an automated repository for storing and retrieving Reusable Software Assets (RSAs) [22]. The DSRS software now manages inventories of reusable assets at seven software reuse support centers (SRSCs). The DSRS serves as a central collection point for quality RSAs, and facilitates software reuse by offering developers the opportunity to match their requirements with existing software products. DSRS accounts are available for Government employees and contractor personnel currently supporting Government projects. The Account Request Form must be approved and signed by the requestor's Government Project Manager prior to submission to the SRP. The Customer Assistance Office (CAO) is the SRP point of contact for both technical and non-technical information and support [22].

The Defense Software Repository System (DSRS) supports reusable asset classification to comply with published guidance (DoD 8020.1-M and TAFIM), support domain engineering, establish more effective asset searching, and increase interoperability. The DoD software community is trying to change its software engineering model from its current software cycle to a process-driven, domain-specific, architecture-based, repository-assisted way of constructing software [23]. In this changing environment, the DSRS has the highest potential to become the DoD standard reuse repository because it is the only existing deployed, operational repository with multiple interoperable locations across DoD. Seven DSRS locations support nearly 1,000 users and list nearly 9,000 reusable assets. The DISA DSRS alone lists 3,880 reusable assets and has 400 user accounts.

DSRS is adaptable to additional types of reusable assets and better methods of describing them. The description of repository assets is called classification. The far-term strategy of

the DSRS is to support a virtual repository. These interconnected repositories will provide the ability to locate and share reusable components across domains and among the services.

2. Library Interoperability Demonstration (LID)

The Library Interoperability Demonstration (LID) is a prototype library system [24]. It is used to illustrate how monolithic reuse libraries can be decomposed into distinct, functional layers connected by open interfaces, such as those specified by Asset Library Open Architecture Framework (ALOAF). It is a collaboration between SAIC and Unisys. A reuse library can be divided into three distinct layers which are connected via open interfaces, thus providing opportunities for interoperability at each layer [24].

The three layers are:

1. User Interfaces: The demonstration includes two user interface tools: a graphical browser derived from the Unisys Reuse Library Framework (RLF) and a text based browser modeled after SPS's InQuisiX reuse library system. Both tools are built upon Ada bindings to OSF/Motif.
2. Asset Catalogs: The demonstration shows two asset catalogs. The first catalog is derived from the Unisys collection of ASW components, and resides on an IBM RISC System/6000 at the STARS Technology Center. The second catalog is derived from SAIC's collection of flight simulator components, and resides on an IBM RISC System/6000 at the SAIC offices in Orlando, FL. The interface between each of the catalogs and the user interface tools is defined by the ALOAF.
3. Asset Storage: In the demonstration, the storage of assets is provided by the AFS cell at the STARS Technology Center. Neither catalog stores assets itself; instead, both catalogs "subcontract" the storage function to the AFS server.

3. Multimedia Oriented Repository Environment (MORE)

As the World Wide Web (WWW) becomes very popular among internet users, an increasing number of public repositories are using the WWW to promote their services. The Electronic Library Services and Applications (ELSA) project is the operational part of the Repository Based Software Engineering (RBSE) program [25]. RBSE is a National

Aeronautics and Space Administration (NASA) sponsored program dedicated to introducing and supporting common, effective approaches to designing, building, and maintaining software systems by using existing software assets stored in a specialized library or repository.

MORE is a public domain, metadata based repository tool employing the WWW as its sole user interface. It consists of a set of application programs which operate together with a stock http server to provide access to a database of metadata. The entire interface, client browsing and searching, repository definition, data entry and other administrative functions, are provided through stock Web clients. Repository assets are classified using a collection (topic) and class (type) paradigm. According to their subject matter, they are included in the collections or subordinate collections that best represent domain coverage. The assets are also classified by media or information type through the class approach

MORE was designed to support this collection and class model. Navigation is achieved through the activation of high-level hypertext links which ultimately lead to metadata or assets themselves. Searching (Natural Language or Pattern Match) is performed against information provided in the metadata. This combination provides users with a reliable and efficient means of accessing a high volume of assets.

4. Asset Source for Software Engineering Technology (SAIC/ASSET)

Asset Source for Software Engineering Technology (SAIC/ASSET) offers products and services in digital library support, electronic commerce and software engineering with an emphasis on reengineering and reuse [26]. SAIC/ASSET, established by Advanced Research Projects Agency (ARPA) as a subtask under the Software Technology for Reliable Systems (STARS) program, is transitioning to a private enterprise as a division of Science Applications International Corporation (SAIC).

SAIC/ASSET's primary mission is to provide a distributed support system for software reuse with the Department of Defense (DoD) and to help foster a software reuse industry within the United States. SAIC/ASSET's initial and current focus is on software development tools, reusable components and documents on software development methods. The goals SAIC/ASSET are pursuing involve:

1. Creating a focal point for software reuse information exchange

2. Advancing the technology of software reuse
3. Providing an electronic marketplace for reusable software products to the evolving national software reuse industry.

To achieve these goals, SAIC/ASSET operates the Worldwide Software Reuse Discovery (WSRD) Library. The WSRD Library is populated with quality reusable software components which can be distributed to its subscribers. WSRD contains over 700 assets available to over 1500 users throughout the world. The library specializes in software lifecycle artifacts and documents written specifically to promote software reuse and development.

5. The Public Ada Library (PAL)

Since 1984, the Ada Software Repository (ASR) has been a major, publicly available source of Ada code. Now called the Public Ada Library (PAL) [27], it provides more than 100 megabytes of programs, components, tools, general information, and educational materials on Ada. It also contains materials on the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL), which is based on Ada. For those with access to the Internet, the PAL can be accessed via the File Transfer Protocol (FTP). The PAL is located on the wuarchive.wustl.edu host, and on mirror sites at ftp.cnam.fr and ftp.cdrom.com. Also, the PAL can be obtained on disk, tape, and compact-disk read-only memory (CD-ROM).

6. CAPS Software Reusable Component Repository

CAPS (Computer Aided Prototyping System) is a research project developed by the Software Engineering Group led by Prof. Luqi at Naval Postgraduate School [33]. Initial implementation of CAPS software base was first explored in 1988. An implementation of the software base was accomplished in 1991 by using ONTOS, an object oriented data base management system that provides an interface to C++ for customization and flexibility. The CAPS software base is being changed to a software component repository since 1998. The CAPS component repository supports two critical functions, component storage and component retrieval. It focuses on retrieval of assets. CAPS Repository is the only one that supports signature matching.

7. The Ada Library and the Reuse Library at the Defense Information Systems Agency (DISA):

The Ada Library and the Reuse Library at the Defense Information Systems Agency (DISA) are public, non-lending, reference libraries for all professionals, students, and researchers seeking information on the Ada programming language and on software reuse [29]. The number of books and articles on Ada and on reuse grows daily. Also, there is a wealth of information available on the Internet and on the World Wide Web. Putting the Net together with the Ada and Reuse Libraries makes a very powerful research tool. Both Libraries collect and hold information found in documents, books, conference proceedings, newspaper and journal articles, and other multimedia material.

The Libraries can provide assistance in two ways:

Helping users find publications in each library, and conducting on-line searches for published information available elsewhere. Users can access these resources in person, and via the Web, or they can call DISA to request a search.

2.3 Comparison of Various Repositories

Table 2.1: Comparison of various Repositories [30]

Features	Web-Based	Security Control	Retrieval Methods
+1 Reuse Repository		Y	Browsing
SALMS	Y	Y	Keywords
ASRR		Y	Keywords
The Universal Repository		Y	Browsing and Keywords
AIRS		Y	Facets Approach
RLT		Y	Keywords
HSTX Reuse Repository	Y	Y	Keywords
DSRS	Y	Y	Keywords
LID		Y	Keywords
I-CASE		Y	Keywords
MORE	Y	Y	Keywords
ASSET	Y	Y	Keywords
PAL	Y	Y	Keywords
CAPS			Browsing, Keywords, Profile & Signature Matching
Ada Library and Reuse Library (DISA)	Y	Y	Browsing and Keywords

As per the survey conducted by Luqi and Jiang Guo on software reuse repository [30], it has been concluded that keyword based search is the most commonly used retrieval methods and most of the repositories are web-based applications. Web-based reuse is the trend of software component repositories supported by the government. To be a part of an

integrated CASE environment is the trend of commercial software component repositories. Usually, the aim of the first one is to provide a service within a domain, organization, or area, such as ASSET for DoD, DSRS for DISA etc. This kind of repository is used in a wide scope. The aim of the second is to provide an integrated CASE environment for a software development organization. So, this kind of repository is generally a part of CASE environment and is used in a relatively narrow scope.

The long-term goal of the CAPS project [31] is to provide a distributed software component repository to support the development of prototype systems through intranet technology. So, it will combine the advantages of commercial component repositories and government supported repositories. This developing research system is an example of future software repositories.

Advantages of Software Repository:

Reuse of software assets at all phases of the software engineering life-cycle is recognized as being one of the major enablers for productivity and quality improvements.

a. Provides boost to productivity and quality.
b. Storage of data. (Cataloging, maintaining reusable assets)
c. Reusing of assets in future development
d. Increases availability of relevant components

This survey will be a base to develop future efficiently searchable, user-friendly, useful, and well-organized repositories.

Chapter 3
Problem Definition and Objective

Develop once, Use once is the common approach followed by software developers in software industry. Lot of software development efforts, cost as well as time is wasted if the software components can't be reused further. These efforts can be saved by reusing available components. In order to increase the availability of reusable components (with-reuse or for-reuse), we need an effective component storage structure i.e. Repository and also automatic component addition and extraction from Component Repository. Software repository is a set of software assets that are maintained by organization for possible browsing and/or retrieval [15]. Application of software repository is software reuse. Software reuse is the process of constructing or modifying software systems using pre-existing software assets.

It is essential to classify and retrieve reusable software component efficiently from repository, for successful development of software reuse. It is also essential that retrieved components must match user requirement, because retrieving and analyzing an irrelevant component may lead to wastage of efforts as well as time. Component storage and retrieval is a complex task, because it is difficult to classify components so that it can be easily traversed, identified and retrieved against submitted query. It all depends on how user query and components inside the repository are represented and also on retrieval mechanism. Repository enhances reusability by specifying, classifying and maintaining software components for long term perspective. Software productivity can be increased with software product reuse because reused software components need not to be developed from scratch. Software reuse can save time to market that result in larger market share. With software reuse prototypes can be developed very quickly and at very low cost as instead of developing specifications, designs and code from scratch, existing components can be reused [32]-[38]. Software reuse is a promising alternative to enhance software productivity [39]. It can improve quality and reliability of the software. Still software reuse is not in practice to its potential due to number of reasons. Foremost reason is that it requires a rich repository of quality software components. Then it is very difficult to retrieve reusable software components effectively from the repository [43]-

[48]. Software organizations face problem in practicing software reuse because of following reasons [49]-[55]:

- Object oriented analysis, CASE tools, formal methods, agile methods etc. are proposed but no clear-cut methodology for software reuse is defined [61].
- There is an absence of models and theories that can help to comment or estimate about the software without developing it.
- Rapid evolution of technologies does not allow proper assessment [52].
- Collecting context independent knowledge or representing context is very difficult [62].
- Software development for reuse especially for large complex systems is very cumbersome [63], [64].
- There is lack of empirical studies and technologies that can help to estimate what a component can perform better and what it cannot, without using it [67].

