Component Retrieval Using Genetic Algorithm

Through this be madness,
Yet there is method in it.
William Shakespeare

Component based software paradigm is receiving an increasingly attention in the software research community. The idea of reusable components to build systems is very appealing as it can reduce the development risks and costs while increasing the functionality and capability of systems. The most critical processes in component based software systems are the selection of the components from repository. There are many factors which influence the selection process; these factors should be considered from the beginning of the selection process and to avoid undesirable restrictions for future process. The retrieval of reusable component from software library known as a repository has been the subject of active research. Despite a large number of solutions have been proposed, it is difficult to consider that this issue has been resolved satisfactorily.

Maintaining consistency between the desired component for the component based software system and the components available in the component repository is a persistent concern in component-based software engineering. Now a day in the software development practice, the percentage of time and efforts allocated to this task is still too small to avoid bugs. Various methodologies exist to annotate component selection models with data related to verification and validation, and to translate the
annotated models into component performance. In any component selection method, it is unrealistic to expect a perfect match between components needed and components available. A group of components that compose a system may have overlaps and gaps in required functionality. A gap represents a lack of functionality; an overlap can cause a confusion of responsibility and degrade nonfunctional properties like size and performance.

5.1 Scope and Goal

Developing component based software involves selecting the appropriate component, building extensions to satisfy specific requirements, and then gluing the component and other units together. The success of component based software development depends heavily on the ability to select the appropriate components. An inappropriate component selection strategy can lead to adverse effects. It could result in a shortlist of component that can hardly fulfill the required functionality, and it might also introduce overheads in system integration and maintenance phases of a project. An effective and efficient component selection process is essential to the delivery of the full potential of component based software development. If the effort required in selecting the appropriate component is too high, then it may offset the time saving in using the component development approach.

The aim of this study is to provide guidance on the evaluation, selection, and qualification of components for integration into systems in acquisition. The selection and evaluation of candidate components for integration into a system involves analysis...
of both user’s requirements and specification provided by repository. Components are in a form of software reuse philosophy that provides a means to ‘plug and play’ paradigm into a system. The use of component in software systems is becoming increasingly common due to less cost, rapid evolution of component, and the growth rate of system requirements to be supported. In the series of these technical unsolved issues, this study would be an attempt to throw light on the one of the major issue of component based software engineering that is concerned with the “Component Selection”. Recently a Genetic Algorithms based approach is used for component selection to minimize the gap between components needed and components available. A relevant objective is to make these methodologies acceptable from the software engineering community. Therefore, in this study we develop Genetic Algorithms based approach for selection component.

5.2 Principles of Component Retrieval

Component based development is gaining popularity in both industry and academia as an effective reuse paradigm. Components in component based development are mainly for inter-organizational reuse, rather than intra-organizational reuse. One of the common form of reusing components is to acquire and customize them for each application. Therefore, components must be developed with consideration of commonality and variability in a domain in order to increase the reusability and applicability. One effective factor in determining the quality of components is how
precisely the variability is modeled and how effective the customization mechanisms are.

Rising costs and schedule constraints have forced software developer to use pre-built components for developing applications. By using pre-existing components to fulfill user’s requirements, software development organizations seek to reduce development costs while still producing quality software in a timely manner. The selection of COTS products is a major challenge to COTS-Based Software developers, due to the multiplicity of similar COTS products on the market with varying capabilities and quality differences [Wanyama2008].

Component selection decisions are often made in an ad-hoc manner. Component selection processes have been proposed to improve upon the efficiency and effectiveness of informal methods. Existing selection processes do not fully address the specification and evaluation of functional and non-functional requirements. Existing component selection processes specify methods to elicit software requirements in the general sense, but they do not explicitly address how to specify quality requirements. Paramount to the success of component based process, are the assessment and selection of the “right set” of components. There are seven major criteria for the assessment and selection of component:

- Assessment of functional and non-functional requirements.
- Evaluation of candidate components specification.
- Assessment of interoperability to ensure that the selected components will satisfactorily interact with each other.

