An Efficient Approach for Evolution of Functional Requirements to Improve the Quality of Software Architecture

M. Sunil Kumar and A. Rama Mohan Reddy

Abstract Software architecture will be designed within the early phases combined with the development process; the huge constraints makes it possible for the achievement of certain functional requirements, quality attributes (non-functional requirements), and also business goals. Metaheuristic search algorithm performs an important role within the software architecture design to improve the performance of obtaining an optimal solution from the huge search space. This particular paper mainly focusses on balancing the combinations of “Adaptive Genetic algorithm,” which has to be applied. It has incorporated the usage of roulette wheel selection operators; this technique is implemented in java and it also finds out global minima as well as time reduction when compared with Genetic algorithm.

Keywords Software architecture · Functional requirements · Quality attributes · Responsibility · Metaheuristic search algorithms · Adaptive genetic algorithm · Simulated annealing

1 Introduction

Software plays an essential role in all the aspects of our life. The rising usage of several applications facilitating varying user’s necessities creates a sharp increase in the scale and complexity of software, which results in the decline of the software quality. As a result, the main challenges in software engineering are to measure, understand, manage, control, and lower the software complexity [1]. Software product line engineering aims at improving the productivity and reduces the realization times by collecting the analysis, design, and implementation actions of a

M. Sunil Kumar (✉)
Department of CSE, Sree Vidyanikethan Engineering College, Tirupati, India
e-mail: sunilmalchi1@gmail.com

A. Rama Mohan Reddy
Department of CSE, S.V.U. College of Engineering, SV University, Tirupati, India

S.S. Dash et al. (eds.), *Artificial Intelligence and Evolutionary Computations in Engineering Systems*, Advances in Intelligent Systems and Computing 394,
DOI 10.1007/978-81-322-2656-7_71

family of systems. Characteristics that vary from product to product form the variabilities. The major challenge in the context of the software product lines (PL) approach is to form and put into practice these variabilities [2]. Along with the magnitude and intricacy of the software systems varies the computational complexity of theirs. The design of the software system, the allied methodologies, and data structures play a major role in fixing the problem. The major structural concerns in the process of software system design include designing efficient communication protocols and simpler synchronization mechanisms, component formation and assessment of the components as a whole, physical placement of components on the network nodes, service distribution on the network nodes in order to support faster services to the user, etc. The design process is a spiral model, which presents the most appropriate design model given in the system requirements by considering all the possible design alternatives.

2 Problem Statement

Along with the growth of software industry, the necessity for online software systems has increased. To accommodate the enhancing technology and needs into the system on-the-fly, the architecture of the software system should be carefully designed and thoroughly analyzed. Standards, procedures, and processes to develop such system architectures need to be established. Automation up to some extent may help software architects to start from a general skeleton architecture and build on it. This leaves the architects more time to refine the architectural designs to the maximum possible extent.

This paper is only a humble beginning and there is yet an ocean of research to be done in the field. The analysis on the associated workings, which uses the genetic algorithm, has revealed that regardless of being technically sophisticated, the methods seem to be short of reduction in the computational time which further influences the excellence of the software. While designing the software, computational time is given the principal consideration. The software is considered to be quality software only when the time for computing a process is less. This has necessitated the requirement for developing better techniques for the architecture of the software with the intention of reducing the computational time and also to boost the quality of the software being designed.

Automation process may further be used to compare various possible architectural styles and designs for a given application in order to select the appropriate and best suitable combination. The idea of using the concepts of genetic science to refine the architecture model to produce the best possible automation is studied in this thesis work. The basic idea is to maintain a database of architectural styles, design patterns, and apply genetic algorithms to derive all the possible design alternatives for a given application/system, by providing the basic constraints/guidelines to be followed while generating the possibilities.

To facilitate in overcoming the disadvantages, a novel approach “An Adaptive Genetic Algorithm” has been taken in the beginning with a set of responsibilities in the class diagram. The responsibility dependency graph has been then derived which is further considered as input to the adaptive genetic algorithm, which produces suggestions for identifying the quality requirements for the architecture.