3.1 Objectives and Proposals

Software Reuse can be categorized as two major activities: Development for reuse and Development with reuse. First part of the research proposes various factors that affect reusability of component; these factors are used to describe components in the repository. This research is related to development for reuse. Following reusability factors have been identified for representing software components in repository, which plays a significant role for effective retrieval.

1. Component Name
2. Component ID
3. Component Description
4. Component Type (.exe, .dll, .doc etc…)
5. Component language in which component is developed (C, C++, VB, C# etc…)
6. Component Domain (Web-based, Desktop or Database application)
7. Required Interface name (e.g. Inumber)
8. Provided Interface name (e.g. Ifactorial)
9. Inputs required by components (no. of inputs)

10. Outputs provided by components (no. of outputs)
11. Document containing all details of components
12. Tags (Closely related keywords)
13. Inner details of source-code components:
    i. Function Name
    ii. Function's Return Type
    iii. Parameter Arguments
    iv. Parameter's Data type
    v. Storage Location

In this research work, focus has been put on developing reusable component repository in which components are described using above highlighted factors to address the main problem of classification and retrieval.

Second part of the research proposes automatic storage technique (Auto-Detect-Fragment-Store); which provides automatic functionality to Auto-Detect the function names and there details in source-code components; and Auto-Fragment their return type, parameter arguments and parameter's data type; and store these details in component repository.

Third part of research proposes a prototype which processes the user's input (component name) and automatically generates the tags (closely-related keywords) from internet web after specific interval of time and store them in dictionary at the backend. It reduces the human efforts for manually storing tags (closely-related keywords). Here tags are basically the closely related keywords which can be searched by user while retrieving software components.

Fourth part of the research proposes User Priority-Based Component Retrieval System. It is based on a specific retrieval technique in which user can specify priority i.e. user can specify the conditions on that basis components can be retrieved. It allows user to prioritize their search by pre-specified parameters.

Fifth part of research proposes most commonly used Keyword-Based Component Retrieval technique. Keyword-based search traverses data available in dictionary (which is maintained at back end and contains tags i.e. set of closely related keywords) and fetches relevant components.

3.2 Problem Statement

Every human has the limitation of inadequate memory to learn things. Similarly, user is not that expert to enter all the details of large number of components like keywords or tags or other inner technical details while storing components in repository. There is always a possibility that user may not be able to store sufficient data that may be required while traversing, identifying, selecting and retrieving component. If the system automatically store the inner details (like functions, there return type and parameters of source-code component) of component that is unknown to user and if system processes the input of user and automatically generates the tags (closely-related keywords), then the system will become automated and based on this automatic learning, system can provide more relevant components that meets user requirement.

This leads us to the problem statement and we proposed an idea by developing a prototype system, which uses Auto-Detect-Fragment-Store functionality to store inner detail of source code components and automatically generates tags (closely-related keywords) by exploring most reliable search engines like Google or other dynamic webpage links and their data; and also provides User Priority-based Component retrieval approach where users can prioritize their input as per requirement. This system is proficient of automatically storing inner detail of source-code components and all closely-related keywords in dictionary, maintained at backend which may be supportive for effective retrieval of components. It also provides alternate method of searching and selecting components by prioritizing user inputs if searching is not possible through keyword based system.

Chapter 4
Compository: The Automated Component Repository

4.1 Proposed Model

The system named "Compository" is proposed which solves all the above discussed problems. The Compository is the component repository that is capable of generating, cataloging, storing, maintaining and retrieving reusable components. The goal is to design software system for component storage in repository and retrieval of components so that they can be reused in future projects. Component storage in a repository is a challenging task because information retrieval or extracting component is all depends on how components are stored in Compository. Compository finds its suitable application in software development organizations. While software development, number of components are generated and related efforts are wasted if these components can't be reused further. These organizations can make use of its own generated components if they have a suitable storage structure for these components. Storage structure of components makes reusability effective and easy.

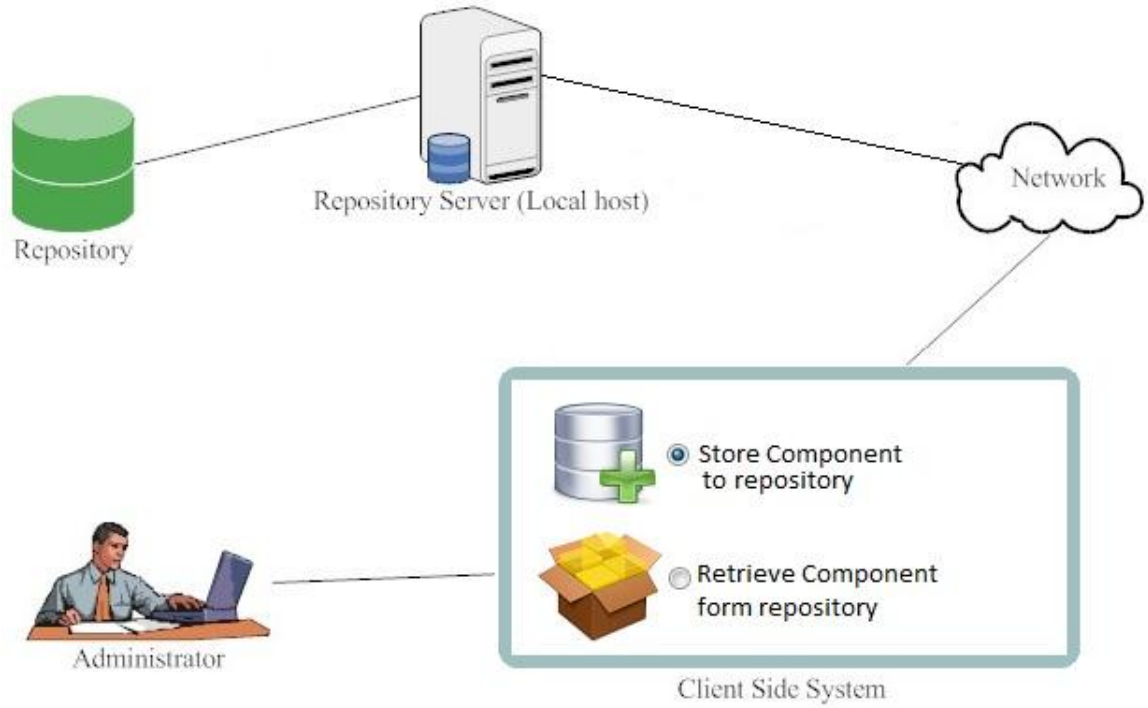

Fig. 4.1: Overview of "Compository"

The product “Compository” will run as a Web-based tool to store, add, maintain and retrieve software components like executable (.exe) components, precompiled components and documents etc. It stores the in-house developed components of the software development organization as well as COTS from vendors which can be reused in future. Here Fig. 4.1 describes the overview of Compository which includes interaction of user at client-end and with back-end. This System is centralized right now. User sends query/request through GUI at client-side which further processed on server, which responds all possible outcomes from repository.

4.2 Proposed System Architecture (Modeling & Design)

4.2.1 High Level Design of Compository

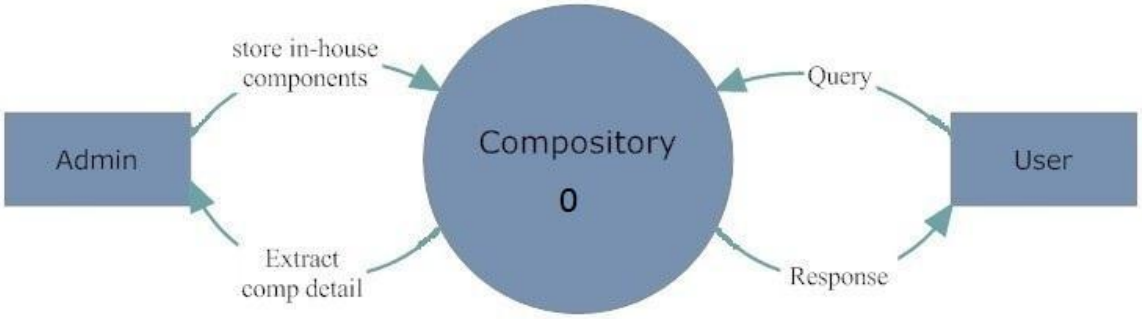

Fig. 4.2: Context level DFD- Compository

There are following types of user that can access our system:

1. Administrator:

   Administrator is responsible for storing in-house components, store COTS components if not available in-house, maintains components and repository to reuse them in future projects. Administrator can also retrieve components and its detail from repository.

2. User:

   User can browse repository and download components (as per requirements) for future use. User can download Component either by User Priority-Based Retrieval or by using Keyword-Based search.

In Fig. 4.2 admin will make a request to store a component in Component Repository. Admin can either upload or add component and its detail manually using an interface of Compository or he can use the tool provided within Compository for adding

component/details automatically. On the other side if user will make a query to extract component from system, then system will display set of all possible matches of components.

4.2.2 Level-1 DFD of Compository

In Level-1 Compository process is divided into six process and some new data stores has been identified as shown in Fig. 4.3

Following data stores has been identified:

1. Comp. Name, ID(auto-generated) <<table>>
2. Comp. Detail <<table>>
3. Comp. Fun/Location <<table>>
4. Dictionary <<table>>

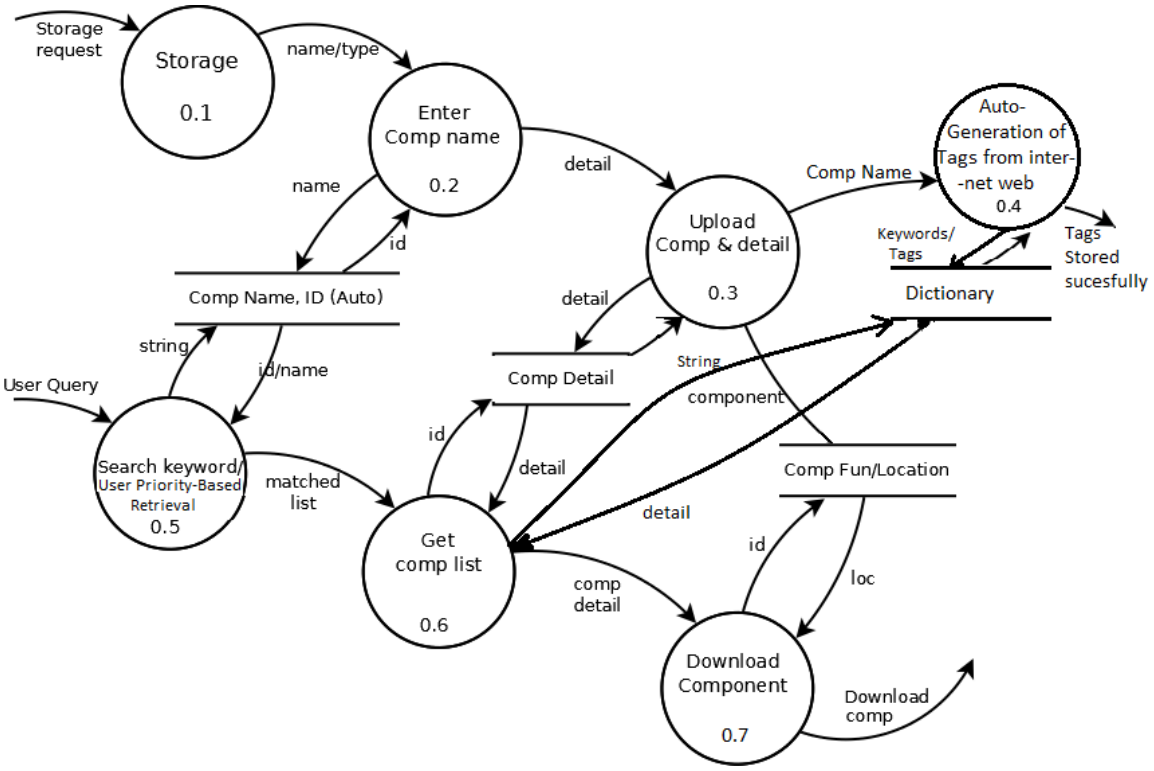

Fig. 4.3: Level-1, DFD of Compository

In Level 1 DFD, single process "Compository" at Context Level DFD is further distributed in 7 processes.