- Portability or Platform Comparability.

- Third Party Certification

- Legal issue to use.

- Cost

The development of software application systems will get benefited from high reusability because similar functionality circumstances reoccur frequently in different applications. However, research in software reuse has shown that complexity of component selection, mismatches of components with the application architecture and other components interoperability destroy the component reusability.

Research in software reuse has shown that “as-is” reuse occurs extremely little and that components generally need to be changed in some way to match the application architecture, state and other components. In the component based software development process, optimum evaluation and selection of component from the repository is one of the key aspects of the life cycle. Its success mainly depends on the user’s requirements, accurate understanding of the functionality and limitations of the candidate component and system architecture.

The selection of suitable components is often a non-trivial task and all these aspects should be carefully considered. Component selection process starts with
decompositions of the requirements into a hierarchical set of criteria for the candidate's component. These criteria usually include cost, legal issue to use, specification, quality attributes i.e., functional, non-functional requirements and architecture constraints. Then, during the selection process the properties of each candidate component are evaluated according to this set of criteria.

Mis-match in component required and component retrieved from component repository is usually costly and possibly life threatening. Faulty component retrieval policy is costly in term of money, public safety and overall quality of system. As component based systems become more complex and increasing used in commercial as well as academics area, the cost of system failure continue to escalate. Component retrieval is an important area of research aimed to produce more reliable systems. In the last decade, a lot of research work was performed in the field of component based software engineering for quality assurance and testing using Genetic Algorithms. Researchers showed that component based software methodologies can greatly be improved by enhancing flexibility and maintainability of systems. This approach can potentially be used to reduce software development costs to assemble systems rapidly, and to reduce the spiraling maintenance burden associated with the support and upgrade of large software systems. While this is certainly true, Integrator is still required to choose right component from component repository. This can lead to multiple, possible incompatible versions of component selection. Recently, researchers began analyzing

*Putting the Pieces Together In Reference of Component Based Software*

*Engineering* 129
inspection component repository to obtain insights on how component based software can be improved.

Software integrator needs to identify potentially required functionality component so that optimum quality system can be generated. Selecting the appropriate component to meet specified functional and Non-functional requirements is a requirement assessment problem. It is widely agreed that requirements must be defined and quantifiable in order to selection of component. For the assessment of component functional and non-functional requirements must be specified and delegated to particular components in the software design. Existing component selection processes specify methods to elicit software requirements in the general sense, but they do not explicitly address how to specify requirements. Once requirements are specified, available components must be evaluated to determine their suitability for use in the system being developed.

5.3 Criteria for Component Retrieval

There are number of quality attributes which directly or indirectly influence the component selection process. These are Performance, Specification, Functionality, Security, Safety, Certification, Maintainability, Update Cycle, Maturity, Upward compatibility of revisions, Quality, Reliability, Architectural compatibility, Portability, Efficiency, Interoperability. The large number of attributes make it very difficult to select appropriate component.
5.4 Retrieval Query of Component

Component retrieval process analogues as a matching of a candidate component against a user's query, the condition or limitation of both the query and the repository is an important consideration.

Select Component from repository name

When condition (Quality Attributes)

It is also analogue to negotiate the process. People have used negotiation as a means of compromise, in order to reach mutual agreements, negotiation is a critical activity in business transactions. In general "negotiation" is defined as an iterative process which aims to achieve a mutually beneficial deal for the component developer and integrator.

Trading is the natural mechanism defined in third party component for searching and locating components. An integrator generate a query for particular required component service by matching components from reusable repository. Component repositories are usually called "exports", while queries are called "imports".

In CBSD, the process of building components based systems includes some tasks, such as [Grundy2000]

(a) Searching for components that satisfy the requirements and operable in the desired system architecture.

(b) Evaluating these components.
(c) Adapting or extending the selected components to fit into the desired architecture.

