European Journal of Operational Research 212 (2011) 301–311

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Production, Manufacturing and Logistics

An optimization model for software component selection under multiple applications development

J.F. Tang [a,*], L.F. Mu [a], C.K. Kwong [b], X.G. Luo [a]

[a] Dept of Systems Engineering, Key Lab of Integrated Automation of Process Industry of MOE, Northeastern University, Shenyang 110004, PR China
[b] Dept of Industrial and Systems Engineering, The Hong Kong Polytechnic University, Hong Kong, PR China

ARTICLE INFO

Article history:
Received 21 May 2009
Accepted 24 January 2011
Available online 1 February 2011

Keywords:
Component based software development
Components selection
Reusability
Multi-applications development
Optimization model
Experimental analysis

ABSTRACT

Component based software development (CBSD) is well acknowledged as a methodology which establishes reusability of software and reduce development cost effectively. While developing enterprise application using component based software engineering (CBSE) methods, software component selection plays a very important role in the process of component retrieval, adaptation and assembly. However, most of current researches focus on technical aspects from domain engineering and application engineering to improve reusability and system efficiency rather than application of optimization methods in CBSD management, especially application in component selection. Moreover, few existing researches have concerned about the situation where a software developer or enterprise develops multi-applications at the same time. By introducing the concept of reusability and a new formulation of compatibility matrix, an optimization model is proposed to solve component selection problem considering reusability and compatibility simultaneously. The model can be used to assist software developers in selecting software components when multi-applications are undertaken concurrently. Four experiments are conducted with the purpose to provide some insights in management perspective.

1. Introduction

For easing pains and distresses given by complexity of software systems, software engineers have explored many techniques to improve software reusability and maintainability, e.g. introduction of function, software modularization, and objective oriented techniques. In recent years, as a promising development approach, component-based software development (CBSD) has greatly changed the scenario that software engineers are faced with [1–3]. Grounded in the concept of component fabrication and assembly, CBSD can help the software industry realize quality and productivity gains similar to those achieved in the hardware and manufacturing industries [4–6].

CBSD focuses on building large software systems by integrating previously-existing software components (for the sake of simplicity, term "component" is used to replace the term "software component"). A component is a system unit offering a predefined service or event, and able to communicate with other components. A component fulfills five attributions: multiple-use, non-context-specific, composed with other components, encapsulated, a unit of independent deployment and versioning. Third party component is a reusable component developed to be either freely distributed or sold by an entity. The third components available in component market are name available components (ACs) in the paper.

CBSD embodies the "buy, do not build" philosophy espoused by Fred Brooks [7], and it is also referred to as component-based software engineering (CBSE) [8]. With CBSD, certain parts of large software systems reappear with sufficient regularity that common parts should be written once, rather than many times, and common systems should be assembled through reuse rather than rewritten over and over. Thus, CBSD can potentially be used to reduce software development costs, assemble systems rapidly, and reduce the spiraling maintenance burden associated with the technical support and upgrade of large systems.

CBSD applications and practices have grown rapidly, and consequently the worldwide component market has grown rapidly. More and more commercial components surged into marketplace [9]. For example, so far, componentsource.com (an online reseller of components and tools for all platforms) provide 1318 components to meet different business requirements. These components are implemented using diverse technologies by various organizations, depend on different platform and provide numerous services. Even though some components provide similar function and can work in the same circumstance, they usually present

* Corresponding author. Tel.: +86 24 83673810.
E-mail address: jftang@mail.neu.edu.cn (J.F. Tang).

0377-2217/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2011.01.045

various compatibilities and require different costs to assemble them into an application. It becomes gradually complex to select a set of components to establish a high quality enterprise application with low cost and make components collaborate perfectly with each other. A few researchers have realized these paucities and have contributed excellent works on the topic [1,10–15] to support component selection process.

To design efficient algorithms to automate component selection, Haghpanah adopted a greedy approach and a genetic algorithm to select a minimal cost set of components to satisfy a set of objectives [11]. Considering components can be physically located in different repositories in Internet, Abraham developed one stochastic model for intelligent selection of components [10]. Bhuta described a framework for selecting COTS components and connectors ensuring their interoperability in software intensive systems with minimal effort [12]. Michael evaluated some of the current component specification techniques with respect to the needs of component selection and reuse [13]. In order to select the appropriate ones for a specific software system, Ardimento et al. proposed a characterization of components aimed at foreseeing the maintenance effort of the component-based systems [14]. Most of these literatures focused mainly on technical details or methods of component selection, management insights provided by models and approaches generally are ignored. However, instead of trying to penetrate technologies deeply, there are a few researches concentrating on the management implications extracted from sensitive analysis of the proposed planning models. For example, In order to reduce the total cost, Sundarraj proposed an application of optimization for the planning of reusable components, and presented a model that selected a set of components that must be built [1]. In his another work, a multi-period integer-programming model was proposed to assist decision-makers in the procurement of components over a given planning horizon [15]. He provides excellent implication to project management based on simulation in both his works. His research methodology was employed in our research. In contrast to all previous literatures that consider single development task, a software developer which undertakes multiple development tasks of enterprise application concurrently, is considered in our research. The proposed model also can help software developers to reduce cost through selecting appropriate components into enterprise applications.

Although several previous researches have concerned with component selection problem of CBSD, relatively few formal methods and techniques were introduced to consider compatibility in the process of component selection and assembly. Jung and Choi made an outstanding contribution to fill this gap [16]. They employed either-or constraints to express compatible relationship between software modules. However, there are two deficiencies in their software module selection approach: misunderstanding of reusability and weak operability of compatibility expression. To make up the deficiencies, another definition of reusability and a new formulation of compatibility matrix are introduced to model component selection problem under multiple application development in our work.

The rest of the paper is organized as follows. A flowchart of CBSD process and a three-layer's structure model of component selection problem under multiple application development as well as new formulation of compatibility are explained in Section 2. Mathematical description of the problem and a planning model are introduced in Section 3. In Section 4, the proposed model is tested using simulated data, experiments are conducted and conclusions are drawn from experiment results. Finally, some concluding remarks and perspectives are given in Section 5.