3 Metaheuristic Search Algorithm

3.1 Justification of Applicability of Metaheuristic Algorithms in Software Architecture Domain

The field of metaheuristic provides solutions to various combinatorial optimization problems like scheduling problem, integer programming, resource assignment problem, cutting stock problem, network routing problem, tree optimization problem, neural network design problem, etc. [3]. The field of Evolutionary Algorithms (EAs) had its beginning when researchers thought of using the phenomena observed in the evolution of life-form in the real world. Since then, a variety of algorithms have been developed. For example, optimization algorithms based on ant-colony behaviors have been designed. The inspiration was taken from the biological processes like population, reproduction, mutation, selection, survival of the fittest, etc. One of the advantages of evolutionary algorithms is that they can be easily modularized and hence parallel programming can be used.

For many applications or real-world problems, there does not exist exact problem-specific solutions and heuristics to be followed to find optimal and feasible solutions. In such cases, the field of metaheuristic can be helpful beyond the expectations in finding an optimal and computable solution for a given problem. The field of evolutionary algorithms along with defined problem-specific heuristics in combination with the other fields like local-search-based techniques, sampling techniques, etc., is aimed at achieving highly efficient optimization algorithms. Any problem when transformed into a combinatorial optimization problem can be solved by using metaheuristics.

3.2 Genetic Algorithms

On observing the evolution of different species of life-form, many biological principles have been found, which can be used for supporting natural evolution of a feasible solution in the process of search-based optimization [4]. It shares a commonality with the simulated annealing approach that it performs better when compared to gradient-based optimization strategy [5]. Its use is particularly appreciated when it is applied to applications or systems that show high nonlinearity. Also, it is

Table 1 Application of genetic algorithm

Some application areas by domain	Some application areas by technique
Industrial design by parameterization	Chromosomes to set membership and function optimization
Scheduling	Real valued chromosomes to function optimization
Network design by construction	Order-based binary chromosomes for optimization by construction
Routing	Tree-based chromosomes for genetic
Time series prediction	Domain-specific chromosomes for specialized solutions to particular problems

useful in situations where global optimum is to be derived from the many existing local optimums. The general working strategy of the GA algorithms can be described as follows in Table 1.

i. A random selection of a set of initial seed points is made from the given design search space, then
ii. Design alternatives are further improved by repeatedly applying genetic principles.

3.3 *Applicability of Genetic Algorithm in Architectural Modeling*

The dramatic improvement in the field of computing technology has given immense scope to researchers world-wide to perform deep and enhanced experimentation with their approaches efficiently and swiftly. The applicability of the newly developed approaches in specific domains could be proved. Further improvements could be effectively suggested. This has speeded up the area of research productively. As genetic algorithms need high computational power, their use after their conception was limited. As the technology grew, their use was steadily boosted up. But in the software architecture design process, the usage of genetic algorithms has been thought of and worked upon only recently. Only since the establishment of the importance of software architecture design (in handling large and complex problems efficiently) has the prospective idea of utilizing genetic principles in the design process been given a thought. Developing various design alternatives and evaluating them to extract a best design solution throws a challenge to the architects, where search-based optimization comes into picture [6]. Genetic algorithms are best suited for handling these challenges in the process of designing software architecture.

In software architecture design, genetic algorithms can be used to handle two challenges. In one way, genetic algorithms help to choose the optimal feasible solution among the given population of the design alternatives by setting objective functions. In another way, genetic algorithms are helpful in the evolution of new design alternatives from the existing ones.

4 Proposed Methodology of Adaptive Genetic Algorithm

This paper uses adaptive genetic algorithm of software architectures to introduce varying levels of automation into design, planning, and maintenance phases. The conducted studies use UML-based representation of software architecture design. The work is in light of the perspective that software architecture is a result of a series of decisions made to incorporate the concerns of different stakeholders in the system. The decisions translate into solutions inside software architectures, where some solutions are specific to the system while others can be reused. In this work, all architectural changes resulting from architectural decisions are called architectural solutions or simply *solutions*. Software architecture can be considered as a collection of architectural solutions. A solution in its entirety enters or leaves software architecture thereby affecting some property of the system.