(d) Integrating these components together.

It is very important for these processes to use complete, concise and unambiguous specifications of components in order to guarantee a successful component software development. In addition, these specifications could be later registered into well-know repositories by developers or third parties, facilitating the component development process.

By selection components, we accept their architectural restrictions. When the architecture does not accommodate a particular component, we have a choice: either to change the architecture or to use a different component. The presence of this choice is fundamental to the selection of multiple components. However, if a candidate component provides a limited applicability and customizability so that it does not completely satisfy the functionality and interface needed, then a component integrator cannot reuse the component in application development. It's known as a partial matching problem.

5.5 Component Selection Strategies

There are three basic strategies for selecting a component, depending on whether an application development needs the best available component:

(i) Best-fit strategy: the selection process is aimed at identifying the best component among the candidate component from repository.
Algorithms for Best-fit Component Retrieval Strategy

Identify Requirement
  While (repository #empty) // repository not empty
      Goto 8
  do
    if (Search (req== Si))
      Collect[j]=Si
      j++
      Call Best( Collect[], Req, Sr_i)
      Select (C_i)

(ii) First-fit strategy: the selection process is aimed at identifying the first component
that satisfies all of the requirements from the available repository is selected.

Algorithms for First-fit Component Retrieval Strategy

Identify requirement
  // Repository not empty
  While (repository # empty) goto 8
  for(i=0;i<N;i++)
    // search repository for required candidate
    component with given specification
    {search(req==Si)
      Select(C_i)Break;
    }
  Construct component

Figure 5.1 Component Selection Strategies
(iii) **Worst-fit strategy:** the selection process is aimed at identifying the first component that satisfies all of the requirements. If no component satisfies all of the requirements, then the best component from the available repository is selected. This is termed as worst-fit strategy.

In the component based development process Best-fit Strategy is mainly considered, due to large cost associated with systems failure.

<table>
<thead>
<tr>
<th>First Fit</th>
<th>Best-Fit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. Identify requirement</td>
<td>0. Identify Requirement</td>
</tr>
<tr>
<td>1. // Repository not empty</td>
<td>1. // repository not empty</td>
</tr>
<tr>
<td>2. While (repository≠ empty) goto 8</td>
<td>2. While (repository≠empty) Goto 8</td>
</tr>
<tr>
<td>3. for(i=0;i&lt;N;i++)</td>
<td>3.do</td>
</tr>
<tr>
<td>4. // search repository for required candidate component with given specification</td>
<td>4. if(Search (req== S_i))</td>
</tr>
<tr>
<td>5. { search(req==S_i)</td>
<td>Collect[j]=S_i</td>
</tr>
<tr>
<td>6. Select(C_i)Break;</td>
<td>5 j++</td>
</tr>
<tr>
<td>7. }</td>
<td>Call Best( Collect[] ,Req, S_{r_i})</td>
</tr>
<tr>
<td>8. Construct component</td>
<td>Select (C_i)</td>
</tr>
</tbody>
</table>

### 5.6 Prerequisite for Component Retrieval

There are two goals of component retrieval operation:
(i) Exact retrieval, whose purpose is to identify reusable component assets that are correct with respect to a desired specification, exact retrieval fits in the life-cycle of black box reuse, whereby correct assets are identified and retrieved. Here assets are evaluated with respect to the query for selection of component which follows the best fit strategy.

(ii) Approximate retrieval, whose purpose is to identify reusable component assets that can be modified with minimal effort to satisfy a required specification, Approximate retrieval fits in the life-cycle of white box reuse, whereby reusable assets that approximate the query are identified and retrieved. Here retrieved assets are evaluated with respect to the query to select to minimize the modification effort.

Researchers have noted that a technique for retrieving and selecting a component which satisfies a requirement specification has not yet been established [Mili1997]. A component repository and a retrieval mechanism which appropriately supports the retrieval of components from the repository are necessary to implement the software reuse.