2. Component selection problem in multiple application development

2.1. A flowchart of CBSD process

Reusable components make it possible to divide the application development and deployment process into distinct roles so that different people or companies can perform different parts of the process. Most of researches mentioned the CBSD life cycle as four phases including: Domain analysis, Component design and development, Component cataloging and retrieval, component selection and application assembly [4]. During the development of a CBSD software product, available components (ACs) are assembled or adapted to an application with some development tools, e.g. assemble tools, packaging tools, operating systems, DBMS, application server. The providers of these tools or architectures play quite distinct roles and collaborate together to deliver the final software system. These software organizations, according to the roles they play, can be classed into: Component Developers, Tool Providers, Environment Providers, Component Broker, Software Developers (some literature also name them as Application Assembler) [17]. The overall procedures of CBSD application software and providers involved as well as their roles are given in Fig. 1.

The Environment providers are typically operating system, database system, application server, or web server vendors. The tool provider is the company or person who creates development, assembly, and packaging tools used by component providers and software developers. The component provider is the organization which extracts general business logic and designs, develops and delivers components. Component broker is the group who evaluates and procures components from component developers, and builds a component library or store. Sometimes, a component broker also designs and develops components himself. A software developer is the company who procures components from component providers or component brokers, then assembles them together to implement applications according to customer requirements. The proposed model in this paper is introduced for software developers to assists them in selecting components and assembling them into a final system.

2.2. Understanding of component reusability

According to definition of reusability of a component [18,19], reusability is the extent to which parts of a software can be reused in other applications. Software reuse means to abstract the general logic from different applications, implement the logic, and be used into more applications (rather than only one application) with slight or no modification. Jung and Choi [16] considered that, software modules or "parts of software" are reused repeatedly in the same software system. However, in practice, that is nearly not happened in a mature software design. In other words, that is strongly not recommended in CBSD.

In CBSD, if a component is deployed or adapted into an application many times, in the maintenance process, all deployed components in the application will have to be found and modified one by one, which lead to huge maintenance efforts. If so, the software reuse techniques born for reducing design and maintenance efforts will not make his original sense any more. There also exists another important reason why it is not necessary to deploy a component into the same application repeatedly. The delivered component is a package of class files, which will be concrete objects only after the application invokes and instantiate them. Through these instantiations, many objects can be generated from a relevant class repeatedly if necessary. Therefore, one component is enough for an application, and the application will decide when and which

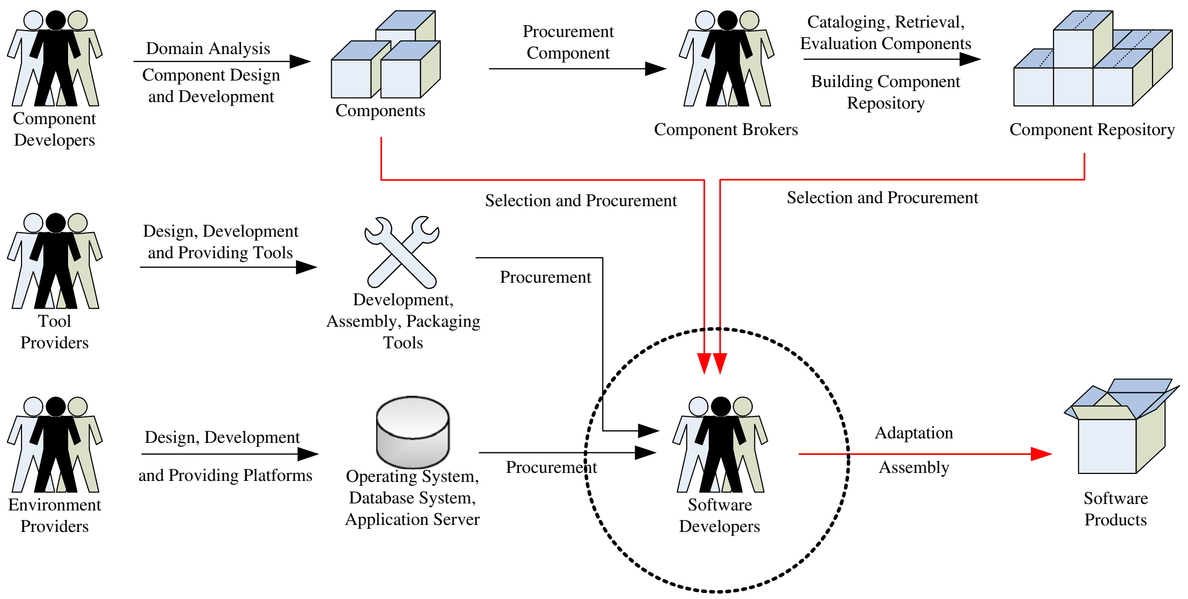

Component Developers
Domain Analysis
Component Design and Development
Components
Procurement Component
Component Brokers
Cataloging, Retrieval, Evaluation Components
Building Component Repository
Component Repository
Selection and Procurement
Selection and Procurement
Tool Providers
Design, Development and Providing Tools
Development, Assembly, Packaging Tools
Procurement
Environment Providers
Design, Development and Providing Platforms
Operating System, Database System, Application Server
Procurement
Software Developers
Adaptation
Assembly
Software Products

Fig. 1. A flowchart of CBSD process.

component objects are created and invoked to provide services. That is also one of important differences between the components in software industry and in manufacturing industry. It is obvious that some researchers confused "adapting" with "invoking". In this research, the component reusability among applications is introduced correctly to the proposed model.

2.3. A new formulation of component compatibility

In practices, not all of components can be integrated together in an application [16]. Therefore the compatibility among components has to be considered in the process of component selection. A component's behavior can be represented by the set of all possible sequences of services (required or provided by a component). Two components are compatible if all possible sequences of services requested by one of the interacting components can be provided by the other component [20].

Jung and Choi [16] employed either-or constraints to express compatible relations between components. Nevertheless, there are generally a large amount of compatible relations among components in practices, and thus the number of either-or constraints will explode rapidly. It is really not operable to use either-or constraints in modeling a practical selection problem. For instance, one AC has been selected into a system, and there are other lACs are compatible with it, but only one of them can be selected into the system. If either-or constraints are employed to express the condition, l + 1 constraints will introduced into the model.

For the sake of operability and simplicity in modeling, a compatibility matrix $R_{L\times L}$ amongst components is introduced in our work. A binary element that lies in the kth row and the k'th column of $R_{L\times L}$ refers to the compatibility relation between the$k$th AC and the k'th AC.

2.4. Component selection problem under multiple applications

For concentrating on the problem, it is assumed that all enterprise applications can be implemented by assembling pre-existing components, besides procured components, a few components might be developed in-house in practice. This particular problem will be discussed in future research.