If we consider software architecture as a combination of solutions, then, designing a system can be understood as finding a feasible configuration of the solutions. Each solution not only resolves a functional requirement but also has an impact on quality attribute(s). The quality-related implications of the well understood solutions are usually well-documented and are known in advance. Thus, a careful selection of the solutions can be expected to lead to an acceptable architecture. This scenario can be formulated as a search problem: in principle, an algorithm can be designed which will find an optimal configuration of the solutions available in a solution database, given the architecturally significant requirements of the system. In this kind of problem, metaheuristic search methods usually outperform deterministic approaches. Adaptive genetic algorithms belong to the metaheuristic search methods family and have been employed in many studies to address software engineering problems.

In this paper, adaptive genetic algorithms have been employed to design software architectures. The adaptive genetic algorithm synthesizes software architectures and applies solutions from a solution base, thus producing improved designs. Each of the solutions is introduced through adaptive mutations employed in the adaptive genetic algorithm. The adaptive genetic algorithm is provided with an objective or fitness function to gauge the modifiability, efficiency, and complexity of the architectures. The algorithm takes an input initial design and properties of the developing organization to produce initial work distribution plans. The solutions are introduced in the architecture to ease its distribution among the teams as well as to reduce the inter-team communications during the architecture development. The difficulties in communication among the involved teams are therefore taken into account in the fitness function. The difficulties usually have their origin in the cultural, lingual, or social dissimilarities among the teams.

Consequently, the adaptive genetic algorithm favors low coupling among the components to be assigned to teams with significant overhead in communication and vice versa. Furthermore, extreme over- or under- loading of the teams are also discouraged by the fitness function. Our adaptive genetic algorithm-based maintenance is planned around some specific modification needs; instead, one or more properties, (e.g., efficiency, reliability, etc.) are targeted for maintenance. The major preplanning activities involve designing of the fitness function and selection of the solutions that have an influence on the maintained properties. Once equipped with the solutions and the fitness function, our adaptive genetic algorithm can address a wide range of future, possibly unpredictable modification needs associated with the maintained properties.

The developed genetic algorithm includes a set of solutions with an effect on the performance and reliability in distributed systems settings. The fitness function measures efficiency and reliability of architectures. The infrastructure allows run-time updating of Java classes. All the modifications, manual or automatic, are performed within the UML class diagram-based architectural representation of a running system. The software is considered to be quality software only when the time for computing a process is less. This necessitated for the requirement of developing a better technique for the architecture of the software with the intention of reducing the computational time and also to boost the quality of the software being designed. Besides the computational time, the fitness value for the solution also stays to be less which can influence the architecture of the software by and large. To overcome these disadvantages, we have designed a technique using the adaptive genetic algorithm which is proved to be a competent method for software architecture.

The alteration operation passed out is adaptive alteration in adaptive genetic algorithm which demonstrates to be further competent than the standard mutation operation in GA. Regarding its enhanced steps such as crossover, mutation, and also offers improved fitness value, the adaptive genetic algorithm is leading over normal GA. By pertaining to the adaptive alteration, the fitness value of the consequent solution for the architecture is enlarged when balanced with the algorithm utilized in the presented method, which can result in enhanced presentation of software. Consequently, the computational time has been enhanced to a larger extent when compared with the architecture produced by means of the standard GA process by utilizing the adaptive genetic algorithm for software architecture.

The software architecture is generally a set of decisions related to the organization of the software system and selection of the structural elements and their interfaces by which the system is composed together. Software architecture, along with the structure and behavior, is also concerned with functionality, performance, reuse, economic and technological constraints, etc. In software, its components are related to one another in a variety of ways. In order to develop good quality software, this relationship has to be represented in the architecture. The software architecture is an extremely challenging process. The functional and the non-functional necessities mostly enable the software architecture. In our proposed methodology, the responsibilities and dependency values are given as an input for finding the optimal software design.

4.1 Architecture Representation Using Adaptive Genetic Algorithm

In order to maintain variety in the course of optimization process, the chromosomes are separated into chromosomes in the recommended software architecture by means of adaptive GA. The chromosomes are similar and the folks are replaced among chromosomes, till the most excellent individuals are got through the set of chromosomes in this procedure. The substitute operation is executed by means of migration operator. Relocation of individuals among dissimilar chromosomes chased by the application of genetic operators results in the production of new individuals. To manage the level of variety, the rate of migration permits the algorithm and is continued inside the chromosomes.