5.7 Existing Component Retrieval Strategies

Several methodological proposals have been formulated to improve effectiveness of the selection of components. Component selection methods are traditionally done in an architecture-centric manner. They aim to answer the question: a component needed for this system, what is the best existing alternative available in the market?
Existing methods include OTSO, PORE, and CRE. These methods define criteria upon which to judge alternatives for a component role and the synthesis of multiple criteria to decide the most promising alternative. [Sora2006] proposes fuzzy logic based solution for the specification and retrieval of software components.

5.7.1 Off-The-Shelf-Option (OTSO) Selection Method

Kontio proposed the Off-The-Shelf-Option (OTSO) selection method. The OTSO method assumes that the requirements of the proposed system already exist. However, in practice, the requirements cannot be defined precisely because the use of certain COTS products may require some changes to the requirements. The OTSO selection method comprises three phases: searching, screening and evaluation.

The searching phase attempts to identify all potential COTS candidates that cover most of the required functionality. The objective of the screening phase is to decide which COTS candidates should be selected for detailed evaluation. In the evaluation phase, COTS candidates undergo a detailed evaluation [Kontio1995,1996].

5.7.2 Procurement-Oriented Requirements Engineering (PORE)

The PORE method guides a software development team in acquiring customer requirements and selecting COTS products that satisfy those requirements [Maiden2000]. It uses a progressive filtering strategy, whereby COTS products are initially selected from a set of potential candidates, and then progressively eliminated when they do not satisfy the evaluation criteria.
5.7.2 COTS-based Requirements Engineering (CRE) Method

The CRE method was developed to facilitate a systematic, repeatable and requirements-driven COTS product selection process. A key issue that is supported by this method is that of the definition and analysis of the non-functional requirements during the COTS product evaluation and selection [Alves2001]. This method has four iterative phases: identification, description, evaluation and acceptance.

Another type of component selection approach is built around the relationship between requirements and components available for use. The goal here is to recognize the mutual influence between requirements and components in order to obtain a set of requirements that is consistent. This is done by merging requirements engineering and component selection techniques. In any component selection method, it is unrealistic to expect a perfect match between components needed and components available. An architectural decision is analyzed by observing the effect it has on certain properties (quality attributes) of an architecture. Conventional retrieval approach for the software component which is in the wide sense can be classified into four groups: (i) automatic extraction approach, (ii) specification-based approach, (iii) similarity distance-based approach and (iv) type-based approach.

(i) The automatic extraction approach is based on the automatic extraction of structural information from components. The user’s queries are expressed as keywords corresponding to the names of interfaces, components, and so forth. This approach is effective in the case where source codes of the components are available. However, in
the case that the source codes are not available, the extracted information is insufficient for the retrieval.

(ii) The semi-formal specification-based approach is based on catalog information of the components. The user's queries are given as keywords which correspond to a specification of the catalog. In addition, the formal specification-based approach uses semantic description of the component's behavior.

(iii) The similarity distance-based approach is based on the similarity between a user's query and the component stored in the repository. There are two major approaches in the similarity evaluation method: an approach using the class inheritance relation in OO language and an approach using the similarity of element names. The user's queries are given by a prototype of the component which satisfies the user's requirement [Frakes 1994].

(iv) The type-based approach is based on component type and component interface type. The user's queries are given by type information expected to realize the user's requirement. Search results are classified according to adaptability, for example, exact match and generalized match, but more detailed ranking with each match set cannot be obtained.

These conventional approaches consider only a single characteristic of the component when retrieving. Therefore, these approaches cannot evaluate the total semantic adaptability of the component. The retrieval mechanism should be able to consider two or more characteristics simultaneously when retrieving. In addition, not all components
circulated over the Internet have additional specification descriptions. Therefore, the
retrieval mechanism should not require any additional information other than the
components themselves. The current component selection methods can be classified
into three categories, namely, the intuition approach, the direct assessment approach
and the indirect assessment approach [Leung2003].

