A Genetic Algorithm Approach to Software Components Search using Software Functional Requirement Checklist

Wiwat Vatanawood[1] and Prabhas Chongstitvatana[2]
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University, Bangkok 10330 Thailand
Email: wiwat@chula.ac.th[1], prabhas@chula.ac.th[2]

Abstract

In this paper, we present genetic algorithm for software components searching at the early stage of conceptual design. We propose an alternative to convert software functional requirements checklist, a mandatory checklist in software requirements specification document produced by software analyst, into a set of computational operations which is needed as the key to search for software component candidates from component libraries. Moreover, component searching is practically performed to select not only the most relevant candidate but also several feasible sets of component candidates that are capable to satisfy all expected operations.

In our approach, it is considered as an optimization problem to search a set of software component candidates that is most fit to given constraints. To consider at least N software components and at least T operations expected to satisfy the functional requirements of an application, the component search space is exponential. In our experiments with synthetic data, we investigated the case of $N = 250$ and $T = 250$. The results demonstrated that genetic algorithm is applicable and produces a set of solutions along with the order of merit.

1. Introduction

Demands for higher software production and lower maintenance cost with expected quality can be met by reusing approval existing software components. Reuse technique is widely accepted, especially for complex and large-scale software applications. Recently, a brand new software development process called Component-Based Software Development (CBSD) is introduced, mentioned in [1,2]. In CBSD, software analysis and design phase in software development life cycle become more critical since a software application is expected to be constructed by set of existing software components and their interconnections without writing program code from scratch. In practical, the number of software components tentatively increases as time goes by and large numbers of software components are desirable in order to provide software designer with variety of possible choices. However, it is very difficult and highly time-consumed to search a relevant set of expected components and their substitutes from a large number of existing components.

In this paper, we present genetic algorithm for software components searching at the early stage of conceptual design. We propose an alternative to convert software functional requirements checklist, a mandatory checklist in software requirements specification document produced by software analyst, into a set of computational operations which is needed as the key to search for software component candidates from component libraries. Firstly, each of software functional requirements is decomposed into a set of preconditions and associated actions – using decision table technique. Secondly, all of the actions decomposed are converted into a set of computational operations using Action-to-Operation

mapping table. In general component libraries, a software component API is visible for developers and described in terms of component's operations, also called methods, and their interface parameters. Thus, component searching is practically performed to select not only the most relevant candidate but also several feasible sets of component candidates that are capable to satisfy all expected operations. All of the entries in software functional requirements are implicitly satisfied and make software development efforts more predictable at the early stage of conceptual design before proceeding its implementation.

In our approach, it is considered as an optimization problem to search a set of software component candidates that is most fit to given constraints - minimum cost and development efforts. This paper is organized as follow: Section 2 briefly describes software functional requirements and how to define preconditions and associated actions in decision table. Section 3 defines Action-to-Operation and Component-to-Operation mapping tables. The experimentation on searching software component candidates using genetic algorithm is reported in section 4. Section 5 is our conclusion.

2. Software Functional Requirements Checklist

In general, software requirements specification document (SRS) is expected to be complete before pursuing software design phase. Each functional behavior of target software application is collected from end-users and documented. In our approach, software functional requirement checklist, which is a mandatory section in SRS, is exploited. Each of functional requirements is written in either English language or Thai language is considered as a rule in decision table, shown in figure 1. [3] mentions decision table as a perfect tool to describe a complicate event happened. Software analyst may use decision table to identify a set of required associated actions to satisfy specified any combination of defined preconditions happened in the software system. With this alternative of describing, the requirement specifications are unambiguously defined.