To be able to elucidate that the proposed *Adaptive Genetic Algorithm* is a novel approach that has been taken in the beginning with a set of responsibilities in every class diagram shown in Fig. 1. A responsibility dependency graph is then developed for the use as input to the proposed adaptive genetic algorithm. In any field, the adaptive genetic algorithms are used to evolve best solution by performing genetic operations on the existing set of solutions. For applying adaptive genetic algorithm on a specific problem, a mapping of elements of the given problem to the elements of the adaptive genetic algorithm needs to be done—the smallest basic element of the existing solution that can be assumed as a chromosome in terms of genetic algorithm, an initial set of solutions that can be considered as the initial population, the genetic operators that should be used in the genetic algorithm, an appropriate objective or fitness function to evaluate the solutions and a selection operator to choose the survival of the fittest to be forwarded to the next generation.

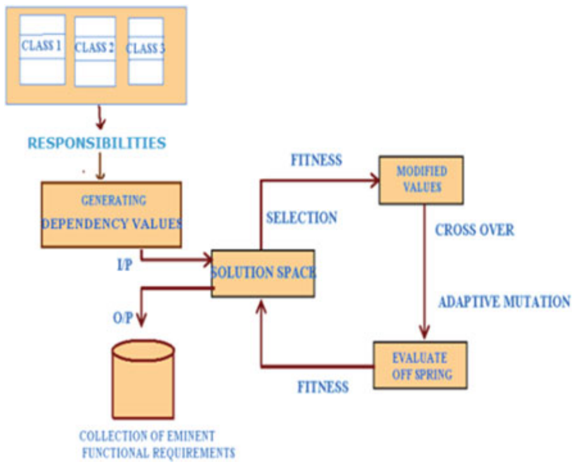

Fig. 1 Conceptual model for AGA

In order to maintain variety through optimization process, the chromosomes are separated into chromosomes within the recommended software architecture by using adaptive genetic algorithm. The chromosomes have been similar as well as the individuals are replaced among chromosomes. Throughout the selection procedure, the indiscriminately generated chromosomes and also the new chromosomes are positioned in a selection pool on the basis of their fitness values. The chromosomes which contain good fitness occupy the top positions in the pool within the selection pool.

The first chromosomes which are at the top of the selection pool are selected for the next generation within the chromosomes. Here, the selection is based on the fitness and execution time for every task. This technique is repeated through the crossover with all the selected chromosomes until it reaches the termination criteria. Throughout the process, the number of iteration reaches the absolute maximum generation and then the entire process is terminated. The chromosome which can be at the top of the selection pool is selected as the best chromosome. Depending upon the identification of quality functional requirements, quality architectural design is produced.

4.2 Adaptive Genetic Algorithm

```
Input: Dependency values, Path coverage, Statement coverage.
Output: best chromosome after termination of condition.

S1:   Initialize population with 'n' chromosomes
S2:   Identify the fitness f(x), each chromosome x in population.
S3:   For generating random population we are taking Roulette wheel selection
      method. [Selection] Select two parent chromosomes from a population according
      to their fitness
      (b)   Cross over operator is used to identify new offspring (children).

      (c)   Adaptive mutation, mutate new offspring ( new children) at each locus

      (d)   [Selection] set new offspring in new population. The new offspring will
            Give the quality responsibilities. Base on the responsibilities we are able to
            construct best software architecture.
S4:   [Replace] Use new generated population
S5:   End condition.
S6:   Go to s2
```

4.3 Generating Input for Adaptive Genetic Algorithm

In this paper, the responsibility dependency values are the inputs to the solution space. Assigning responsibilities to class is done using MH optimization algorithms. Initially, a design with responsibilities assigned to a class on differences serves as basis for applying more advanced OO mechanisms like inheritance, interfaces, abstract classes, etc.

4.3.1 Responsibility Value

These values are gathered from the class diagrams; assigning responsibilities to classes is among first and arguably most important step when creating object-oriented software design. This step depends greatly on human judgment and experience to automate the process of assigning responsibilities by using classes. Here is the example of retrieving the responsibilities from the class diagram.