In general, a software developer is undergoing multiple applications concurrently, e.g. N, in CBSD environment. Fig. 2 illustrates a typical component selection problem while developing multi-application. After system analysis, it has been decided how many and what kind of components should be assembled into applications. These components are required to deploy logically into applications in design process, and are referred as required components (RCs) hereafter. As shown in Fig. 2, it needs total number of M RCs to cover N applications and the ith application requires m_i RCs. Therefore, $M=\sum_{i=1}^{N} m_i$. Each RC belongs to an application uniquely. For example, in Fig. 2, the first four RCs belong to the first application, and the 5th to 7th RCs are assembled to the 2nd applications, and so on. It is assumed that there is no common RCs for different applications. For the sake of simplification, an assumption is made: During implementation, each RC will be fulfilled through selecting and adapting one and only one AC from component market or some component providers. In practice, a RC definitely may be achieved by composition of several ACs. It is not considered in the research or the sake of simplicity. With regard to this condition, for using the proposed model, the RC need to be decomposed further until it can be fulfilled by only one AC. Let $c_k^p,\ k=1,\dots,L$ be procurement cost of the kth AC, where L is the total number of AC. Of course, there may be more than one alternative AC in the component market to fulfill a specified RC, and each has its own adaptation costs. Let $c_{jk}^a,\ j=1,\dots,M,\ k=1,\dots,L$ denote the adaptation cost that the kth AC is assembled into the jth RC. However, only one AC among them will be selected to implement the RC finally during the process of implementation. There exists compatible or conflicts among these ACs; for example, the 1st AC is compatible with 5th AC, and the latter is with 8th AC in Fig. 2. In general, to take advantage of reusability technique, an AC can achieve several needs of RCs belonging to different applications through certain adaptation efforts. Considering compatibility of components, in each undertaken application, all selected AC should be guaranteed to work conflict-free.

To explain the relationship between application and RCs and between RCs and ACs clearly, a simple example as shown in Fig. 3 is considered. A software development organization undertakes two enterprise applications for a bookseller and a airline tickets agent. In general, software engineers will construct the architecture of application after requirement analysis. According to customer requirements, online bookstore application should support online payment function and online airline tickets booking system should provide online electronic check service. Because

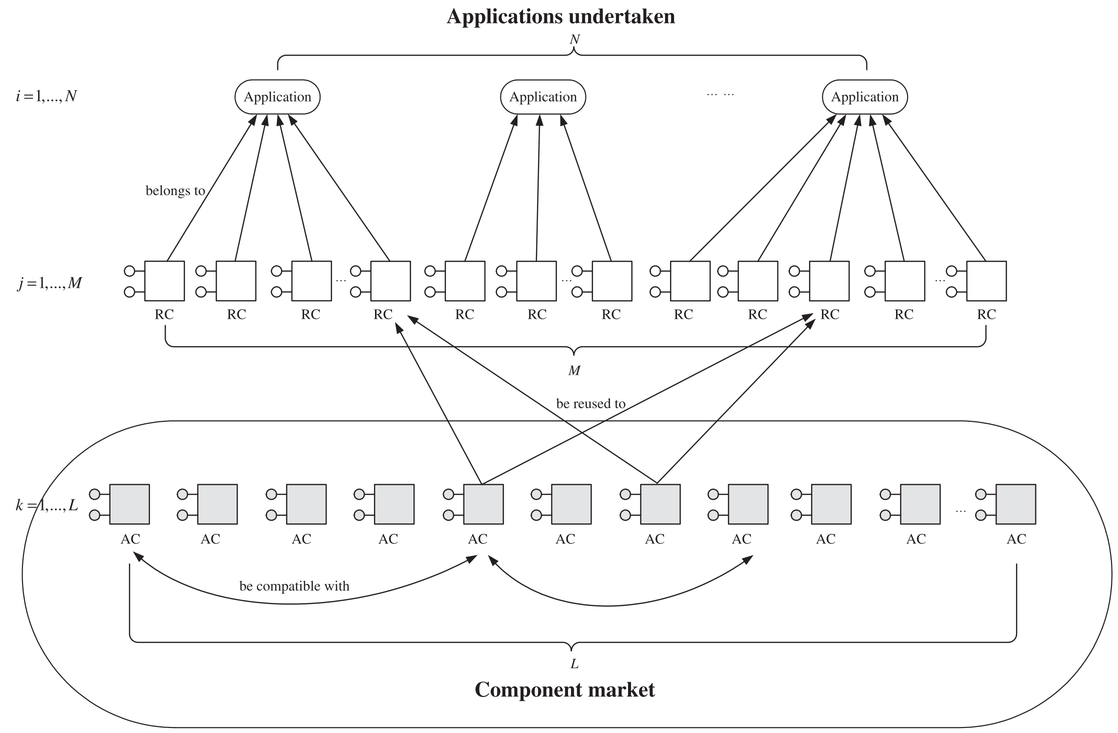


Fig. 2. Three-layers of component selection problem with multiple applications.