To be formally, we can describe software functional requirement checklist by using a decision table, as a set of rules (x,y) as follow:

$$\begin{aligned} \textit{Specification} = \{\ (x,y) : \textstyle\prod \textit{Precondition} \times \prod \textit{Action} \mid (\forall x: \prod \textit{Precondition}; \\ y_1,y_2: \textstyle\prod \textit{Action} \infty\ (x \mapsto y_1) \wedge (x \mapsto y_2) \Rightarrow y_1 = y_2)\ \} \end{aligned}$$

where *Specification* is a set of many-to-one relations from power set of *Precondition* to power set of *Action*; *Precondition* is a set of conditions which is valid in any particular time and *Action* is a set of associated actions to be performed. For example, ({"class opens", "a new student registers"}, {"add student in the class", "print class schedule"}) is a rule describing how to enroll a student for a class, etc.

Considering specification rules defined, the possible number of rules is exponential. Thus, it means a decision table is efficient enough to define a complicate specification rule. However, it is still not simple to practically define preconditions and actions.

3. Define Mapping Table

All marked actions from specification rules, called target actions, are gathered and used as keys to search target components. In our approach, both actions and components are defined as a set of computational operations so that a set of software components is possibly

searchable as to perform all of operations needed in target actions for a software application. Each action is mapped into a set of operation using Action-to-Operation mapping table, shown in figure 2.

Given a set of computational operations called *Operation*, Action-to-Operation (*ActionTab*) and Component-to-Operation (*ComponentTab*) mapping table, shown in figure 2 and 3, are formally defined as follow:

$$ActionTab = \{ (a,b) : Action \times \prod Operation \mid (\forall a: Action; b_1,b_2: \prod Operation \infty (a \mapsto b_1) \wedge (a \mapsto b_2) \Rightarrow b_1 = b_2)\}$$

$$ComponentTab = \{ (c,d) : Component \times \prod Operation \mid (\forall c: Component; d_1,d_2: \prod Operation \infty (c \mapsto d_1) \wedge (c \mapsto d_2) \Rightarrow d_1 = d_2) \}$$

where *Component* is a set of components to be selected. The mapping tables are simply a set of many-to-one relations from either action or component to a set of operations. Our approach for searching is to find a set of final components, called *TargetComponent*, which perform all of target actions, called *TargetAction*. We further our formal descriptions as follow:

$$TargetAction == \cup(\text{ran } Specification)$$

$$TargetComponent = \{ c : Component \mid (\forall c: Component; d_1,d_2: \prod Operation; a: TargetAction \infty (c \mapsto d_1) \in ComponentTab \wedge (a \mapsto d_2) \in ActionTab \Rightarrow \cup d_1 = \cup d_2)\}$$

Genetic Algorithm technique is used to search several *TargetComponent* sets with minimum cost as final results. For any component c_1 and c_2 selected within the same set *TargetComponent*, both components may provide the capability of performing the same operation.

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	..	Rule R
Precondition 1	X	X					X
Precondition 2		X	X				
Precondition 3	X		X	X			X
...							
Precondition P					X		X
Action 1	X						
Action 2		X	X				X
Action 3	X	X			X		
...							
Action A			X		X		

Figure 1: An example of decision table.