Six new processes and there functions are describes as follows:

1. Store (0.1)
2. Enter Component Name (0.2)

3. Upload Component & its detail (0.3)

4. Auto Generation of tags (0.4)

5. Search Keyword/Fixed input (0.5)

6. Get Component list (0.6)

7. Download Component (0.7)

Here admin makes a request to store Component in repository, enters Component Name, language, type and domain which is stored in "Comp Name & Id" table and CompID is generated. Component is uploaded in "Comp Fun/Location" table & details get stored in "Comp Detail" table from where Source-code component is processed and fragmentation using "Auto-Detect inner detail" process starts. "Function Name", "Return-type", "Parameter detail" is stored in "Comp Function" table and it displays successful storage message and directed to Auto-generation of tags module, and process the user input i.e. component name and explore the internet to gather closely related keywords and store them in dictionary at the back end.

Now user enters query to extract component from repository. User enters a keyword or Specific Input, which is further processed and keyword matching is done with "Comp Name/Id" table. All the possible components matches will get displayed to user. User can select and download the best component as per his/her requirement.

The Compository performs the following operations:

4.2.3 Component Storage Structure

Compository is capable of storing components in repository so that they can be reused in future projects; it provides storage structure as a repository.

4.2.3.1 Component Specifications:

A Software Component is a unit of composition with contractually-specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties [68].

Component specifications describe the functionality, properties and characteristics of a component. Fig. 4.4 describes the metadata of a software component or its various parameters that are essential for classifying software components.

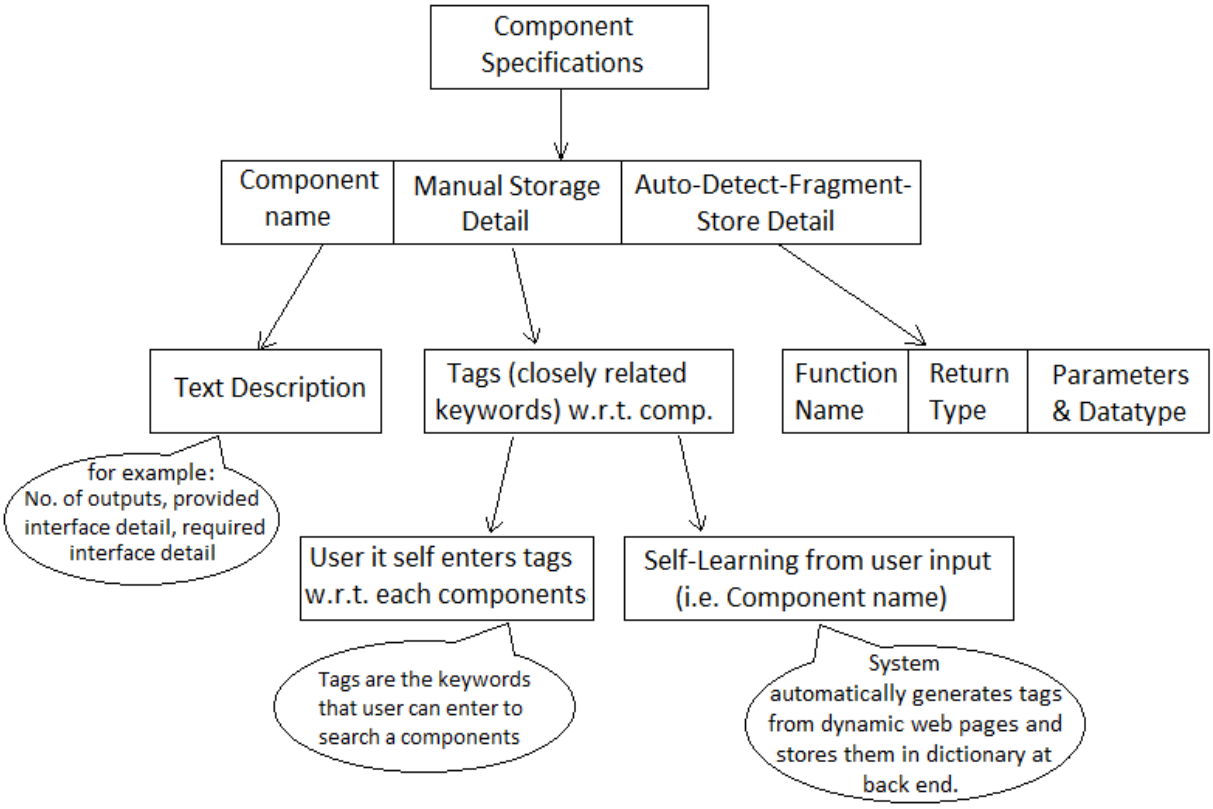

Fig. 4.4: Structure of Component Specifications

Above hierarchy defines various parameters of components that can act as fields in component repository. A Component can be defined by Component Name; Component-ID for its unique identification, its manual description that can be entered by user while adding components and Automatic storage module of repository. Manual description of components is the text description like number of inputs and outputs, provided interface detail, required interface detail etc. as described in Table 4.1.

"Tags" are the set of closely-related keywords that enhances the vocabulary of software component in repository. It can be stored manually as well as automatically by maintaining dictionary at the back end, by auto mechanism which processes the user input (i.e. Component Name) and explore its closely-related keywords on internet web. The proposed system is also capable of auto-storing inner details of source code component by using Auto-Detect-Fragment-Store technique.

Table 4.1: Component Specifications

<table>
<tr><td>Parameters</td><td colspan="2">Description</td></tr>
<tr><td>Component Name</td><td colspan="2">Name of Component</td></tr>
<tr><td>Component ID</td><td colspan="2">Unique ID number (Auto generated)</td></tr>
<tr><td rowspan="8">Manual Description</td><td>Tags</td><td>Text Description</td></tr>
<tr><td rowspan="7">Tags are the collection of closely-related keywords that user may enter while retrieving components. It enhances the vocabulary of software components. Tags can be entered manually as well as can be generated and stored automatically at back end (after specified interval).</td><td>Description about Component.</td></tr>
<tr><td>Type (EXE, DLL, DOC etc.)</td></tr>
<tr><td>Language in which component is developed. (C, C++, java, VB, C# etc.)</td></tr>
<tr><td>Web-based, Desktop or database application</td></tr>
<tr><td>Name of Required interface example: Inumber</td></tr>
<tr><td>Name of Provided interface example: Ifactorial</td></tr>
<tr><td>Inputs required by components</td></tr>
<tr><td>Outputs provided by components</td></tr>
<tr><td>Automatic storage of source code components.</td><td colspan="2">Function Name
Function’s Return Type
Parameter Arguments
Parameter’s Data Type
Storage Location</td></tr>
</table>

4.2.3.2 Database Diagram of The Compository

Database diagram describes the tables and their relationships as shown in Fig. 4.5. There are five tables which stores data about components. Comp table include a column “compid” which act as a primary key as shown above and same represent as a foreign key in other four tables. So it represents a dependency relationship of other four tables (compdetail, compfunction, stored location and dictionary) with comp table.

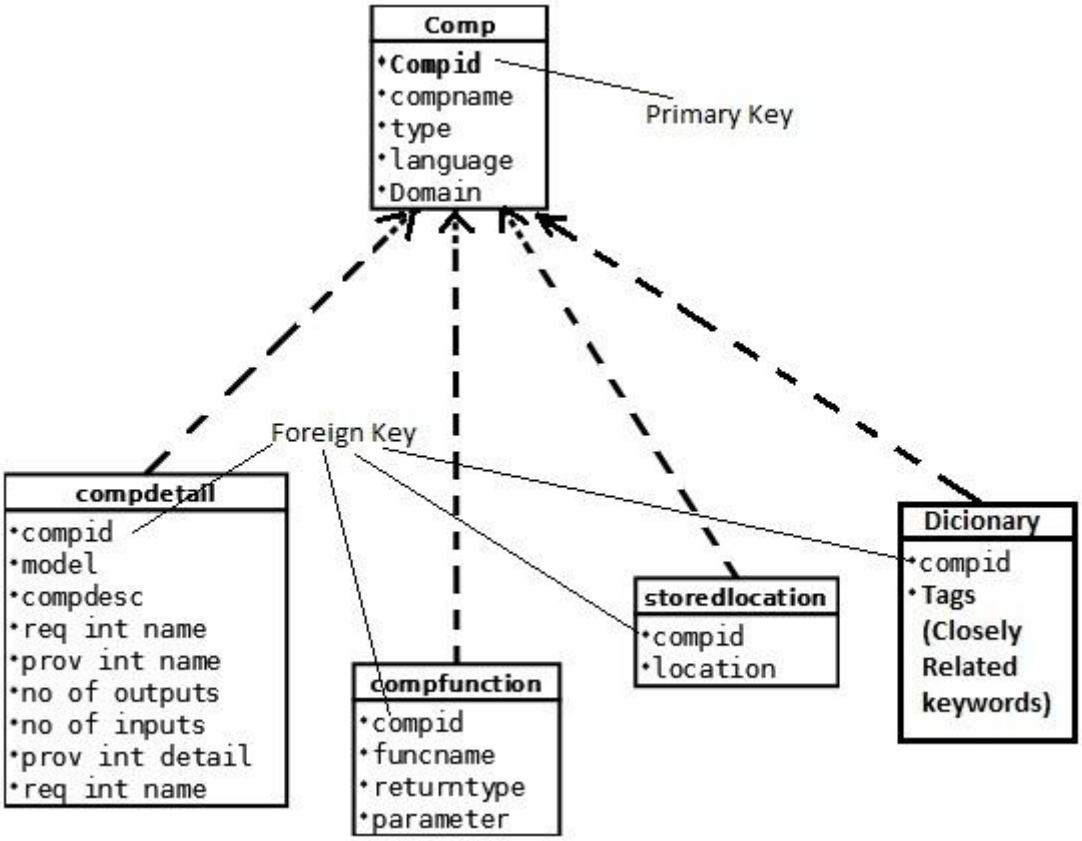

Fig. 4.5: Database diagram

4.2.4 Component Storage:

The Compository provides the functionality of storing components that are developed in-house as well that are purchased from third party vendors. There are two methods for adding component in repository:

4.2.4.1 Manual addition of components:

It provides interface to enter the appropriate detail of each component manually, for example: .exe, .dll components.