In the intuition approach, software developers select component according to their
experience and intuition. This approach is subjective, and some components that are
qualified candidates for an application may be omitted inadvertently. In the direct
assessment (DA) approach component are selected directly from their source. These
methods consider ALL of the descriptions of the component and then try to make
decisions on their suitability.

5.8 Component Retrieval Process

One of the essential tasks in component-based development (CBD) is to locate and
reuse the right components that provide the desired functionality required by user,
called component acquisition. Once appropriate components are selected, they are
customized for the target application through available customization mechanisms,
called component adaptation [Vallecillo2000]

First of all requirements is gathered from the end user side which is further
analyzed and matched with the component specification. In case component
requirement matches with the specification provided by reusable repository then it is
selected otherwise requirement is modified accordingly. This process will be repeated till solution is achieved. This sequence has been shown in the diagram and is self explanatory.

![Component Selection Process Diagram](image)

**Figure 5.2 Component Selection Process**

**Algorithm** Component Selection

1. **procedure** Selection Component

   \[ i := 0 \]
Dynamic Array C[n]

check

if (repository ≠ null)

loop

for (i=0; i ≤ m ; i++)

    Function _Choose ( Requirement , repository[size],
precision)

    If Match(Requirement==Specification_component[i])

    Choice[]=component[i]

Else MSG("No Component with require functionality call
Modify Requirement")

else error_msg(O)

    Modify_requirement( User , Intergrator,
Specification_component)

In the context of component selection, it has been realized that diversity in
component selection models [PORE, CRE, OTSO, etc] imposes some restrictions that
bounds one to select component of specific features within a particular model. Such
processes are time consuming. So to overcome these problems and to extend

Putting the Pieces Together In Reference of Component Based Software

Engineering 141
component selection procedure, we are presenting XML and GA based component retrieval approaches.

5.8.1 **Component Retrieval Implementation through XML Modeling**

The component selection technique is to define the selection of optimum component and matching process for client’s requirement to available repository. Select one of the variants realized inside the components. Once a variant is selected, the value is stored and remembered so that further invocations can refer to the selected variant. The selection mechanism works in six steps.

- **Step 1. Requirement gathering**
- **Step 2. Defining requirement specification**
- **Step 3. Matching process**
- **Step 4. Select Component**
  
  Otherwise goto step 6

- **Step 5. Composition break**

- **Step 6 Modify Requirement.**

XML is a standard design modeling language [DAUM2003].

**XML Modeling Component Repository**

XML has been introduced by the World Wide Web Consortium (W3C) as a way to describe structured data. It is a text-format based, platform-independent markup...
language that is derived from SGML (Standard Generalized Markup Language). Since it is text-format, it is human readable and editable. It is similar to HTML but unlike HTML, one can define his/her own set of tags. The big advantage of XML documents is the separation of syntax and semantics. The content of an XML document is independent from its rendering. And because of this, XML is now playing an increasingly important role in data exchanging over the web [Zhuk2004].

An XML document is mainly composed of elements which are enclosed by start tags and end tags. Elements are nested and they can have attributes that are assigned values in the start tags. The whole document is contained by a distinguished root element which appears at the very beginning of the document

```xml
<? xml version = "1.0"?>
<? xml version = "1.0" encoding= " ISO -10646-ucs-2"?>
<! Component Repository Modeling-->
<title xml:lang= "en-us"> Query based Component Repository Modeling </title>
<repository> Component Bank 
</repository>
<components>
<component cid= "c1">
<specification>
```

*Puting the Pieces Together In Reference of Component Based Software*  
*Engineering*  143
Component selection decisions are often made in an ad-hoc manner. Component selection processes have been proposed to improve upon the efficiency and effectiveness of informal methods. Existing selection processes do not fully address the specification and evaluation of functional and non-functional requirements. Effective selection of reusable components in component-based software systems is a hot issue for the research community. A number of integration risks can often be resolved by selecting the 'right' set of components. One of the most critical processes in Component Based
Software Engineering is the selection of the components from a reusable repository that meet the user requirements. However, we have found that the information required for evaluating those components used for in quality models and metrics is not usually available in the existing commercial software repositories. In any component selection method, it is unrealistic to expect a perfect match between components needed and components available.