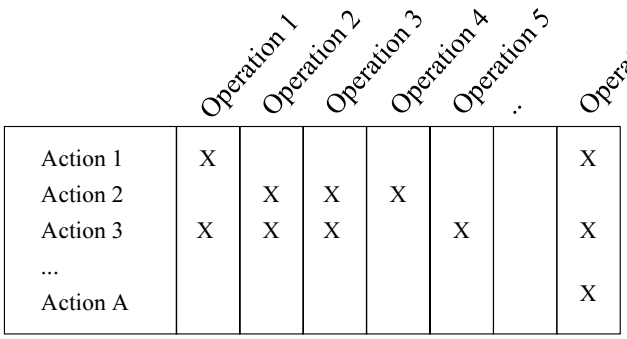

Figure 2: An example of Action-to-Operation mapping table

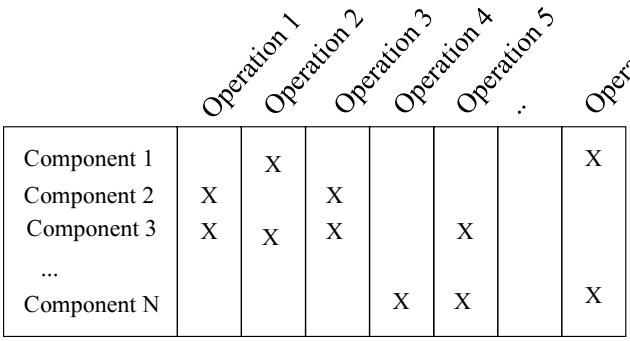

Figure 3: An example of Component-to-Operation map table

4. Experiments and Results

In our experiments, genetic algorithm is expected to illustrate the ability of components searching and propose the best target component set for software designer with minimum estimated cost of efforts. However, since the capability of a component may vary, we investigate the effect of both special-purpose components and multi-purpose components to component search as well. Special-purpose and multi-purpose components are formally defined as follow:

$$SpecialComponentTab = \{ (c,d) : Component \times \prod Operation \mid (\forall c: Component; d_1,d_2: \prod Operation \infty (c \mapsto d_1) \wedge (c \mapsto d_2) \Rightarrow d_1 = d_2 \wedge \#d \leq 5) \}$$

$$MultiComponentTab = \{ (c,d) : Component \times \prod Operation \mid (\forall c: Component; d_1,d_2: \prod Operation \infty (c \mapsto d_1) \wedge (c \mapsto d_2) \Rightarrow d_1 = d_2 \wedge \#d > 5 \wedge \#d \leq 10) \}$$

where *SpecialComponentTab* and *MultiComponentTab* denote component mapping tables and # denote the number of members in a set (Z notation in [4]).

Using genetic algorithms for searching, an objective function is required as to evaluate the survival set of each generation. Our cost model defines an equation to calculate estimation of cost of efforts. A set of components with minimum cost of efforts is the survivor.

Specification rules, components and operations are synthetically built – at least 250 components and 250 computational operations are expected.

4.1 Genetic Encoding

GA problems are typically encoded as an n-digit string that represents a complete solution to the problem. In our experiments, a chromosome is composed of *N* tuples. Each tuple is defined as three attributes: action, operation and component as $(act_i, op_i, comp_i)$. Figure 4 shows a chromosome of *N* tuples.

(act1, op1, comp1)	(act2, op2, comp2)	….	(act*N*, op*N*, comp*N*)

Figure 4: Chromosome encoding as a stream of tuple *(action, operation, component)*

4.2 Genetic Parameters

In our experiments, we applied genetic parameters as shown in Table 1

Table 1: Genetic parameters being used in GA experiments

Parameter	Value
Selection Strategy	Tournament size 7
Mutation Strategy	Change one gene
P_c Probability of Crossover	1.0
P_m Probability of Mutation	0.1
Population	5,000

Unlike the typical genetic algorithms, initial sets of chromosomes are randomly generated with the possible solutions from *SpecialComponentTab*, *MultiComponentTab*, and *ActionTab* defined earlier. Thus, each chromosome in any generation is always feasible solution. During the experiments, the crossover and mutation techniques always preserve the feasible chromosome as well.

4.3 Cost Model

We simply define a cost estimation function *Cost*, in terms of indirect efforts to be used as follows:

$$Cost = \sum_{Component_i \in Target Component} (C1(Component_i) + C2(Component_i) * (OccurCount(Component_i) - 1))$$

where *C1* and *C2* calculate initial effort and modification effort respectively. *OccurCount* returns the number of modification for each $Component_i$. Figure 5 shows sample of initial effort and modification effort for *C1* and *C2*. We simply assume that initial effort of reusing each component varies with the number of computational operations served, while modification effort of each component is approximately one-third of initial effort.