4.3.2 Responsibility Dependency Graph

The concept of responsibility dependency graph is developed in order to efficiently represent the set of functional requirements of the software systems. The graph represents the functional requirements in the form of simple responsibilities. A responsibility can be defined as a job to be handled by either the complete software system or by a module or component of the software system. It can also be termed as a data object or item that has to be maintained in integrity by either the complete software system or by a module or component of the software system.

The responsibility dependency graph, as the name itself indicates, represents responsibilities and the dependency relationships between the responsibilities. Responsibilities are denoted using an oval symbol which is technically termed as a node and the dependency relationships between responsibilities is represented by using directed edges. A directed edge specifies that the fulfillment of the source responsibility is dependent on the fulfillment of the target responsibility. Using this representation format, a responsibility dependency graph for any given software system can be developed. Once this graph is developed, properties such as path coverage, dependency value, and statement coverage of the responsibilities are considered for the purpose of evaluating the software system.

The hospital management software system considered here can be divided into five major modules—new user registration and user login module, patient module, doctor module, bill entry and maintenance module, and report generation module. On the analysis of these modules, a set of functional requirements have been identified which have been modeled as 26 responsibilities having 52 dependencies relating them. A typical selection approach *assigns* the probability of selection P_j to each and every individual j according to its fitness value. A number of N random numbers is actually generated as well as compared toward the cumulative probability $C_i = \sum_{j=1}^{i} P_i$ from the population. A proper individual i will be selected along with the one copied to in the new population if $C_{i-1} < U(0, 1) \leq C_i$. A variety of methods exist in order to assign probabilities to individuals: roulette wheel, linear ranking as well as geometric ranking.

Roulette wheel, may be the first selection method. The particular probability P_i for each and every individual is actually described by:

$$P[\text{Individual } i \text{ is chosen}] = F_i / \sum_{j=1}^{\text{pop size}} F_j$$

In this, F_i equals the particular fitness of individual i. The usage of roulette wheel selection limits the actual genetic algorithm in order to maximize, since the evaluation function needs to map the particular solutions in order to a completely ordered number of values on $\mathfrak{R}^+$. Extensions, such as windowing as well as scaling are actually proposed to permit intended for minimization and negativity.

In roulette wheel selection, the particular individuals tend to be mapped in order to contiguous segments of the line, so that every individual's segment is every bit sized in order to its fitness. A random number can be generated as well as the individual whose segment spans the particular random number will be selected. The procedure repeats before the desired number of individuals is actually obtained (called mating population). This method is actually analogous with a roulette wheel together with each slice proportionally sized toward the fitness.

Algorithm for Roulette Wheel Selection Method

```
Step1: r := random number, where 0 =< r < 1;
Step2: sum := 0;
           for each
Step 3: individual i
                   {
Step 4:            sum := sum+p(choice = i);
Step 5:            if r < sum
                             {
Step6:                       return i;
                             }
```

The flow diagram for our proposed adaptive genetic algorithm is shown in Fig. 2.

The software architecture is represented in terms of UML class responsibility diagrams. The optimality can be proved based on the dependencies and responsibilities of the system. Here, we have used bank management application as an example system. It contains 47 responsibilities and 94 dependencies between them. A class diagram will clearly structures the functional responsibilities, as shown in Fig. 3.

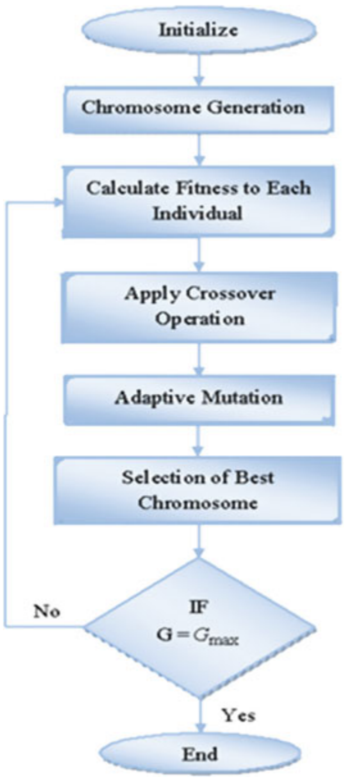
Initialize
Chromosome Generation
Calculate Fitness to Each Individual
Apply Crossover Operation
Adaptive Mutation
Selection of Best Chromosome
No
IF G = G_{max}
Yes
End