Software component retrieval is an area of active research in the field of component-based software engineering (CBSE). A number of components are required to have a reasonable chance of finding matching components for a given task. This study proposes a Genetic Algorithms based approach to satisfy such a need to its ability to naturally represent data selection. To support the development of high quality component-based systems, component selection processes need to address the problem of Functional and Non-Functional requirement evaluation of component-based software systems. This research proposes a component selection process, which is based on the Genetic Algorithms. Identification of candidate component is a complex activity itself.

Selection of Right Component –

How can we meet the software integrator’s needs for finding, use and reuse components?

We argue that the matching process between component available and user’s requirement is a fundamental issue to be treated as a highest priority for quality
assurance. In this context we propose a Genetic Algorithms approach to identify suitable component from component repository. A research oriented Genetic Algorithms should have two basic properties.

- It should be easily customizable and extensible.
- It should support software engineering principle.

Component Retrieval Procedure Phases:

Step 1. Requirement Analysis

Step 2. Check Specification of Component

Step 3. Compare Requirement with Component Specification

Step 4 if match occurs select component otherwise

elsif modified requirement

5.8.2 Genetic Algorithms Procedure for Component Retrieval

A Genetic Algorithms is a search procedure modeled on the mechanics of natural selection rather than a simulated reasoning process. The approach is inspired by Darwin’s theory of evolution, which is based on the survival of the fittest. A solution to a problem is considered an individual in a population of solutions. Domain knowledge regarding the problem is embedded in the abstract representation of a candidate solution termed as organism. Populations are sets of these organisms successive populations are called generations. Generations are produced by first identifying the
mating partners, and then applying the genetic operators on them to produce their offspring. Each individual so produced is checked for feasibility using a fitness function.

The methodologies have significant impact on the retrieval performance of a component based software systems. A variable selection that satisfies the coordination constraint, is the feasible solution to the design problem. These methods for generating feasible component selection employ expensive computation and perform blind search to identify candidate component in the reuse component repository. A classification schemes based on a Genetic Algorithms were proposed to promote the qualities. Effective reuse of component requires well-defined repository. Without these, the component repository becomes a write only storage medium. The repository of component is the link between Integrator, developer and user of the systems.

The genetic approach views a variable organization as an organism with several layers of chromosomes [Sivanandam2008]. The populations of these component repositories are produced using genetic operators and each individual in these generations is checked for feasibility using a fitness function, which evaluates each individual component against user-defined criteria. Information obtained during this evaluation is used in genetic operators to direct a parallel heuristic search of the solution space for more fit component in the next (following) generations.

The initial search space (population) of a Genetic Algorithms usually consists of solutions (individuals) generated randomly. With the initial population as the starting
point, a Genetic Algorithms mounts a parallel heuristic search of the entire solution space for better solutions. The structure of a component repository may be seen as a collection of independent several components capable of performing appropriate tasks. The inputs to the Component Repository (organization) are data from several sources of information, called requirement (sensors). This requirement (sensory) information determines which function is to be performed by the component. A component repository with each constitute chromosome representing user-defined constraints, shows the schema for the generation of future populations of structures. A ‘1’ or ‘0’ at any position means that the chromosomes in future populations must have the same value at that position for them to belong to the schema. The x’s represent the genes (interactions) that can be replaced by either 1s or 0s genetically to generate new populations of solutions.

The first step in the genetic algorithm approach requires an initial population of organisms to start the process. The bit strings representing the individual chromosomes are used to initialize the population. In order to feasible component repository, or a fit organism in genetic terminology, each one chromosome in an organism represent feasible fixed structure, the organism is checked for coordination constraint [Dixit2009a].