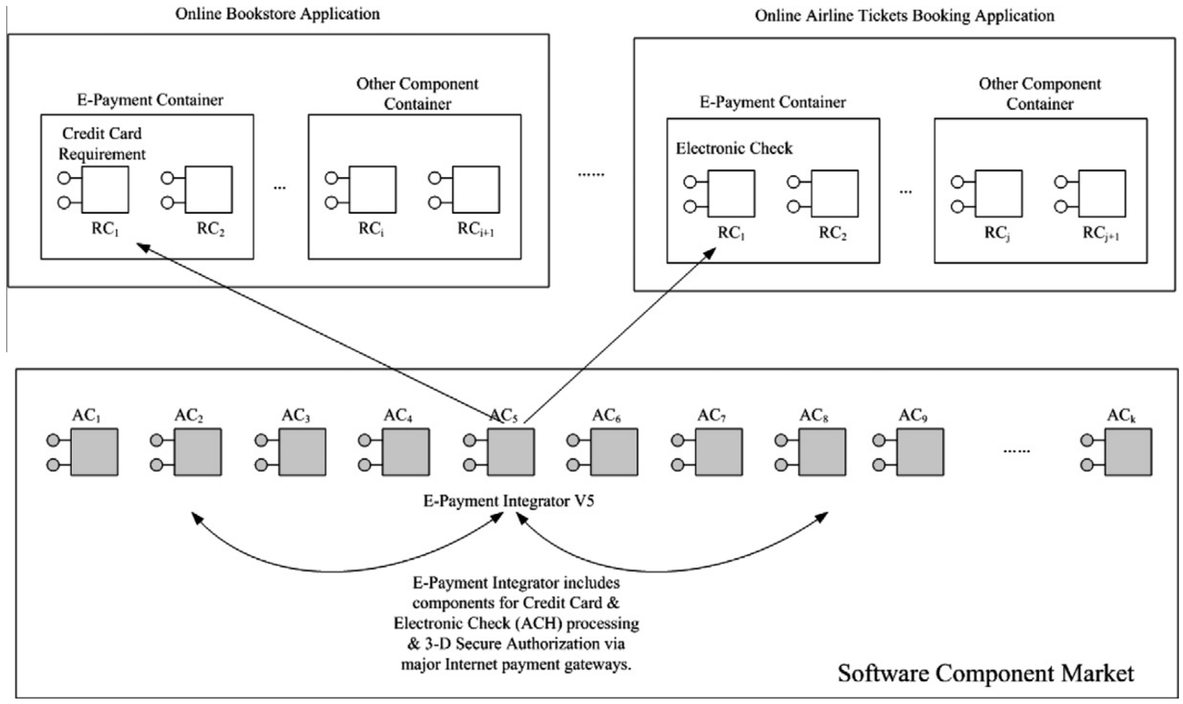

Online Bookstore Application
E-Payment Container
Credit Card Requirement
RC1
RC2
Other Component Container
RCi
RCi+1
Online Airline Tickets Booking Application
E-Payment Container
Electronic Check
RC1
RC2
Other Component Container
RCj
RCj+1
AC1
AC2
AC3
AC4
AC5
AC6
AC7
AC8
AC9
ACk
E-Payment Integrator V5
E-Payment Integrator includes components for Credit Card & Electronic Check (ACH) processing & 3-D Secure Authorization via major Internet payment gateways.
Software Component Market

Fig. 3. An example for explaining relationship between RC and AC.

two requirements are two tight cohesion function units and it is highly possible to achieve them by using preexist commercial component in market, software engineers can treat them as two independent RCs, and deploy them into two component containers (containers can provides a run-time environment for components within the application server) belong to two applications respectively. However, even through after careful and systemic design details of interfaces and services of RCs, they still just stay on the process of design. For implementing them concretely, software engineers should select and procure appropriate ACs from component market. The selected components will be deployed into relevant component containers in the architecture to fulfill all function requirements of RCs. This deployment is referred sometimes as "gluing". It includes generally: creating required class files, interface files, and control information, analyzing input files, generating necessary classes, etc. Moreover, not all of components can be deployed directly into containers, and some of them might need to be re-encapsulated further before assembling them into containers.

Thus, the component selection problem addressed in this paper is how to select ACs and deploy into which applications, to minimize the total costs of adaptation cost of ACs to RCs and procurement cost of ACs on all undertaken applications, considering the compatibility among ACs.

3. Mathematical formulation of component selection problem

The cost of component procurement and adaptation cost are considered to be development costs of an application in this paper. To formulate the problem mathematically, notations are introduced as follows:

N	The number of enterprise applications which a software developer undertakes concurrently.
M	The number of RCs which all N enterprise applications include.
L	The number of all ACs.
s_{ij}, $i = 1,\ldots,N$, $j = 1,\ldots,M$	Binary parameter; $s_{ij} = 1$ denote the jth RC belongs to the ith application; otherwise, $s_{ij} = 0$. The matrix $S = [s_{ij}]_{N\times M}$ is named membership matrix. It represents "belong to" relation between RCs and applications.
$c_k^p,\ k = 1,\ldots,L$	The procurement cost of the kth AC.
$c_{jk}^a,\ j = 1,\ldots,M,\ k = 1,\ldots,L$	The adaptation cost that the kth AC is assembled into the jth RC.
$r_{kk'},\ k,\ k' = 1,\ldots,L$	Binary compatibility parameter; $r_{kk'} = 1$ denotes that the kth AC is compatible with the k'th AC; otherwise, $r_{kk'} = 0$. Because the kth AC is compatible with itself, $r_{kk} = 1$, $k = 1,\ldots,L$, and $r_{kk'} = r_{k'k}$ because compatibility relationships are symmetric. The matrix $R = [r_{kk'}]_{L\times L}$, is named as Compatibility Matrix.
b_{jk}, $j = 1,\ldots,M$, $k = 1,\ldots,L$	Binary parameter; $b_{jk} = 1$ denotes that the kth AC can be reused to achieve the jth RC; otherwise, $b_{jk} = 0$. The matrix $B = [b_{jk}]_{M\times L}$ is Reuse Matrix. It represents "can be adapted to" relation between ACs and RCs.
x_{jk}, $j = 1,\ldots,M$, $k = 1,\ldots,L$	Binary decision variable, $x_{jk} = 1$ means that the kth AC is selected to implement the jth RC; otherwise, $x_{jk} = 0$.
y_k	It is an auxiliary variable. $y_k = 1$indicates the kth AC is selected; otherwise, $y_k = 0$.

With these notations and assumptions, the component selection problem under multiple applications is formulated as the following programming model:

$$\min \sum_{k=1}^{L} c_k^p y_k + \sum_{k=1}^{L}\sum_{j=1}^{M} c_{jk}^a x_{jk}, \tag{1}$$

$$\text{s.t.} \sum_{k=1}^{L-1}\sum_{k'=k+1}^{L} r_{kk'}\left(\sum_{j=1}^{M} s_{ij}x_{jk}\right)\left(\sum_{j=1}^{M} s_{ij}x_{jk'}\right) = \binom{\sum_{j=1}^{M} s_{ij}}{2} \quad i = 1,\ldots,N, \tag{2}$$

$$\sum_{j=1}^{M} s_{ij}x_{jk} \leqslant 1 \quad i = 1,\ldots,N, \quad k = 1,\ldots,L, \tag{3}$$