1. **Upload Component:** It provides interface to upload and store component so that it can be extracted and reused later.
2. **Store Component descriptions:** It is capable of recording component description so that user can easily find the suitable component as per his/her requirement.

4.2.4.2 Auto-Detect-Fragment-Store Components

Auto-Detect-Fragment-Store is a staged technique which is used for the automatic

storage and retrieval of source-code components. A component repository is constructed which stores automatically, fragmented code on the basis of inputs and outputs. This work is an effort towards automating component storage, its addition and retrieval in a way component can be reused and software reusability is enhanced.

1. **Upload Source-code component:** It can upload source-code component so that it can be fragmented.
2. **Auto-Detection of interfaces:** Auto-Detection of inner details of source-code component.
3. **Fragmentation of Source-code:** Fragment source-code components by Function inputs
    i. Function Name
    ii. Function Return Type
    iii. Parameter Arguments
    iv. Parameter's Data-Type
4. **Store Inner detail of Source-code:** All the fragmented details must be stored in repository so that while retrieval user can easily get & reuse the coding detail of source-code components.

4.2.4.3 Automatic Tags Generation and Storage

Tags are the closely-related keywords w.r.t. name of particular component, which can be searched by user while component retrieval. It basically enhances the vocabulary of software components. A prototype of automated system is proposed which automatically generates the closely-related keywords from most trustable search engines or dynamic links available on internet web, by processing the input from user (i.e. name of particular component). After detecting tags w.r.t. name of component system stores the same in dictionary that is maintained at the backend.

4.2.5 Component Retrieval

The Compository must be able to retrieve all the possible components so that user can select the best component as per their requirements.

4.2.5.1 User Priority-Based Component Retrieval

Compository must include functionality to retrieve components through Fixed Inputs as follows:

i. Component Type

ii. Component Language

iii. Component Domain

4.2.5.2 Keyword-Based Component Retrieval

In Keyword Retrieval user has to enter a keyword and a query string is passed to repository. After keyword matching repository provides set of possible components, from where user can select the required one.

4.3 Implementation

Implementation of Compository System contains following main modules:

4.3.1 Interface description of "The Compository''

The proposed model "The Compository" is a web-based application which provides user friendly interface as shown below; its user friendly interface provides easy access.

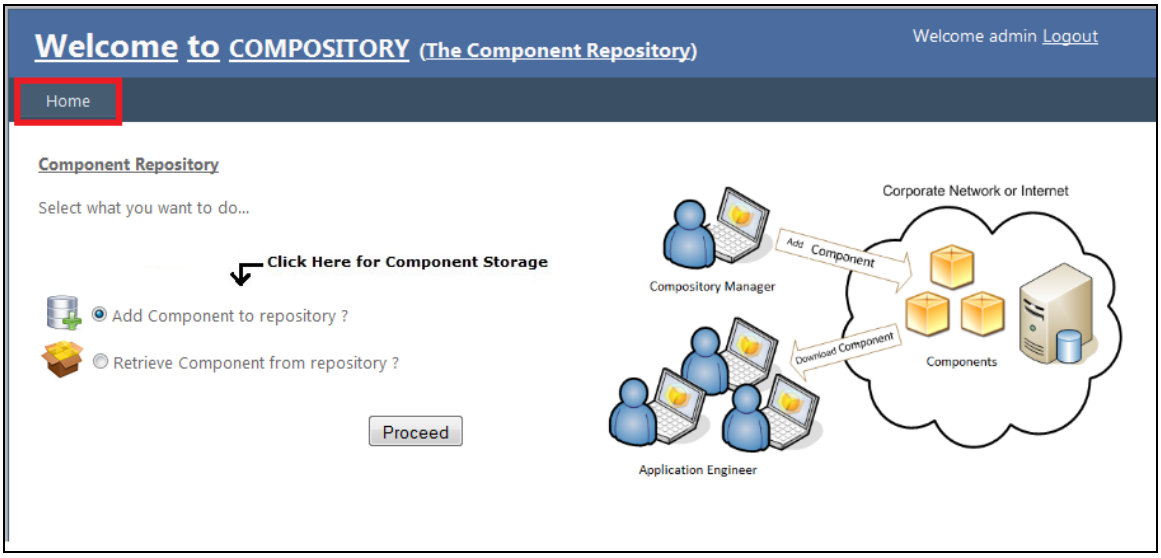

Fig. 4.6: Welcome to Compository, Select Add/Retrieval.

It has several modules; each is discussed in detail with respective interfaces.

4.3.2 Auto-Detect-Fragment-Store module

1. After login to Compository System, user is directed to Home page as shown in Fig. 4.6

2. User needs to make a selection either to store component to repository or to retrieve component from repository.

3. On click "Proceed" user is directed to storage module as shown below.

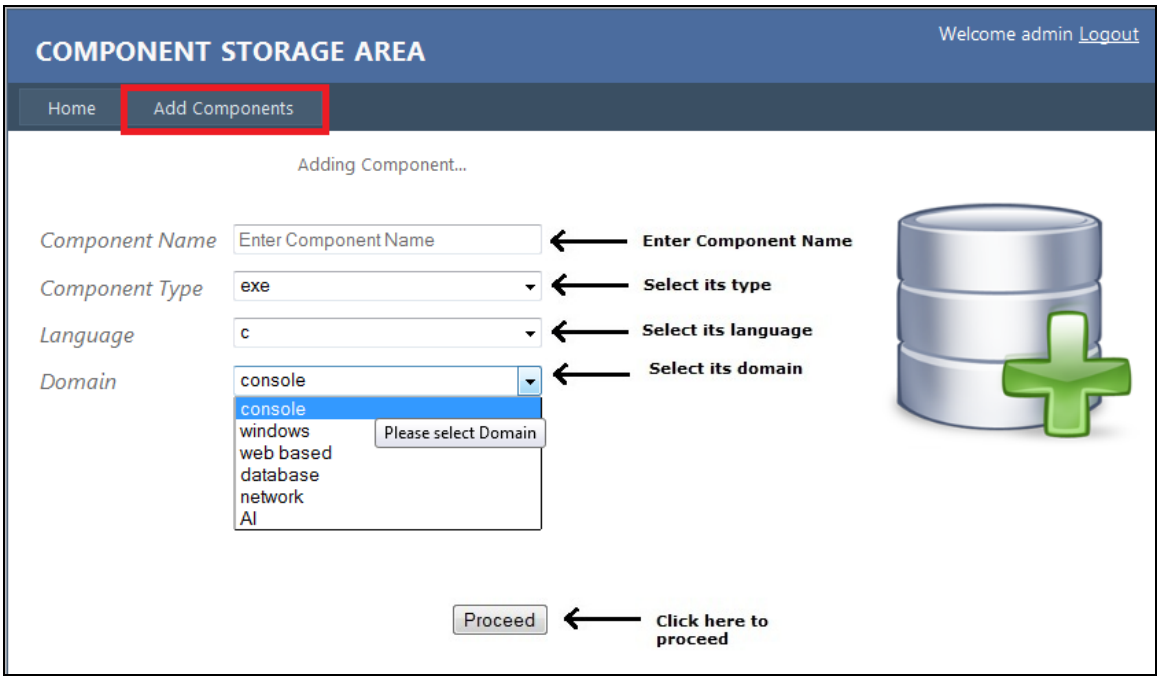

Fig. 4.7: Add Component Name

4. Here system asks to enter following as shown in Fig. 4.7.
    a. "Component name"
    b. "Component Type"
    c. "Component Language"
    d. "Component Domain"
5. On click "Proceed" user is directed to "addcompdetail.aspx" as described in Fig. 4.8
6. Here system asks to enter following:
    a. "Component Detail"
    b. "Upload Component"
    c. "Are you uploading Source-Code?"