Component Name	Number of Operations served	Initial effort for *C1* (man-hour)	Modification effort for *C2* (man-hour)
Component 1	5	25	8
Component 2	3	18	5
…	…	…	…
…	…	…	…
Component N	12	36	11

Figure 5: Sample of initial and modification effort of components

4.4 Conducting Experiments

We investigate 10 cases of *Specification* which are randomly generated with experiments settings in table 2. Each action in *Specification* performs 1 to 5 operations. Special-purpose and multi-purpose components are expected to serve 1 to 5 operations and 6 to 10 operations respectively. Special-purpose component is called fine-grain component that is developed for special-purpose operations while multi-purpose component is called coarse-grain component. Multi-purpose component is developed for multi-functional tasks and more efforts are expected to reuse once.

Table 2: Experiments settings

Experiments Settings	Value
Specification rules	10 different synthetically built cases
Each action performs	1-15 operations
Each special-purpose component serves	1-5 operations
Each multi-purpose component serves	6-10 operations
Component's initial cost of effort	10-1,000 man-hours
Component Library to be searched (Special-purpose:Multi-purpose)	150:100

Table 3: Minimum cost of efforts purposed by GA

Spec.	**Average Initial Cost**	**Average Reuse Cost**	**Average Total Cost**	**Actions expected**
Case 1	22,953	38,328	61,281	848
Case 2	22,803	36,470	59,273	780
Case 3	20,840	30,105	50,945	690
Case 4	23,117	37,507	60,624	792
Case 5	25,229	42,197	67,426	870
Case 6	23,452	40,929	64,381	817
Case 7	21,505	33,439	54,944	775
Case 8	21,949	33,425	55,374	765
Case 9	24,579	41,783	66,362	883
Case 10	25,098	28,360	53,458	734