5.8.2.1 Genetic Algorithms

1. Initiate a population of chromosomes.

2. Evaluate each chromosome in the population.
3. Create new chromosome by malting current chromosome
4. Delete members of the population to make room for the new chromosome.
5. Evaluate the new chromosome and insert them into the population.
6. If time is up or get require, stop and return the best chromosome.

In the following section we present a novel approach for selecting candidate component from component repository.

5.8.2.2 Algorithm for component Selection

Input: Component Repository $C_R$; User Requirement $U_R$

Output: Component Select $C_i$

Begin

$\text{find} \leftarrow \text{true}$;

While $C_R \neq \emptyset$ and find = true do

Select $U_{Ri} \in U_R$

for $i = 0$ to $n$

for $j = 0$ to $m$

if $(C_{Rj} = U_{Ri})$

---

*Putting the Pieces Together In Reference of Component Based Software Engineering* 149
select $C_{Rj}$

find $\leftarrow$ false;

break

else

Call Modify_Requirement($U_R$, $C_R$, Result)

return $(U_R')$

endif

endif

call Create_Component($U_R$)

end while

5.8.2.3 Genetic Algorithm Implementation for Component Selection

begin

t=0; initial

time at the start of the algorithms

Initialize population $C_R(t)$ component Repository

Evaluate population $C_R(t)$ compute fitness of all

initial component in the population

While $C_R\neq {}$ & find $=$true do
5.8.2.4 Binary -Coded GA

Consider a Component Selection problem using binary-coded GA

In the binary-coded GA, the solutions are represented in the form of binary strings composed of 1s and 0s. A binary string can be compared to a biological chromosome.

Step1- Generation of a population of solutions: An initial population of solution of size N, here N may be 10, 15, 20, .......

1. N depending on the complexity of the problem
2. The length of a binary string based on a desired accuracy in the result.

Let N=4 it is too small for GA Application but we only consider how to find best one
Table 5.1 Initial Population

<table>
<thead>
<tr>
<th>Chromosome label</th>
<th>Initial Population</th>
<th>Decoded Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100100</td>
<td>36</td>
</tr>
<tr>
<td>B</td>
<td>011011</td>
<td>27</td>
</tr>
<tr>
<td>C</td>
<td>101010</td>
<td>42</td>
</tr>
<tr>
<td>D</td>
<td>111010</td>
<td>58</td>
</tr>
</tbody>
</table>

Step 2: Fitness Evaluation:

The fitness value of each population is calculated. Let the fitness be equal to the no. of ones in the strings.

\[ P_{select} = \frac{f_i}{\sum f_i} \]

A common selection method in GAs is fitness-proportionate selection, in which the number of times an individual is expected to reproduce is equal to its fitness divided by the average of fitnesses in the populations. The population of solutions is then modified using different operators.
Table 5.2 GA Based Fitness Evaluation

<table>
<thead>
<tr>
<th>Initial Population</th>
<th>Value</th>
<th>Fitness ( f_i )</th>
<th>Pselect</th>
<th>Expected Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>100100</td>
<td>36</td>
<td>2</td>
<td>0.143</td>
<td>0.572</td>
</tr>
<tr>
<td>011011</td>
<td>27</td>
<td>4</td>
<td>0.286</td>
<td>1.144</td>
</tr>
<tr>
<td>101010</td>
<td>42</td>
<td>3</td>
<td>0.214</td>
<td>0.856</td>
</tr>
<tr>
<td>111011</td>
<td>58</td>
<td>5</td>
<td>0.357</td>
<td>1.428</td>
</tr>
</tbody>
</table>

Step 3: Reproduction:

All the GA-Strings may not be equally good in terms of their fitness value calculated in Step2. In this step, an operator named reproduction is used to select the good ones from the population of strings based on their fitness information. There are several reproduction schemes developed by various investigators.