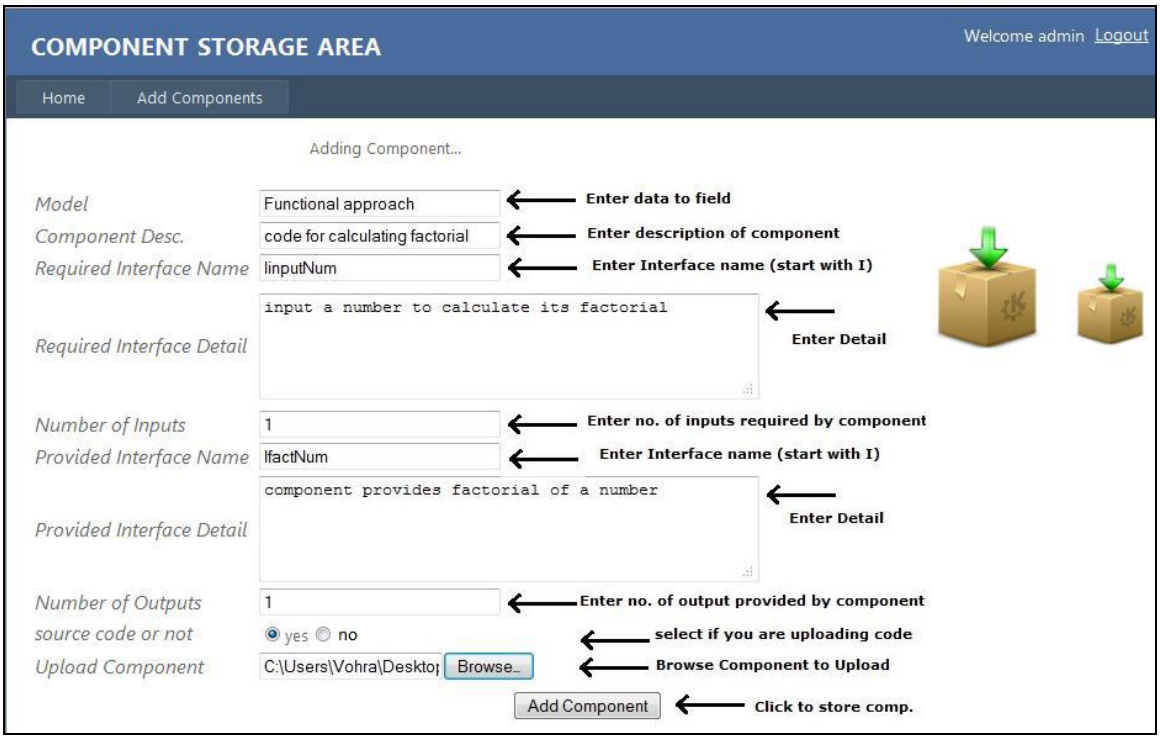

Fig. 4.8: Add and Upload component detail

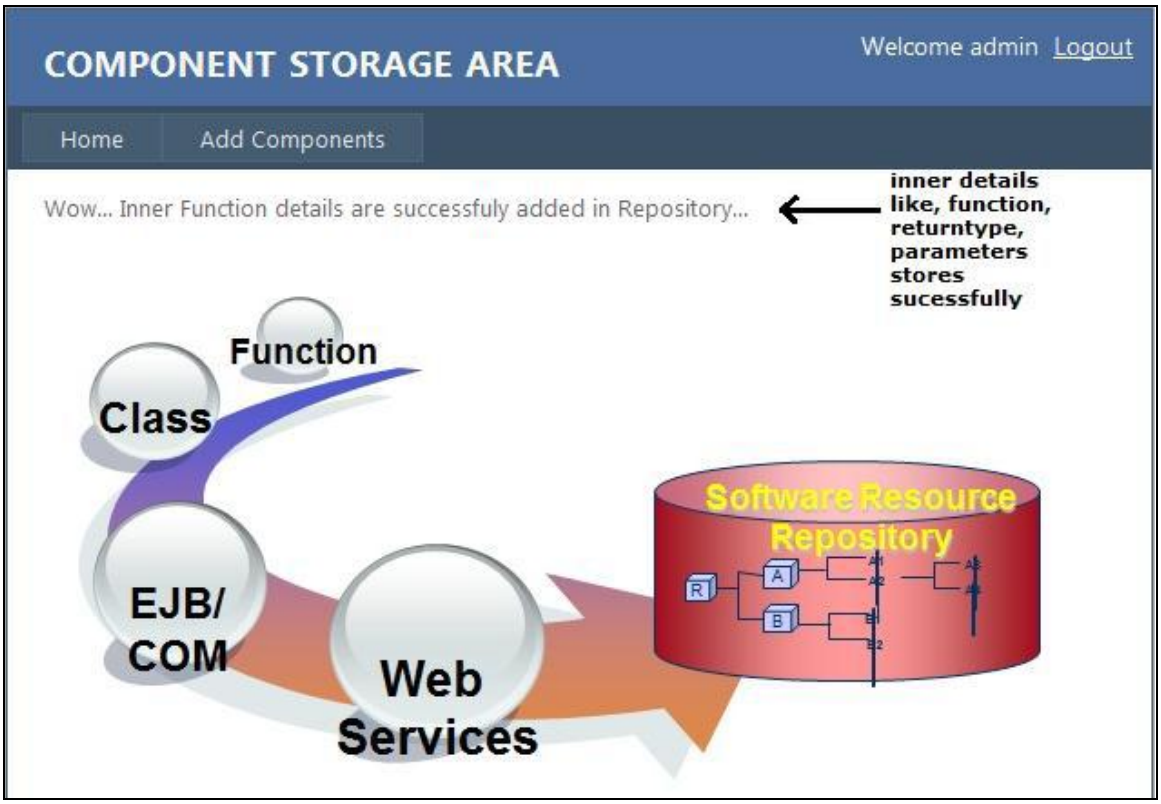

Fig. 4.9: Auto-Detect-Fragment-Store message.

7. If user select “Yes”, after proceeding system will run code in back end to Auto-Detect and Fragment the following inner detail

a. Function Name
b. Function’s Return type
c. Its Parameters
d. Parameter’s Data type

8. Finally a message will display for successful storage as shown in Fig. 4.9 above

4.3.3 Manual Storage of Component

9. But if user is not uploading a source code, then tool stores details manually (Auto-Detect-Fragment-Store will not run).

10. A success message will get displayed at the bottom of same page as in Fig. 4.10

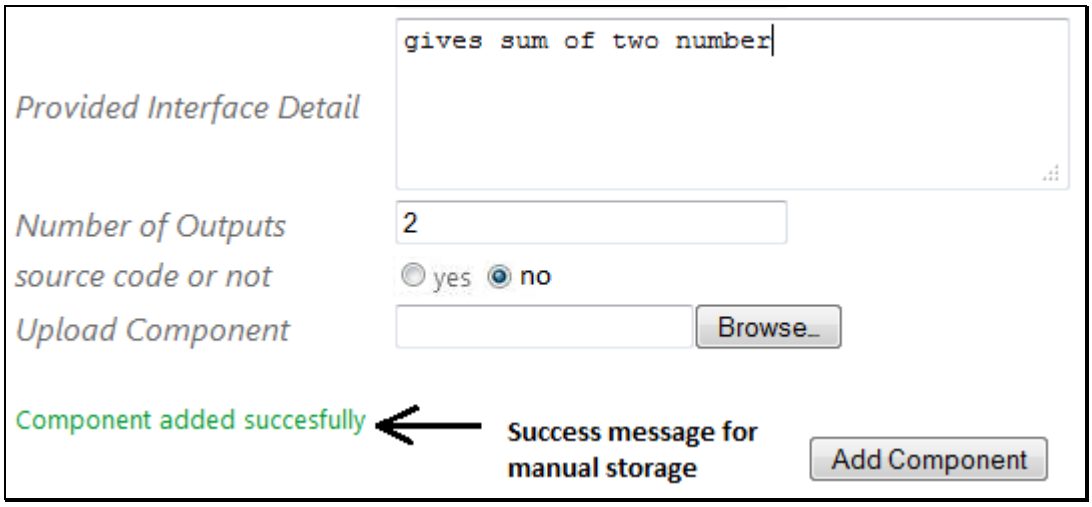

Fig. 4.10: Successful Storage message if component is not source-code.

4.3.4 Auto-Generation of Tags

11. After uploading the components and its detail, system asks for storing tags (closely related keywords) as shown below in Fig. 4.11.

12. On single click of “Add keywords” button, system processes the user input i.e. component name and explore it on internet web and starts finding closely-related keywords using most trustable and reliable search engines link Google and first few dynamic links produced by www.google.com .

Fig. 4.11: Adding tags to dictionary of repository.

13. It processes the user input at backend and gather all the closely related keywords for internet web, and stores in dictionary maintained at backend as shown in Fig. 4.12. Logic for Backend implementation of Auto-Generation of Tags is mentioned in Appendix-I.

compid	keyword	did
216	Input driver	271
216	Mouse driver	272
216	Network device...	273
216	Printer driver	274
216	Patch	275
216	Software definit...	276
216	Class driver	277
216	Device driver sy...	278
216	Firmware	279

Fig. 4.12: Auto-generated tags from internet web

4.3.5 User Priority-Based Component Retrieval

In User Priority-Based Component retrieval user can assign their priorities by checking and unchecking the check box available at user interface. System also provides the functionality to make choices from dropdown as shown in Fig. 4.13.

1. User checks the priorities, with which user want to retrieve components.
2. On checking the check boxes system automatically displays the dropdown icon in from of checkbox and asks for making choice from dropdown.

On click of “retrieve button” system will display all the possible matches w.r.t. to user’s priority.

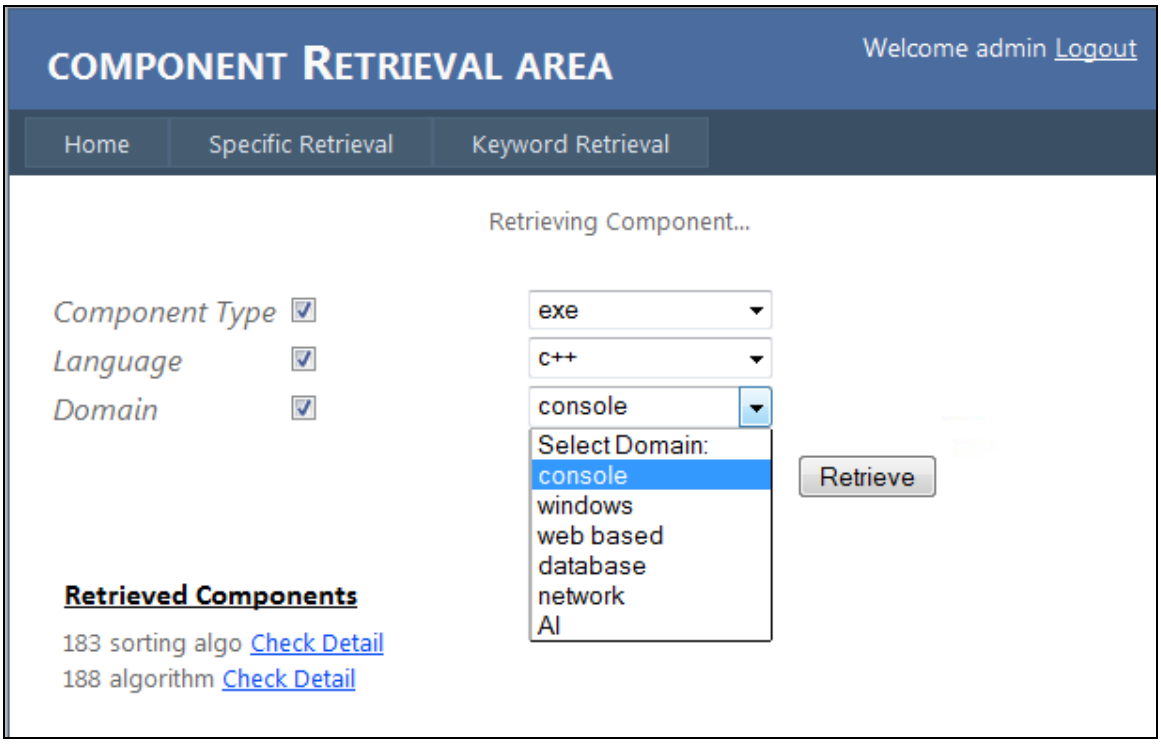

Fig. 4.13: User Priority-Based Component Retrieval interface

4.3.6 Keyword-Based Component Retrieval

1. If user selects component retrieval on Home Page as shown in Fig. 4.14, then user is directed to component retrieval area as shown in Fig. 4.15.