$$\sum_{k=1}^{L} x_{jk} = 1 \quad j = 1,\ldots,M, \tag{4}$$

$$\sum_{j=1}^{M} x_{jk} \leqslant y_k \bullet N \quad k = 1,\ldots,L, \tag{5}$$

$$x_{jk} \leqslant b_{jk} \quad j = 1,\ldots,M, \quad k = 1,\ldots,L, \tag{6}$$

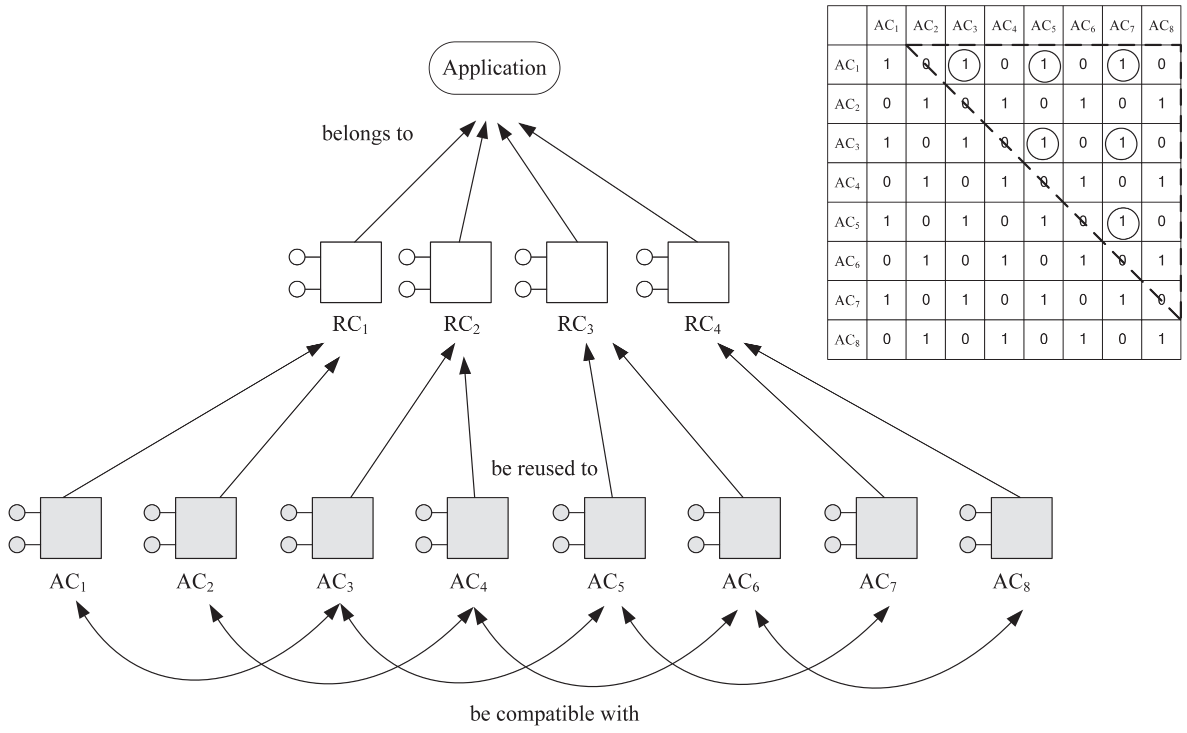
$$x_{jk}, y_k \in \{0,1\} \quad j = 1,\ldots,M, \quad k = 1,\ldots,L, \tag{7}$$

where, the first term of the objective function (1) represents the total procurement cost of all selected components, and the other one represents the total adaptation cost of all selected components. Hence objective (1) denotes minimizing the total cost for accomplishing all applications.

In Eq. (2), $\sum_{j=1}^{M} s_{ij}x_{jk}$ expresses whether the kth AC is selected to be deploy into the ith application. $\sum_{j=1}^{M} s_{ij}x_{jk} = 1$if and only if the kth AC is selected to implement the ith application, and $\left(\sum_{j=1}^{M} s_{ij}x_{jk}\right)\left(\sum_{j=1}^{M} s_{ij}x_{jk'}\right) = 1$ if and only if the kth AC and k'th AC belongs to the ith application simultaneously. So the formula $\sum_{k=1}^{L-1}\sum_{k'=k+1}^{L} r_{kk'}\left(\sum_{j=1}^{M} s_{ij}x_{jk}\right)\left(\sum_{j=1}^{M} s_{ij}x_{jk'}\right)$ denotes the sum of compatible binary parameters of all selected components in the ith application. To make sure that all of them are compatible with each other, it should be equal to $\binom{\sum_{j=1}^{M} s_{ij}}{2} = \frac{\left(\sum_{j=1}^{M} s_{ij}\right)!}{2!\left(\sum_{j=1}^{M} s_{ij}-2\right)!}$, where $\sum_{j=1}^{M} s_{ij}$ refers to the number of RCs belonging to the ith application, and $\binom{\sum_{j=1}^{M} s_{ij}}{2}$ represents the number of combinations of 2 elements from $\sum_{j=1}^{M} s_{ij}$ elements set. A simple Example (shown in Fig. 4) is introduced for explain Eq. (2). An application can be covered by four RCs (RC_1,RC_2,RC_3,RC_4), and they can be fulfilled by eight ACs. The relationship between RCs and ACs is shown in Fig. 4. There are two compatibility sets in ACs. The first compatibility set include AC_1, AC_3, AC_5 and AC_7, and the second compatibility set include AC_2, AC_4, AC_6 and AC_8. The compatibility matrix is shown on top right corner.

In the example, $N = 1$, $M = 4$ and $L = 8$, therefore, $\sum_{k=1}^{L-1}\sum_{k'=k+1}^{L} r_{kk'}\left(\sum_{j=1}^{M} s_{ij}x_{jk}\right)\left(\sum_{j=1}^{M} s_{ij}x_{jk'}\right)$ represents accumulation of elements of selected ACs in the compatibility matrix. If AC_1, AC_3, AC_5 and AC_7 are selected finally, it denotes an accumulation of all circled number and it equals 6. $\sum_{j=1}^{M} s_{ij} = 4$ in the example. Binomial coefficient $\binom{\sum_{j=1}^{M} s_{ij}}{2}$ refers to the number 2-combinations of an 4-element set, and $\binom{\sum_{j=1}^{M} s_{ij}}{2} = \binom{4}{2} = \frac{4!}{2!(4-2)!} = 6$.