Fig. 2 Flow chart for AGA (adaptive genetic algorithm)

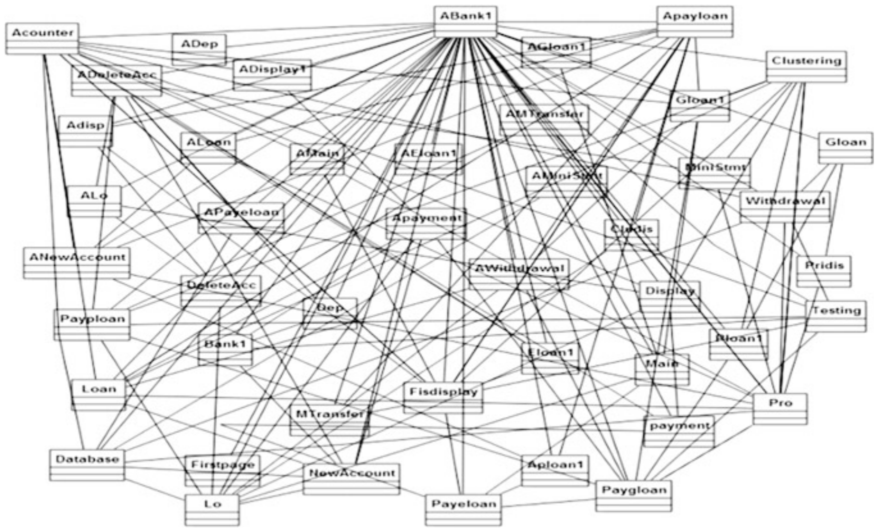

Fig. 3 The class diagram

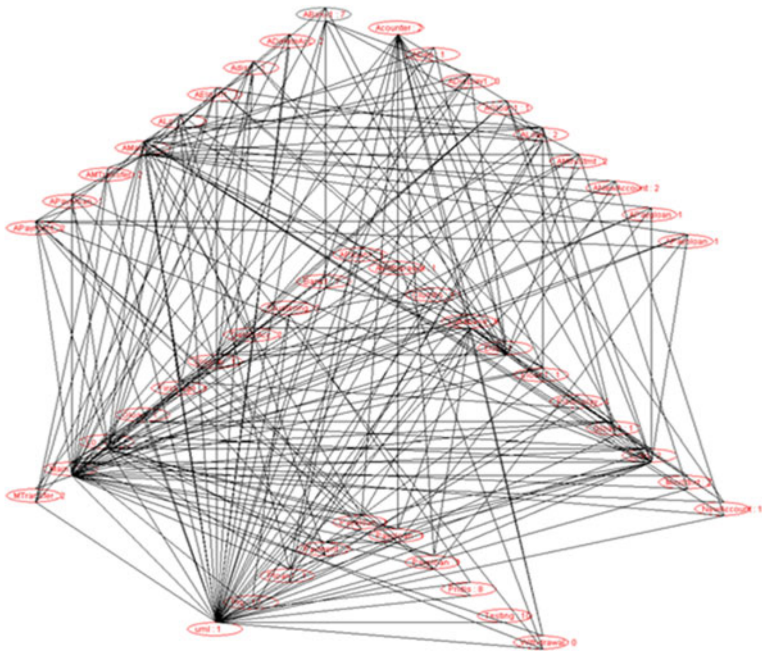

Fig. 4 Responsibility dependency graph

4.4 Generating Dependency Graph from Class Diagram

The concept of responsibility dependency graph is developed in order to efficiently represent the set of functional requirements of the software systems. The graph represents the functional requirements in the form of simple responsibilities. A responsibility can be defined as a job to be handled by either the complete software system or by a module or component of the software system. It can also be termed as a data object or item that has to be maintained in integrity by either the complete software system or by a module or component of the software system (Fig. 4).