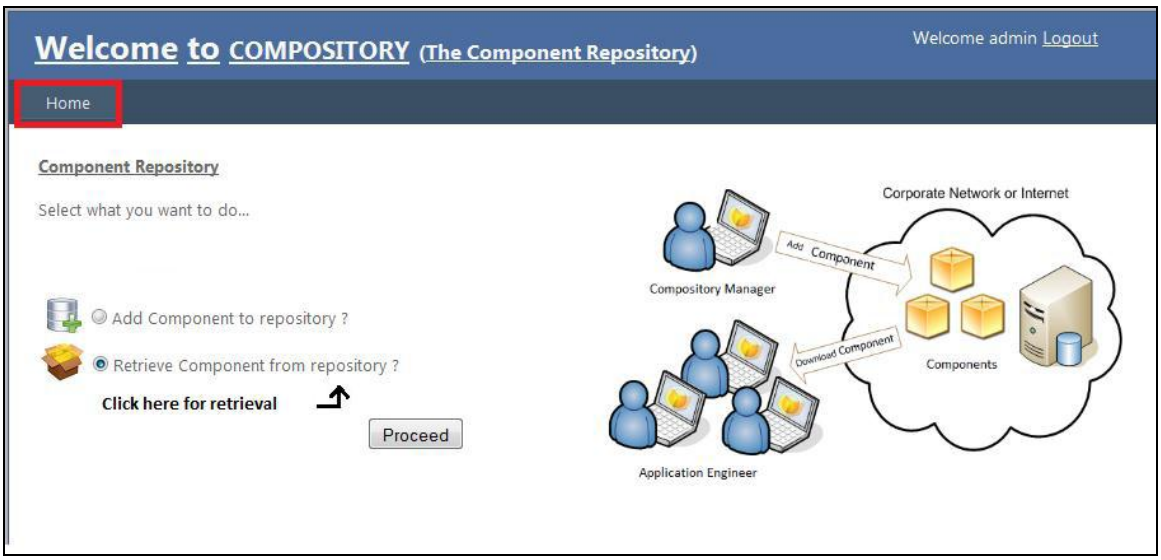

Fig 4.14: Component Retrieval interface

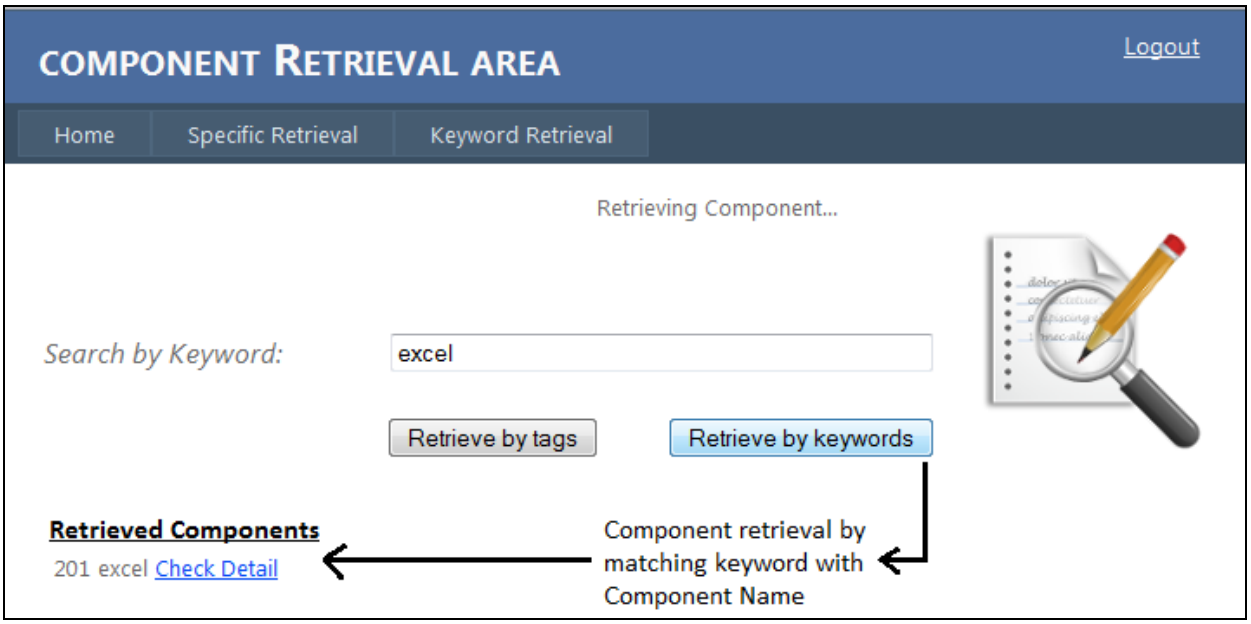

Fig. 4.15: Component retrieval by keyword based search (with component name)

2. On selecting Keyword Retrieval user need to enter keyword to search component. Keyword based search is the most common technique for searching and extracting components as discussed in [5][6].

We can discuss this flow of retrieval with the help of an example using below mentioned tables in Fig. 4.16 and Fig. 4.17 at back end. Example: Suppose, goal of a user is to search sorting algorithm which uses divide and conquer approach. Say user enters “divide and conquer algorithm”. Then it will first search the query against "compname" in table shown in Fig. 4.16, but there is no such component, then it'll extend its search to dictionary [dictionary is a table in which user can store the keywords or set of keywords that specify individual component in repository] as shown in Fig. 4.17. Here it matches the tags against query and finds the relevant match with "compid = 153". It will find the corresponding component of “compid = 153” in Fig. 4.16 and extract the component having name "Program for merge sort". Finally “Program for merge sort” is retrieved which is relevant to user query and goal. Hence user can retrieve the required component.

compid	compname	type
150	factorial of a number	source code
151	linklist insertion program	source code
152	Sum of two numbers	exe
153	Program for merge sort	source code

Fig. 4.16: Table containing detail of each component ("compid" acts as Primary key in this table)

compid	tags
150	calculate factorial, sourcecode, algorithm
151	linklist algorithm, node insertion, at end, at given position
152	adding 2 numbers, upto three digit number
153	sorting algorithm, divide and conquer

Fig. 4.17: Dictionary table containing tags or keywords for corresponding component ID ("compid" acts as Foreign key in this table)

3. User enters the keyword "excel" and select retrieve by keywords, then system will traverse the component names and displays/retrieve the component when found the exact match as shown in Fig. 4.18.

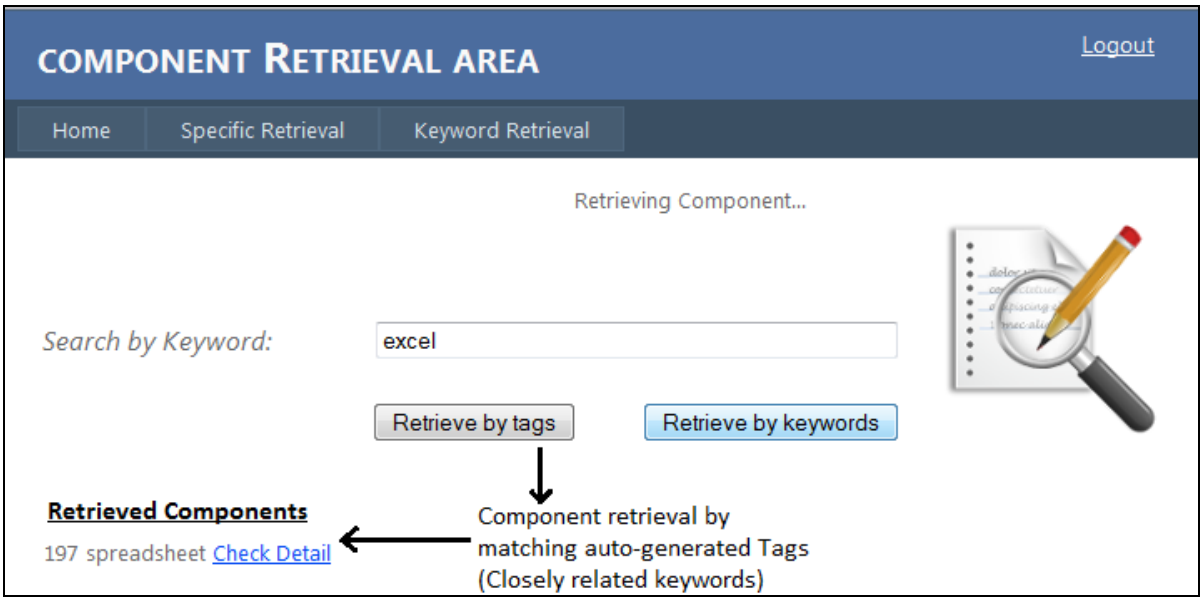

Fig. 4.18: Component retrieval by Tags (Closely-Related keywords)

4. If user clicks on "Retrieve by tags" then system will traverse the dictionary i.e. maintained at the back end, and matches the user input with auto-generated tags and

display the relevant component with unique component id as shown below. Here, system displays the "spreadsheet" component i.e. is relevant with user input.

4.3.7 Downloading Component & its Details

5. User can check detail of selected component on click "Check Detail" and user is directed to "compdetail.aspx" page.

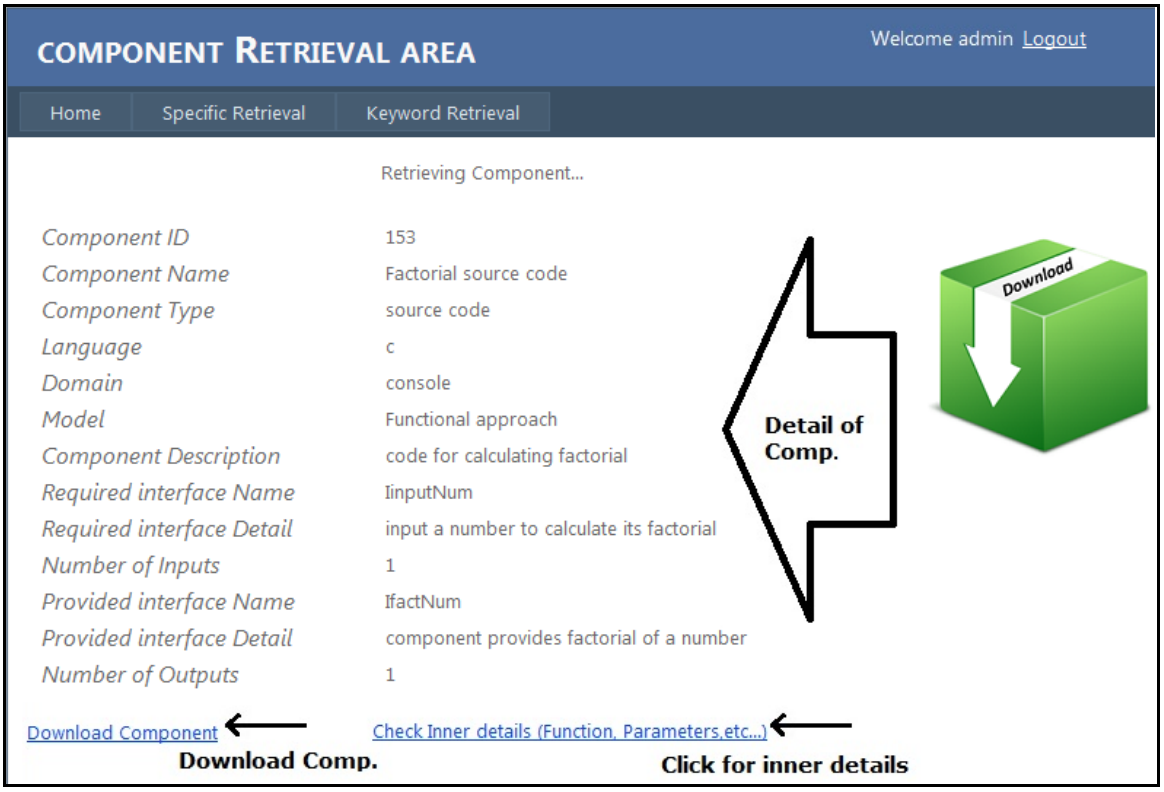

Fig. 4.19: Component Detail page/download component

6. Here User can download component as shown in Fig. 4.19 or can check following inner details for Source-code components as shown in Fig. 4.20.

a. "Function Name"
b. "It's ReturnType"
c. "Parameter & datatypes"

7. This system Auto-Detect the inner details of Source-Code components. These details are very helpful for reusing the same component in another project.