Because the reusability of a component means that an AC can be reused into different applications rather than one application. Eq. (3) expresses each AC can be reused into an application system at most once. If $\sum_{j=1}^{M} s_{ij}x_{jk} = 0$, that means the kth AC is not selected into the ith application, otherwise, it means the kth AC is selected. Eq. (4) denotes that each RC can be implemented by adapting only



	AC_1	AC_2	AC_3	AC_4	AC_5	AC_6	AC_7	AC_8
AC_1	1	0	1	0	1	0	1	0
AC_2	0	1	0	1	0	1	0	1
AC_3	1	0	1	0	1	0	1	0
AC_4	0	1	0	1	0	1	0	1
AC_5	1	0	1	0	1	0	1	0
AC_6	0	1	0	1	0	1	0	1
AC_7	1	0	1	0	1	0	1	0
AC_8	0	1	0	1	0	1	0	1

Fig. 4. A simple example.

one AC into it. Eq. (5) is introduced to indicate the logic relation between the decision variables y_k and x_{jk}. It expresses that a CA can be available for implementing a RC into an application, and at most N times each for an application if the CA is selected, i.e. $\sum_{j=1}^{M} x_{jk} \leqslant N$ while $y_k = 1$, and $\forall j$, $x_{jk} = 0$ when $y_k = 0$. Eq. (6) denotes that the kth AC is selected to implement the jth RC if and only if $b_{jk} = 1$.

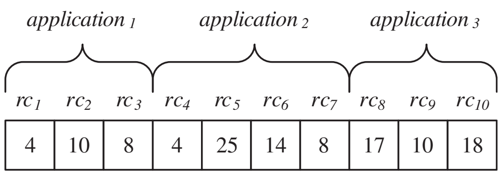
4. A customized genetic algorithm

Since the optimization model is a complex nonlinear binary integer programming, an important issue with this type of problem is to deal with combinatorial explosion effectively. As a well-known evolutionary algorithm, GA was employed to solve the optimization problem because of their good performance in combinatorial explosion [21–23]. Therefore, a customized GA (implemented using ECJ [24]) is proposed to solve large instances of the optimization problem.

4.1. Representation of chromosomes

According to attributes of the problem, a special encoding scheme of the proposed GA, called RC position-based representation, is introduced. With this encoding scheme, each gene of the chromosome gives an integer number that indicates the index of AC to fulfill a specified RC. It is explained by a simple case (shown in Fig. 5). As in the example, there are three applications undertaken concurrently, i.e. $N = 3$. For accomplishing those applications, 10 RCs need to be implemented, i.e. $M = 10$. The first application includes rc_1, rc_2 and rc_3, the second application comprises rc_4, rc_5, rc_6 and rc_7, the last application includes rc_8, rc_9 and rc_{10}. 30 ACs are provided in a component market, i.e. $L = 30$. Each integer in the vector indicates the index of selected ACs, that is to say, the 4th, 10th, 8th, 4th, 25th, 14th, 8th, 17th, 10th and 18th AC are selected to fulfill the corresponding RCs respectively. Because, in the customized GA, Eq. (1) in the proposed model is employed to be fitness function, the encoding scheme makes decoding to be computed quickly and easily. Given the index of selected ACs, each procurement cost and adaptation cost can be got easily (through getting relevant elements in c_k^p and c_{jk}^a). One thing need to be noticed that the procurement cost of each selected AC be accumulated only once. For example, in Fig. 5, AC_4 present twice, however, according objective function in the proposed model, the procurement cost of AC_4 can be accumulated once. In Fig. 5, the fitness value should be:

$$c_4^p + c_{10}^p + c_8^p + c_{25}^p + c_{14}^p + c_{17}^p + c_{18}^p + c_{1,4}^a + c_{2,10}^a + c_{3,8}^a + c_{4,4}^a + c_{5,25}^a + c_{6,14}^a + c_{7,8}^a + c_{8,17}^a + c_{9,10}^a + c_{10,18}^a.$$



rc_1	rc_2	rc_3	rc_4	rc_5	rc_6	rc_7	rc_8	rc_9	rc_{10}
4	10	8	4	25	14	8	17	10	18

Fig. 5. Representation of chromosomes.

4.2. Crossover and mutation

Similar to the classical GA, two crossover points are randomly generated, and swap their parents' genes between two crossover points. In contrast to the classical GA, a further judgment needs to be made after crossover. If a crossover point is located between two different applications, then nothing is done, otherwise, i.e. it is located inside an application. It needs to be verified further whether the same AC present twice in the application. If it does, one of them should be justified by using other AC to replace it.

The mutation can occur at any position of the chromosomes. Like crossover strategy, after a mutation operation, the mutated index should be verified further to make sure it is not adapted from the same AC with other RCs in the same application.

After the crossover and mutation operations, a validation test on the individual is performed to ensure that the combination of ACs satisfy conditions (2), (3), (4) and (6). If the solution is infeasible, it will be abandoned, and a new individual will be regenerated until it can pass the validation test.

Given the index of selected ACs, compatibility relationship among them can be checked using Eq. (2) and the compatibility matrix *R*.

Mutation rate and crossover rate of the GA were set as 0.01, and 0.9, respectively. While the size of a special problem increasing, population size of the proposed GA should be justified accordingly. After lots of experiments, an instructive advice is given here: Generally, if there are 5 applications, about 50 RCs and about 500 ACs, it is recommended to set population size to be between 100 and 200. Under this configuration, fitness values converge after around 200 generations.

5. Example & illustration

In this section, assume hypothetical small-scale case study is presented to illustrate how to use the proposed approach to assist software engineers in making decision. Suppose that a software developer undertakes two financial applications for two kinds of small-size companies, i.e., Garment Industry Financial System (App1) and Pharmaceutical Industry Financial System (App2). After performing requirement analysis and design, App1 and App2 are supposed to have three RCs, respectively, as shown in Table 1 i.e., Garment eCommerce Component (RC1), Garment Business Rule (RC2), Account for Garment Industry (RC3). App2 includes: Pharmaceutical eCommerce Component (RC4), Pharmaceutical Business Rule (RC5) and Account for Pharmaceutical Industry (RC6). There are 12 available components in related component markets, they are: eCommerce component 1 (AC1), available business rules component 1 (AC2), available account component 1 (AC3), eCommerce component 2 (AC4), business rule component 2 (AC5), account component 2 (AC6), eCommerce component 3 (AC7), business rule component 3 (AC8), account component 3 (AC9), eCommerce component 4 (AC10), business rule component 4 (AC11) and account component 4 (AC12). Though AC1, AC4, AC7 and AC10 are the eCommerce component and all of them can be reused to implement RC1 and RC4, they come from different component suppliers and have different procurement and adaptation costs (as shown in Table 1). The procurement cost of each AC is given in the second row, and adaptation cost of each AC to RC is shown in the matrix in Table 1, of which $c^a_{jk} = \infty$ denotes the *k*th AC cannot be reused to implement the *j*th RC. Compatibility Matrix of ACs is shown in Table 2.