Two basic elements are used in the responsibility dependency graph for the purpose of representation. As the name itself indicates, the graph represents responsibilities and the dependency relationships between the responsibilities. Responsibilities are denoted using an oval symbol which is technically termed as a node and the dependency relationships between responsibilities is represented by using directed edges. A directed edge specifies that the fulfillment of the source responsibility is dependent on the fulfillment of the target responsibility. Using this representation format, a responsibility dependency graph for any given software system can be developed. Once this graph is developed, properties such as path

Table 2 Responsibilities with dependency values

Responsibility number	Responsibility name	Depending value	Type
1	ABank1	48	F
2	Acounter	2	F
3	ADeleteAcc	2	F
4	ADep	2	F
5	Adisp	2	F
6	ADisplay1	2	F
7	AEloan1	2	F
8	AGloan1	3	F
9	ALo	3	F
10	ALoan	2	F
11	AMain	2	F
12	AMiniStmt	2	F
13	AMTransfer	2	F
14	ANewAccount	2	F
15	APayeloan	2	F
16	APaygloan	2	F
17	APayment	2	F
18	APayloan	2	F
19	APloan1	2	F
20	AWithdrawal	3	F
21	Bank1	48	F
22	Clusdis	2	F
23	Clustering	2	F
24	Database	2	F
25	DeleteAcc	2	F
26	Dep	0	F
27	Display	2	F
28	Eloan1	0	F
29	Firstpage	2	F
30	Fisdisplay	2	F
31	Gloan	2	F
32	Gloan1	2	F
33	Lo	2	F
34	Loan	2	F
35	Main	2	F
36	MiniStmt	2	F
37	MTransfer	2	F
38	NewAccount	1	F
39	Payeloan	2	F
40	Paygloan	2	F

(continued)

Table 2 (continued)

Responsibility number	Responsibility name	Depending value	Type
41	Payment	2	F
42	Payploan	2	F
43	Ploan1	3	F
44	Pridis	2	F
45	Pro	7	F
46	Testing	7	F
47	Withdrawal	2	F

coverage, dependency value, and statement coverage of the responsibilities are considered for the purpose of evaluating the software system.

For testing the given software system, identify the functional requirements in the form of responsibilities and their dependencies and having developed the responsibility dependency graph, a sample data is designed to test the system. Here, the type "F" represents the functional responsibilities. All the values are shown in the Table 2.

The bank management system contains 11 modules: login, customer, account, loan, bank, account, display, loan, clustering, transfer, and testing. These modules are independent of each other. On analysing these modules, a set of functional requirements have been identified which have been modeled as 47 responsibilities having 94 dependencies relating them. The details of the same are shown in Table 2. Effort is made to achieve a balance between complexity and structure of the graph. The parameter values are fine-tuned. Here, we illustrated path coverage, dependency values, and statement coverage values as taken from the above dependency graph.

5 Results of Final Test Case

Initial population in solution space is consisting of chromosome with length of two genes, i.e., {{23, 44}, {23, 44}, {23, 26}, {23, 26} {23, 26}}. Evaluation of fitness for each and every offspring chromosome in the solution space is show in Tables 3, 4, 5, 6, 7.

Based on the fitness values, identifying eminent functional responsibility is shown in Table 8.

From the above responsibilities, Table 8, eminent functional responsibilities are identified and represented as a class diagram as shown in Fig. 5.

After all iterations, the fitness values in the case study are constant; there is no longer generation of new offsprings. Hence, adaptive genetic algorithm is the best when compared to genetic algorithm.

Table 3 New offspring chromosome with fitness value

Iteration: 5		
Initial chromosome		Fitness values
23	44	781
23	44	781
23	26	720.5
23	26	720.5
23	26	720.5

Table 4 Crossover operation

Crossover chromosome	
23	44
23	44
23	26
23	26
23	44

Table 5 Adaptive mutation operation

Mutation chromosome		
23	2	1
23	13	1
23	44	1
23	26	1
23	44	1

Table 6 Over all fitness for all chromosome

S. no	Chromosome		Fitness values
1	23	44	781.0
2	23	44	781.0
3	23	26	720.5
4	23	26	720.5
5	23	26	720.5
6	23	44	781.0
7	23	44	781.0
8	23	26	720.5
9	23	26	720.5
10	23	44	781.0
11	23	2	551.0
12	23	13	571.0
13	23	44	781.0
14	23	28	597.5