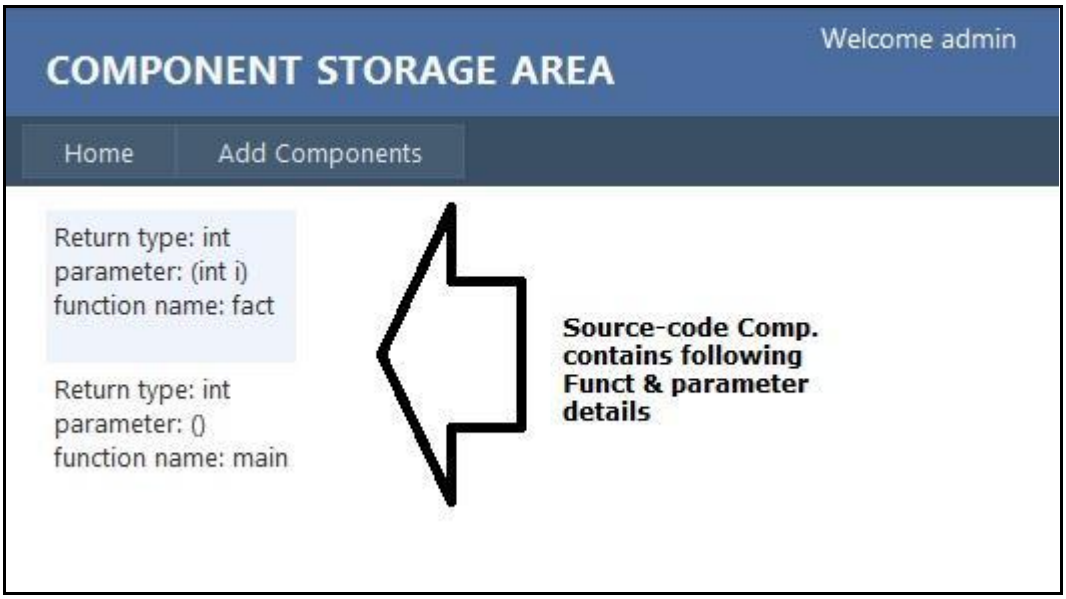

Fig. 4.20: Inner details of Source-code component "Auto-Detect method".

Chapter 5
Conclusions & Future Scope

This chapter concludes the work done in thesis and discusses future scope of the work. The main objective of the thesis is to design and develop an effective reusable software component repository. Reusable software component may include source-code, algorithm, executable code, test cases, test plans and documentation. Developing a software component from scratch takes lot of efforts and development time as well; these efforts can be saved by reusing the pre-existing components. It's difficult to specify a component in the repository so that it can be easily traversed and retrieved efficiently. An attempt has been made in this thesis to specify components in repository in such a way that helps in retrieving the components. In fourth chapter automatic storage technique "Auto-Detect-Fragment-Store", "Auto-generation of tags" has been proposed. This reduces the efforts made by user while storing software components; and Retrieval methods "User Priority-Based component retrieval" and "Keyword based retrieval" has been proposed which asks user priority to search and select desired component.

5.1 Conclusions

"The Compository" is an effort towards Automatic Component Storage and Extraction. As Compository finds its scope of application in any software development organization interested in reusing components, it is capable of generating, retrieving, storing and maintaining components in effectively designed component repository.

1. The main function of Compository i.e. "Auto-Detect-Fragment-Store" practically elaborates different components when in bulk have to be added in Component Repository. Its special technique efficiently & automatically stores in-house developed as well as off the shelf components (COTS) in component repository. Source-code components can be fragmented on following basis: Below discussed inner detail possibly shows input/output interfaces respectively.
    i. Function Names
    ii. Function's Return-type

iii. Parameter Arguments list
iv. Data-type of arguments

2. Another important functionality i.e. “Auto-Generation and Storage of Tags” using automation technique, uses most reliable and trustable search engines Google and its API service to generate dynamic links to access and gather closely-related keywords to store in dictionary that is maintained at back end. It enhances the vocabulary of software components in dictionary and reduces human efforts and human error by processing user input.
3. Components can be retrieved either by “Keyword-based Component Retrieval” or by “User Priority-Based Retrieval approach.” User Priority-Based Retrieval includes following inputs and can be prioritized by user:
   i. Component Type
   ii. Component Language
   iii. Component Domain

5.2 Future Scope

This work opens up on a number of avenues for future research. The work can be extended in several directions and some are summarized below:

1. Auto-generation of tags can be applied on large number of dynamic links to reduce the human efforts and error, which enhances the possibility of getting most relevant software components.
2. Future work can be extended on automatic storage of different types of components like test cases, test plans, UML diagrams as it is only deployed on Source-code. It can be also include enhancement on retrieval methods, in order to retrieve best components from repository which best suits user requirements.

References

1. B. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, vol. 21, no. 5, pp. 61-72, May 1988.
2. V. R. Basili and R. Reiter, "A Controlled Experiment Quantitatively Comparing Software Development Approaches". *IEEE Trans. on Software Engineering*, vol. 7, pp. 299-320, 1981.
3. C. Larman & V. R. Basili "Iterative and Incremental Development: A Brief History," *IEEE Software*, vol. 20, pp. 47-56, 2003.
4. H. D. Benington, "Production of Large Computer Programs," *Ann. of the Hist. of Comput.*, vol. 5, 4 (4), pp. 350-361, October, 1983.
5. W. W. Royce, "Managing the Development of Large Software Systems," *The proceedings of the WESCON*," San Francisco, IEEE CS, pp. 328-339, 1970.
6. M. Lycett, R. D. Macredie, C. Patel and R. J. Paul, "Migrating Agile Methods to Standardized Development Practice," *Computer*, Vol. 36, 6 (6), pp. 79-85, June, 2003.
7. C. Larman, & V. R. Basili, "Iterative and Incremental Development: A Brief History," *IEEE Software*, Vol. 20, pp. 47-56, 2003.
8. C. Larman, "Agile and Iterative Development: A Manager's Guide," *Pearson Education, Inc. Boston*, p. 342, 2004.
9. T. Gilb, "Evolutionary Development," *ACM SIGSOFT Software Engineering Notes*," vol. 6, no. 2, p. 17, April, 1981.
10. T. Gilb, "Principles of Software Engineering Management," *Addison-Wesley*, Wokingham, UK, p. 464, 1988.
11. T. Gilb, "Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage," Butterworth-Heinemann, 2005, p. 480.
12. K. Beck, "Extreme Programming Explained: Embrace Change," *Addison Wesley Longman*, Inc. 190 p, 2000.
13. L. Williams and A. Cockburn, "Agile Software Development: It's about Feedback

and Change," *IEEE Computer Society*, Vol. 36, June, 2003. pp. 39-43.

14. R. S. Pressman, *"Software Engineering-A Practitioner's Approach,"* New York, McGrawHill Int, Ltd., 2010. p. 847.
15. Mili, Mili, Yacoub and Addy Edward, *"Reuse-Based Software Engineering,"* AWiley-Interscience Publication, John Wiley & Sons, INC. 2002. pp. 445, pp. 540 and pp-7.
16. "+1 Software Engineering Corporate Mission," Available: http://www.plus-one.com/company.html
17. M. Elisabetta, "SALMS v5.1: A System for Classifying, Describing, and Querying about Reusable Software Assets", The Proc. of 5th *Int. Conf. on Software Reuse*, 1998.
18. *"Project Management Tool Suite System (Automated Software Reuse Repository),"* Available: http://wv.ewa.com-/srr_overview.html
19. *"Universal Repository,"* Available: http://www.marketplace.unisys.com/urep/
20. *"Reuse Library Toolset of EVB Software Engineering,"* Available: http://gopher.metronet.com:70/0/newprod/by-vendor/E-/evb_software_e/941208.01
21. *"The HSTX Software Reuse Repository,"* Available: selsvr.stx.com/~eryq/swreuse/home.html
22. "*DSRS - Defense Technology for Adaptable, Reliable Systems*," Available: http://ssed1.ims.disa.mil/srp/dsrspage.html
23. *"STARS - Software Technology for Adaptable, Reliable Systems,"* Available: http://www.stars.ballston.paramax.com/index.html
24. D. Garlan, "Research Directions in Software Architecture", *ACM Computing Surveys*, 27(2), June 1995.
25. "ELSA - Electronic Library Services & Applications," Available: http://rbse.mountain.net/ELSA/
26. "*ASSET - Asset Source for Software Engineering Technology"* Available: http://source.asset.com/asset.html
27. *"PAL - Public Ada,"* Available: http://web.cnam.fr/Languages/Ada/PAL/
28. Luqi and M. Ketabchi, *"A Computer-Aided Prototyping System,"* IEEE Trans. on

Software Engineering, October, 1988.

29. *"STARS Q9 BASELINE Ada LIBRARY, Technical Reports on the Software Reuse*" CFCSE-IC Available: http://diisw.ncr.disa.mil/ReuseIC/guidelines/ReusabilityGuidelines.html
30. Luqi and Jiang Guo, "A Survey of Software Reuse Repositories", Research supported by ARO (38690-MA) and DARPA(99-F759).
31. Luqi and Jiang Guo, "Toward Automated Retrieval for a Software Component Repository", Proceedings of *IEEE Int. Conf. and Workshop on the Engineering of Computer Based Systems*, Nashville, USA (IEEE ECBS), 1999, March 7-12, pp. 99-105.
32. M. David Balda and A. David Gustafson, "Cost-estimation models for the reuse and prototype software development lifecycles", *ACM SIGSOFT*, July 1990, pp. 42-50,
33. R. G. Fichman and C. F. Kemerer, "Incentive Compatibility and Systematic Software Reuse", *J. of Systems and Software*, vol. 57 no. 1, pp 45-60, 2002.
34. J. E. Gaffney and R. D. Cruickshank, "A General Economics Model of Software Reuse," in proc. of the 14th *Int. Conf. on Software Engineering*, ACM Press, Melbourne, Australia, May 1992. pp 327-337.
35. J. R. Herrero and J. J. Navarro, "Building Software Via Shared Knowledge", Proc. of *Int. Conf. on Software Engineering Research and Practices*, June 2003, pp. 861-870.
36. J. A. Goguen, "Reusing and Interconnecting Software Components," *IEEE Computer*, pp 16-28, February 1986.
37. P. Kruchten, E. Schonberg, and J. Schwartz, "Software prototyping using the SETL programming language," *IEEE Software*, vol. 1, pp. 66-75, October 1984.
38. Ravichandran T. and Marcus A. Rothenberger, "Software Reuse Strategies and Component Markets", ACM Transactions on Software Engineering, vol. 46 no. 8, pp 109-114, August 2003.
39. Gerhard Fischer, "Cognitive View of Reuse and Design." IEEE Software, pp 60-72, July, 1987.
40. G. Caldiera and V. R. Basili, "Identifying and Qualifying Reusable Software

Components", *IEEE Computer* vol. 24, pp. 61-70, February 1991.