1. Proportionate Selection/Roulette—Wheel Selection

In proportionate scheme the probability of a string for being selected in the mating pool is considered to be proportional to its fitness.

2. Ranking Selection
Ranking selection is also based on fitness value of population. In ranking based selection string are arranged in an ascending order of their fitness value. The lowest fitness value string gets rank one and other strings are ranked accordingly.

3. Tournament Selection

In this selection scheme first we select random tournament size, which is smaller than our population size N, than we choose n string at random and determine the best one in the fitness value.

4. Elitism

In this selection scheme first we select an elite string and directly copied into next generation to ensure its presence.

We used Ranking Selection; there are only four binary strings in initial population of GA-solution, where fitness values are 2, 4, 3, 5 respectively. The process of ranking selection consists of two steps.

Step 1: Initial Population is arranged by ascending order of their fitness value. The string having the lowest fitness is assigned rank 1 and other
### Table 5.3 Rank Based Reproduction

<table>
<thead>
<tr>
<th>Initial Population</th>
<th>Value</th>
<th>Fitness $f_i$</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>100100</td>
<td>36</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>011011</td>
<td>27</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>101010</td>
<td>42</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>111011</td>
<td>58</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Step 2: A proportionate selection scheme based on the assigned rank is adopted.

**Step 4: Crossover:**

In the crossover parents are selected at random from populations. There is an exchange of properties between two parents and as a result two children solutions are produced. There are number of techniques present for crossover.

1. **Single Point Crossover**

Crossover site lying between 1 and L-1, where L denotes the string length. The left side generally kept unchanged and swapping is done the two sub strings on right side of the crossover site.

2. **Two Point Crossover**
Two different crossover sites are selected in between 1 and L-1.

3. Multi Point Crossover

A numbers of crossover point are selected along the length of string.

4. Uniform Crossover

Uniform crossover has equal chance to every bit for interchange.

Using single point crossover

100100 | 011011
101010 | 110111

Two children produced due to the single -point crossover are:

100100 | 110111
101010 | 011011

Step 5: Mutation

The biology term "Mutation" means a sudden change of parameters. In the GA , the mutation is performed by making a local change over the current solution. In mutation, 1 is converted into 0 and vice -versa. The GA run until the termination criterion is reached that depends upon desired accuracy of solution.

This study makes us to put forward our research step with the realization that selection of component is the basic initial requirement to assure the quality of software systems. For desired selection of component, the proper specifications of components
should be provided by the developer. The different strategies for component selection have been discussed with the creation of GA which is a motivational factor to speed-up the retrieval procedure.

Case Study

Data Source: ComponentSource

Component Selection using GA

We produced five implementations of the component repository. Its first implementation consisted of twenty four components. The second implementation had fifty components, the third had hundred components, the fourth implementations consisted two hundred and last contained four hundred components. Our results show GA based retrieval approach do much better as compared to other retrieval principle.
Table 5.4 Speedup by GA Retrieval

<table>
<thead>
<tr>
<th>No of Component</th>
<th>Average Time in Retrieval (other) unit sec</th>
<th>Average Time in Retrieval by GA unit sec</th>
<th>Ratio of speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>5.34- 6.03</td>
<td>4.47</td>
<td>1.19</td>
</tr>
<tr>
<td>50</td>
<td>7.23-8.11</td>
<td>5.02</td>
<td>1.44</td>
</tr>
<tr>
<td>100</td>
<td>8.12-9</td>
<td>6.3</td>
<td>1.28</td>
</tr>
<tr>
<td>200</td>
<td>10.02-10.92</td>
<td>8.46</td>
<td>1.18</td>
</tr>
<tr>
<td>400</td>
<td>13.23-15.37</td>
<td>11.23</td>
<td>1.17</td>
</tr>
</tbody>
</table>

This table shows the speedup provided by our solution with respect to a solution that use other retrieval principle.