Table 7 Eminent chromosome and fitness value

Chromosome		Fitness
23	44	781
23	44	781
23	44	781
23	44	781
23	44	781

Table 8 Eminent functional responsibilities

Responsibility no	Responsibility names
1	ABank1
2	Counter
3	ADeleteAcc
4	ADep
5	Adisp
6	ADisplay1

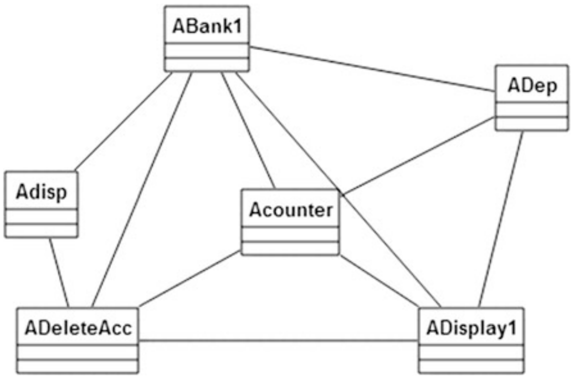

Fig. 5 Eminent functional responsibilities for test case

5.1 Performance Analysis

The whole effort of the genetic algorithms lies on the objective functions and fitness values selected. In each generation, the chromosomes with highest fitness values are chosen. In both the GA and AGA methodologies, in each iteration, the fitness value of each of the chromosomes is calculated and the results are tabulated. By observing the information, Table 9, it is concluded that our AGA technique achieved higher fitness values when compared to the GA technique.

The results shown in Table 9 and Fig. 6 prove the efficiency of the proposed adaptive genetic algorithm in producing a quick convergence scenario when compared to the traditional genetic algorithm.

Table 9 Fitness value for adaptive GA and GA for every iteration

Number of iteration	Fitness for adaptive GA	Fitness for GA
1	978	1123
2	970.2	1110
3	870	1065
4	781	1032
5	781	996
35	781	991

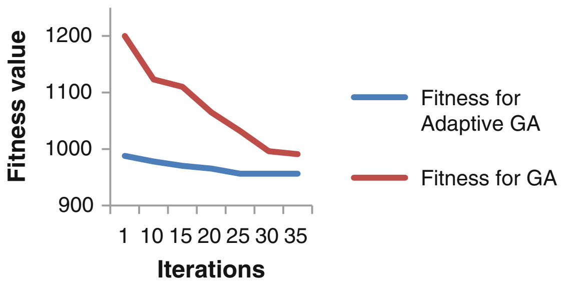

Fig. 6 Comparison of convergence between adaptive genetic algorithm and traditional conventional genetic algorithm

6 Conclusion

The implementation of the work starts from the selection of application with object-oriented approach. A UML class diagram consisting of responsibilities in generating the dependency values; these values are input for AGA. Further, this adaptive genetic algorithm has proved to be an efficient technique for software architecture. In the intended adaptive genetic algorithm, the chromosomes are separated into chromosomes to maintain assortment in the course of optimization process which demonstrated to be successful than ordinary genetic algorithm process.

Our recommended method has enhanced the computational time of the software which is the most important requirement in the software architecture. It also helps in comparing the presentation of every method which will be helpful to scheme improved software architecture. This paper has proposed one of the solutions for evaluating the fitness function dynamically to provide and identify eminent functional requirement to produce a good software architecture design.

References

1. Pan W. Applying complex network theory to software structure analysis. World Acad Sci Eng Technol. 2011;60:1636–42.
2. Abdelmoez M, Jalali AH, Shaik K, Menzies T, Ammar HH. Using software architecture risk assessment for product line architectures. In: Proceedings of. international conference on communication, computer and power (Icccp'09); 2009. Muscat, Feb 15–18, 2009.
3. Yang XS, Deb S. Engineering optimization by cuckoo search. Int J Math Model Num Optim. 2010;1(4):330–43.
4. Frey S, Fittkau F, Hasselbring W. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In: 2013 35th international conference on software engineering (ICSE); 2013, 18–26 May 2013.
5. Zitzler E, Deb K, Thiele L. Comparison of multiobjective evolutionary algorithms: empirical results. Evol Comput. 2000;8(2):125–48.
6. Rela L. Evolutionary computing in search-based software engineering, Lappeenranta University of Technology, Department of Information Technology, M.Sc. Thesis; 2004.