41. T. C. Jones, "Reusability in Programming: A Survey of the State of the Art," *IEEE Trans. on Software Engineering*, vol. 10 no. 5, pp 488-494, September 1984.
42. S. N. Woodfield, D. W. Embley, and D. T. Scott, "Can Programmers Reuse Software," *IEEE Software*, pp 152-59, July 1987.
43. H. Mili et al., "Intelligent Component Retrieval for Software Reuse," proc. of the Third *Maghrebian Conference on Artificial Intelligence and Software Engineering*, Rabat. Morocco, April, 1994.
44. J. M. Conrad et al., "Using a New Software Product Development Process for a Code Reuse Project," proceedings of the 1999 *Engineering of Computer-Based Systems Conf.*, Nashville, TN, March 1999, pp 34-40.
45. T. N. R Shetty, et al. "Hybrid method for knowledge processing, integration and representation", *Int. Conf. on Information Reuse and Integration, Proc. IEEE* , Waikoloa, Hawaii, USA, 2006, pp 45-50.
46. Y. Smaragdakis and D. Batory, "An Object-Oriented Implementation Technologies for Refinement and Collaboration-Based Designs", *ACM Trans. on Software Engineering and Methodolgy*, vol. 11 no. 2, pp 215-255, April 2002.
47. M. Steinbach and V. Kumar. (2006, October 3), "Generalizing the Notion of Confidence", *Knowledge and Information Systems*, [online].
48. B. F. William and B.A. Nejmeh, "An Information System for Software Reuse", *Software Reuse: Emerging Technology*, *IEEE Computer Society Press*, pp 142-151, 1990.
49. E Y. Wang et al., "Formalizing and Integrating the Dynamic Model within OMT", *Proc. of IEEE International Conference on Software Engineering* (ICSE97), pp 45-55, Boston, Massachusetts, May 1997.
50. M. H. Hisham and E. Biberoglu, "Component-Based Software engineering: Issues and Concerns", *Proc. of Int. Conf. on Software Engineering Research and Practices*, June 2003, pp 391-397.
51. G. Jones, (October 1999) "*Object Store 6.0*, [White Paper], Butler Group, Available: http://www.objectdesign.com/whitepapers/objectstore.ht
52. G. Kotonya, and A. Rashid, "A Strategy for Managing Risk in Component- Based

Systems", *proc. of IEEE 26th Euromicro Conference*, Warsaw, Poland, 2001, pp. 12-21.

53. G. Kotonya et al., "COTS-Based System Development: Processes and Problems", *proc. of 4th Int. Information Society Conference: Development and Reengineering of Information Systems*, Ljubljana, Slovenia, October 2001.
54. P. Kruchten et al., "Software prototyping using the SETL programming language," *IEEE Software*, vol. 1, October 1984, pp. 66-75.
55. M. V. Mauco et al. "Formalising a Derivation Strategy for Formal Specifications from Natural Language Requirements Models," *proc. Int. Symposium on Signal Processing and Information Technology, IEEE,* 2005 pp. 646-651.
56. I. Crnkovic and M. Larsson, "A Case study: Demands on Component-based Development", Proc. of *22nd International Conference on Software Engineering*, ICSE, Ireland, 2002, pp 22-30.
57. E. Horowitz and J. B. Munson, "An Expansive View of Reusable Software," *IEEE Trans. on Software Engineering*, vol. 10 no. 5, pp 477-487, 1984.
58. W. Emmrich and N. Kaveh, "Component Technologies: Javabeans, COM, CORBA, RMI and CORBA component model", *Int. Conf. on Software Engineering*, ICSE, Orlendo, Florida, 2002.
59. E. Wang and B. Cheng, "A Rigorous Object-Oriented Design Process", Department of Computer Science, Michigan State University, East Lansing, Michigan, Tech. Rep, MSUCPS: TR97-146, 1997.
60. Kang B. Y., Kim D. W. and Lee S. J., "Exploiting concept clusters for content-based information retrieval," *Int. J. of Information Sciences—Informatics and Computer Science*, vol. 170, no. 2-4, 2005.
61. S. Kumar et al. "An improved Type-Inference Algorithm to expose parallelism in OO Programs", in *Languages, Compilers and Run-Time Systems for Scalable Computers,* Boleslaw K. Szymanski and Balaram Sinharoy (Editors), Boston, MA, Kluwer Academic Publishers, 1995 Ch. 22, pp. 283–286.
62. J. M. Voas, "The Challenges of Using COTS Software in Component-Based Development", *IEEE Computer*, vol. 31, pp. 44, 1998.
63. E. Horowitz and J. B. Munson, "An Expansive View of Reusable Software," *IEEE*

Trans. on Software Engineering, vol. 10 no. 5, pp. 477-487, 1984.

64. M. Evered et al., "Software Reuse in an Object Oriented Framework: Distinguishing types from Implementation and Objects from Attributes", 6th *Int. Conf. on Software Reuse*, Vienna, 2000, pp. 420-435.
65. L Zhang et al. "A novel approach of components retrieval in large-scale component repositories," 3rd *Int. Conf. on Software Engineering and Service Science (ICSESS)*, IEEE, Beijing, 2012, pp. 220 – 223.
66. C. H. Chang et al. "XML-Based Reusable Component Repository for Embedded Software," *Computer Software and Applications Conference Workshops* (COMPSACW), IEEE, Munich 2011, pp. 345-350.
67. R. Prieto-Díaz, "Software Reuse: Issues and Experiences", *American Programmer*, vol.6, no.8, pp. 10-18, April 1993.
68. I. Sommerville, "*Software Engineering*", 9th edition.
69. Grady Booch et al. "The Unified Modeling Language User guide", 2005

List of Publications

1. P. Vohra and A. Singh, "A Contrast and Comparison of Modern Software Process Models," in *Int. Conf. on Advances in Management and Technology*, Patiala, Pb., 2013, pp. 23-27.
2. P. Vohra and A. Singh, "Automatic Fragmentation and Storage of Code in Component Repository w.r.t. their Input and Outputs Interfaces: A Tool," *"Int. J. of Innovative Technology and Exploring Engineering*, vol. 2, no. 3, pp. 235-238, 2013.
3. P. Vohra and A. Singh., "A Tool incorporation different Techniques for Effective Component Storage and Retrieval," *Int. J. of Scientific and Engineering Research*, vol. 4, no. 7, pp. 1134-1137, 2013.
4. P. Vohra and A. Singh, "User Priority Based Component Retrieval Tool: Component Retrieval by Prioritizing User Selection and Maintaining Dictionary in Component Repository," *Int. J. of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 5, 2013.

Appendix-I

Pseudo Code for backend implementation of Auto-Generation of Tags

```
/// <summary>
/// Summary description for Class1
/// </summary>
public class Class1
{
   public static List<string> wikiresults = new List<string>();

   public static cn.DataClasses1DataContext objec = new cn.DataClasses1DataContext();

   public void wiki(string tosearch,string compid)
   {
        GwebSearchClient client = new GwebSearchClient("http://www.google.com");
        IList<IWebResult> results = client.Search(tosearch + " wiki", 1);

        foreach (IWebResult result in results)
        {
           Console.WriteLine("[{0}] {1} => {2}", result.Title, result.Content, result.Url);

      // used to build entire input
                StringBuilder sb  = new StringBuilder();

                // used on each read operation
                byte[] buf = new byte[8192];

                // prepare the web page we will be asking for
                HttpWebRequest  request  = (HttpWebRequest)
```

```
            WebRequest.Create(result.Url.ToString());

        // execute the request
        HttpWebResponse response = (HttpWebResponse)
            request.GetResponse();

        // we will read data via the response stream
        Stream resStream = response.GetResponseStream();

        string tempString = null;
        int    count      = 0;

        do
        {
            // fill the buffer with data
            count = resStream.Read(buf, 0, buf.Length);

            // make sure we read some data
            if (count != 0)
            {
                // translate from bytes to ASCII text
                tempString = Encoding.ASCII.GetString(buf, 0, count);

                // continue building the string
                sb.Append(tempString);
            }
        }
        while (count > 0); // any more data to read?


int index = sb.ToString().LastIndexOf("See also\">edit");
```

```
//List<string> list = GetLinks(sb.ToString());
int min=0, max=0;
string final = sb.ToString().Remove(0, index - 1);

//Console.WriteLine(final);
    string[] final1 = final.Split('<');

    for (int i = 0; i < final1.Length; i++)
    {
       if (final1[i].Equals("ul>\n"))
       {
          min = i;
       }
       if(final1[i].Equals("/ul>\n"))
       {
          max = i;
          break;
       }
    }

    for (int j = min + 1; j < max; j++)
    {
       if(final1[j].StartsWith("a"))
       {
          string[] arr1 = final1[j].Split('>');
          //string temp = arr1[1].Remove(arr1[1].Length-1, 1);
          wikiresults.Add(arr1[1]);
          en.dic objs = new en.dic();
          objs.compid = int.Parse(compid);
          objs.keyword = arr1[1];
```

```
                objec.dics.InsertOnSubmit(objs);
                objec.SubmitChanges();
                //code for indert in dic table
            }
          }
        }
    }

    public void comp(string tosearch1,string compid)
    {
        GwebSearchClient client1 = new GwebSearchClient("http://www.google.com");
        IList<IWebResult> results1 = client1.Search(tosearch1 + " computerhope.com", 1);

        foreach (IWebResult result1 in results1)
        {
            Console.WriteLine("[{0}] {1} => {2}", result1.Title, result1.Content,
result1.Url);

            // used to build entire input
            StringBuilder sb = new StringBuilder();

            // used on each read operation
            byte[] buf = new byte[8192];

            // prepare the web page we will be asking for
            HttpWebRequest request = (HttpWebRequest)
                WebRequest.Create(result1.Url.ToString());

            // execute the request
            HttpWebResponse response = (HttpWebResponse)
                request.GetResponse();
```

```
// we will read data via the response stream
Stream resStream = response.GetResponseStream();

string tempString = null;
int count = 0;

do
{
   // fill the buffer with data
   count = resStream.Read(buf, 0, buf.Length);

   // make sure we read some data
   if (count != 0)
   {
      // translate from bytes to ASCII text
      tempString = Encoding.ASCII.GetString(buf, 0, count);

      // continue building the string
      sb.Append(tempString);
   }
}
while (count > 0); // any more data to read?

// print out page source

int index = sb.ToString().LastIndexOf("Also see:");

int min = 0, max = 0;
string final = sb.ToString().Remove(0, index);
```

```
        string[] final1 = final.Split('<');

        for (int i = 0; i < final1.Length; i++)
        {
            if (final1[i].Contains("bottomad"))
            {
                max = i;
                break;
            }
        }

        for (int j = min; j < max; j++)
        {
            if (final1[j].StartsWith("a"))
            {
                string[] arr1 = final1[j].Split('>');
                //string temp = arr1[1].Remove(arr1[1].Length-1, 1);
                if (!arr1[1].Contains("definitions"))
                    wikiresults.Add(arr1[1]);
                en.dic objs = new en.dic();
                objs.compid = int.Parse(compid);
                objs.keyword = arr1[1];
                objec.dics.InsertOnSubmit(objs);
                objec.SubmitChanges();
            }
        }
    }
    foreach (string s in wikiresults)
        Console.Write(s + "\n");
}
}
```