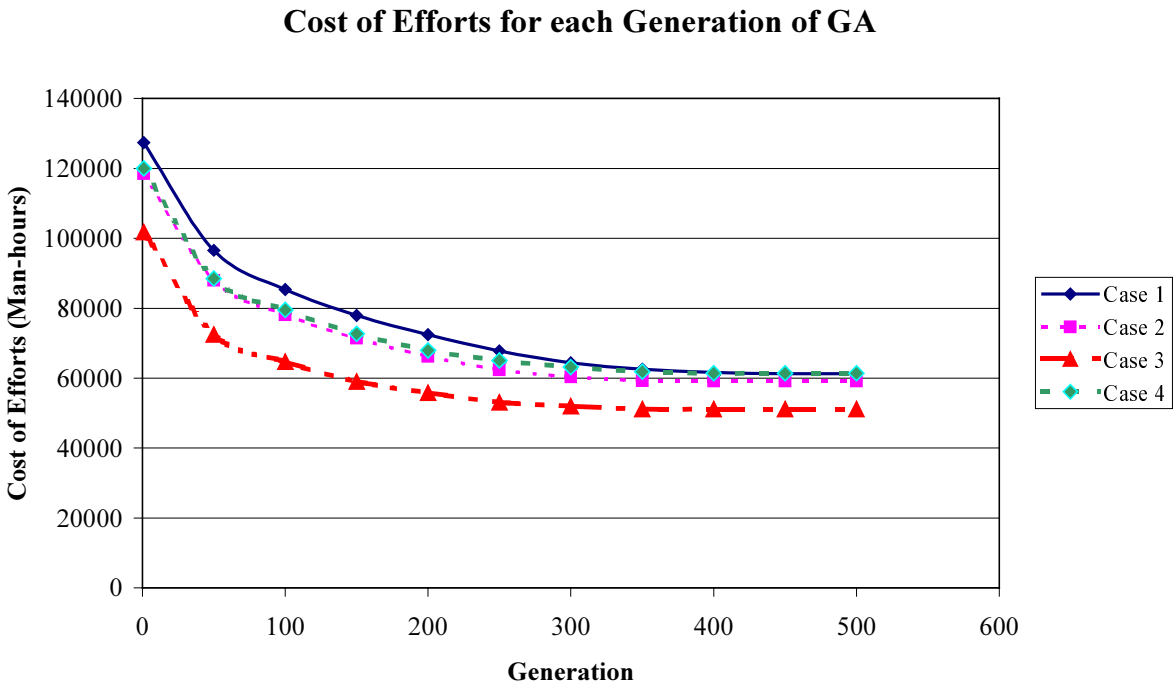

Figure 6: GA shows the prediction of minimum cost of efforts for case 1 to 4

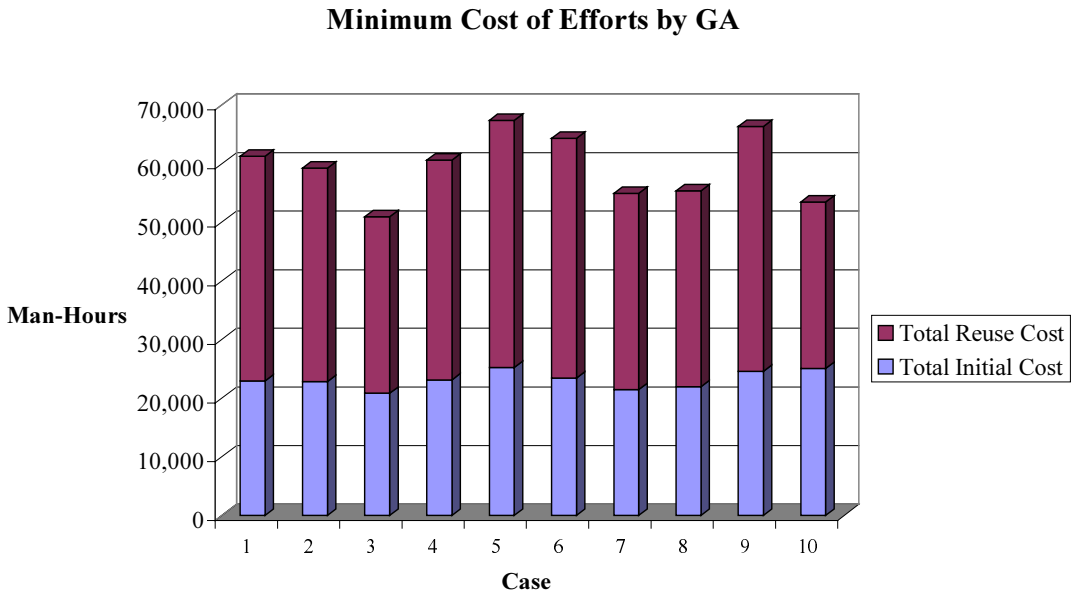

Figure 7: Cost of efforts by GA for 10 cases of specification rules

4.5 Results

As shown in the results in figure 6, genetic algorithm performs a remarkable software component searching. The cost of efforts decreases and converges at a stable value in the

500^{th} generation. Software designer is able to predict the minimum effort to be invested for developing an application. As shown in table 3, cost of efforts is approximately 60,000 man-hours for an application of size 800 operations. Cost of efforts on reusing selected components is approximately 60% of total cost. GA illustrates the ability of selecting the solutions from both *SpecialComponentTab* and *MultiComponentTab* table.

5. Conclusions

As demonstrated in the experiments, software designer may formulate a searching problem to select a set of relevant software components with any constraints required. This approach provides several sets of target components for software architectural design rather than only one best solution. Moreover, each target set of components is proposed in the order of merit so that software designer is able to select any substitute and the search process is repeatable with minimum efforts if any constraint or specification rule is adjusted. The searching problem is applicable by simply using direct encoding method as experimented. The results are satisfactory.

References

[1] P. Clements, "From Subroutines to Subsystems: Component-Based Software Development", *American Programmer*, November 1995

[2] D. Garlan, D. Perry, "Introduction to the Special Issue on Software Architecture", *IEEE Transactions on Software Engineering,* April 1995

[3] J. FitzGerald, A. FizGerald. "Fundamentals of Systems Analysis: Using Structured Analysis and Design Techniques", John Wiley & Sons Inc., 1987

[4] J. Bowen. "Formal Specification & Documentation Using Z: A Case Study Approach", International Thomson Computer Press, 1996.