After solving the model using optimization software ECJ [24], the solution is presented in the shaded cells in Table 1, from which one can find that AC10, AC11 and AC12 are selected and adapted to assemble into the two financial applications. The application App1 is assembled from RC1, RC2 and RC3 through adapting AC10, AC11 and AC12 respectively; while the application App2 is from RC4, RC5 and RC6 by adapting AC10, AC11 and AC12 respectively. That is to say, each ACs is reused two times to develop the financial applications, e.g. AC10 is selected to adapt into RC1 and RC4 for App1 and App2 respectively. The total cost of developing two applications is 230 dollars, of which procurement cost and adaptation cost are 138 and 82 respectively.

6. Management implications from experiments

6.1. Experiment design

To provide implication to software project management, four experiments derived from the model, are conducted in this section. They are:

Table 1
Procurement costs and adaptation costs of ACs (Dollar).

				AC1	AC2	AC3	AC4	AC5	AC6	AC7	AC8	AC9	AC10	AC11	AC12
				eCommerce component 1	Business rules component 1	Account component 1	eCommerce component 2	Business rule component 2	Account component 2	eCommerce component 3	Business rule component 3	Account component 3	eCommerce component 4	Business rule component 4	Account component 4
Procurement cost				48	66	54	55	62	61	74	74	79	47	42	49
App1	Garment Industry Financial System	RC1	Garment eCommerce Component	20	∞	∞	14	∞	∞	15	∞	∞	13	∞	∞
		RC2	Garment Business Rule	∞	20	∞	∞	19	∞	∞	16	∞	∞	17	∞
		RC3	Account for Garment Industry	∞	∞	14	∞	∞	14	∞	∞	16	∞	∞	18
App2	Pharmaceutical Industry Financial System	RC4	Pharmaceutical eCommerce Component	17	∞	∞	11	∞	∞	13	∞	∞	15	∞	∞
		RC5	Pharmaceutical Business Rule	∞	20	∞	∞	14	∞	∞	18	∞	∞	14	∞
		RC6	Account for Pharmaceutical Industry	∞	∞	11	∞	∞	16	∞	∞	20	∞	∞	15

Table 2
Compatibility matrix of ACs.

	AC1	AC2	AC3	AC4	AC5	AC6	AC7	AC8	AC9	AC10	AC11	AC12
AC1	0	1	1	0	0	0	0	0	0	0	0	0
AC2	1	0	1	0	0	0	0	0	0	0	0	0
AC3	1	1	0	0	0	0	0	0	0	0	0	0
AC4	0	0	0	0	1	1	0	0	0	0	0	0
AC5	0	0	0	1	0	1	0	0	0	0	0	0
AC6	0	0	0	1	1	0	0	0	0	0	0	0
AC7	0	0	0	0	0	0	0	1	1	0	0	0
AC8	0	0	0	0	0	0	1	0	1	0	0	0
AC9	0	0	0	0	0	0	1	1	0	0	0	0
AC10	0	0	0	0	0	0	0	0	0	0	1	1
AC11	0	0	0	0	0	0	0	0	0	1	0	1
AC12	0	0	0	0	0	0	0	0	0	1	1	0

Experiment 1. The effect of component reusability ($P(r)$) and number of ACs on average development cost.

Experiment 2. The influence of ratio of adaptation cost to procurement cost ($R(a/p)$) and number of ACs on average development cost.

Experiment 3. The effect of the number of compatibility set (CS) and ACs upon average development cost.

Experiment 4. The influence of the number of RCs and ACs upon average development cost.

In the third experiment, CS is derived from compatibility matrix. For example, there are four CS in Table 2, i.e., {AC1, AC2, AC3}, {AC4, AC5, AC6}, {AC7, AC8, AC9} and {AC10, AC11, AC12}.

Due to the fact that standard industrial data still not available from a wide range of CBSE community, a simulation approach adopted by Sundarraj [1,15] and Frakes [25], is employed here to test our model. Procurement costs and adaptation costs were randomly simulated using a uniform distribution (the notation $U(a,b)$ is introduced to represent uniform random numbers between a and b).

In each experiment, two parameters of the model are varying between different levels. To avoid suffering from the disadvantage of generating a minority of particular cases during simulation, at each level, the simulation is conducted five times, and the average value of five total costs is applied as management measure.

As for the problem instances of same size (the numbers of RCs, which are needed to accomplish all applications, are same among different problem instances), mathematically, the measure is given by:

$$C_{ave} = \sum_{p} C_p/p, \tag{8}$$

where the problem instances of each experiment are indexed by q, and C_p represent the best objective value of the pth problem instance. When problem instances have different the number of RCs, the measure is provided by:

$$C'_{ave} = \sum_{p} C_p/Lp, \tag{9}$$

where L represents the number of RCs.

6.2. Effect of reusability

Reusability means the capability of ACs to be reused into different applications, and higher reusability means better logic abstraction and implementation of a business domain. Generally, reusability is a significant criterion of assessing commercial components. The probability of reuse, which denotes the likelihood that an AC can be reused (adapted) into an application, is employed to represent the reusability, and denoted by $P(r)$.

In this experiment, $P(r)$ of an AC varies from 10% to 90% in steps of 10%, and the number of ACs in market increases from 200 to 1000 in steps of 100. The other parameters, namely, the number of application is constant at 5, the number of RCs and CSs is set at 50 and 3, respectively. Procurement costs and adaptation cost were drawn from the distribution $U(40,80)$ and $U(10,20)$. The simulation results of the experiment are visualized as shown in Figs. 6 and 7.

Fig. 7 present a mathematical contour plot of C_{ave} varying with reusability and number of ACs. In Fig. 6, when the number of ACs is set to be 200, the line of C_{ave} decreases drastically with improvement of component reusability. The decrease tends to smooth and steady while the number of ACs increasing. When 1000 ACs are provided, C_{ave} only decrease from 67.74 to 65.34. The tendency also can be observed in Fig. 7. At the bottom of contour map, colors vary considerably. However, only two colors full most of zone at the top of map.

Through analyzing Figs. 6 and 7, conclusions can be drawn as: If the market does not provide plentiful ACs, reusability of ACs has strong impact on C_{ave} of all applications. On the contrary, the influence of component reusability on C_{ave} is limited.

6.3. Effect of ratio of adaptation and procurement cost R(a/p)

This experiment was performed to observe relationship between C_{ave} and $R(a/p)$, which represents the effort to adapt ACs into

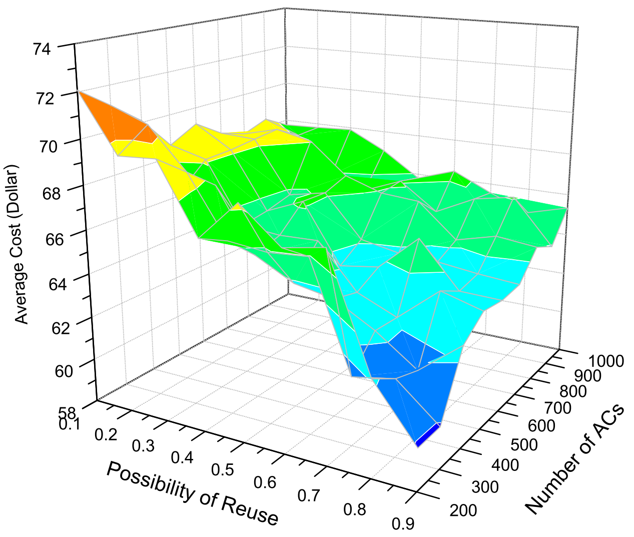


Fig. 6. Effect of reusability and number of ACs.

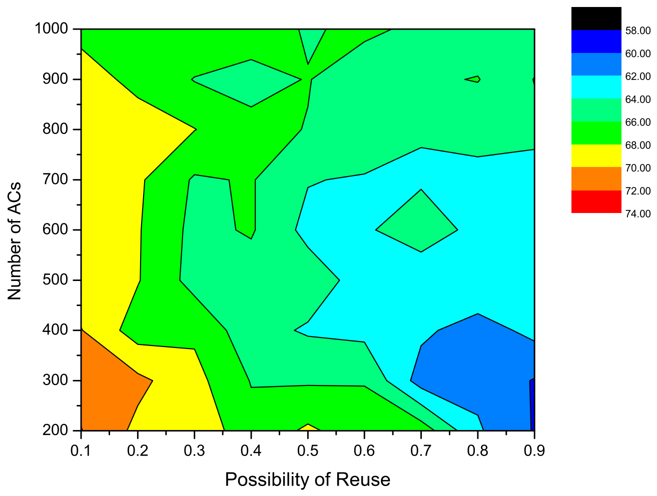


Fig. 7. Contour of C_{ave} while $P(r)$ and the number of ACs changing.

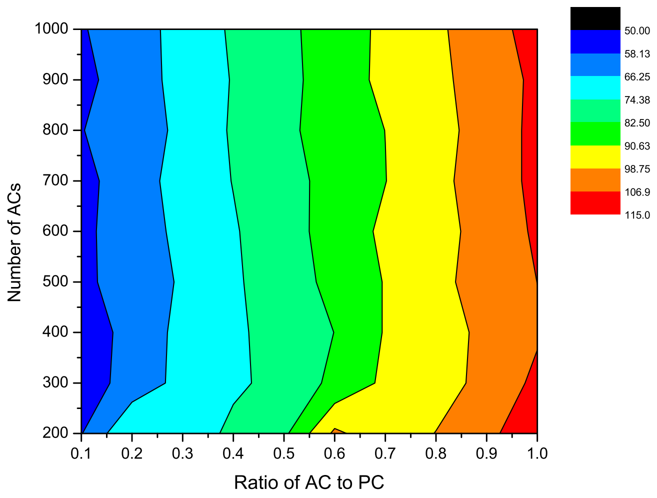


Fig. 9. Contour of C_{ave} while $R(a/p)$ and the number of ACs varying.

applications. In this experiment, $R(a/p)$ varies from 0.1 to 1.0, while keeping procurement cost generating from the distribution $U(40,80)$. The number of ACs changes from 200 to 1000 in steps of 100. $P(r)$ is constant at 50%, and the number of CS is set at 3. The numbers of application and of RCs are given as 5 and 50 respectively. The effect of $R(a/p)$ and number of ACs on the total cost is visualized as in Figs. 8 and 9, from which one can observe the total cost increases steadily with $R(a/p)$, and color zones in Fig. 9 are almost parallel. It means that $R(a/p)$ of ACs has more substantial effect upon C_{ave} than the volume of ACs in a market.

6.4. Effect of the number of CSs

The number of CSs stands for the diversity of compatibility sets in a component market. In this experiment, problem instances vary in the number of CSs from 1 to 9, and the number of ACs changes from 200 to 1000 in steps of 100. Reuse probability is set to be 50%. The distribution of procurement costs and adaptation costs are at $U(40,80)$ and $U(10,20)$, respectively. The number of applications and RCs are set at 5 and 50, respectively. The effect of the number of CSs on the total cost is plotted in Figs. 10 and 11.

In Fig. 10, the C_{ave} surface raises up radically while the number of CSs is larger than 5, and the number of ACs is smaller than 600. It implies that incompatible achievement solutions can increase cost when the number of ACs is limited. In Fig. 11, the colorful zone

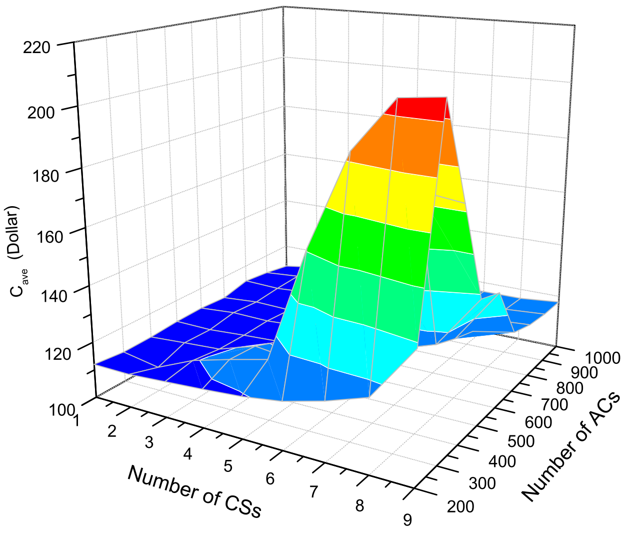


Fig. 10. Effect of the number of CSs and of ACs.

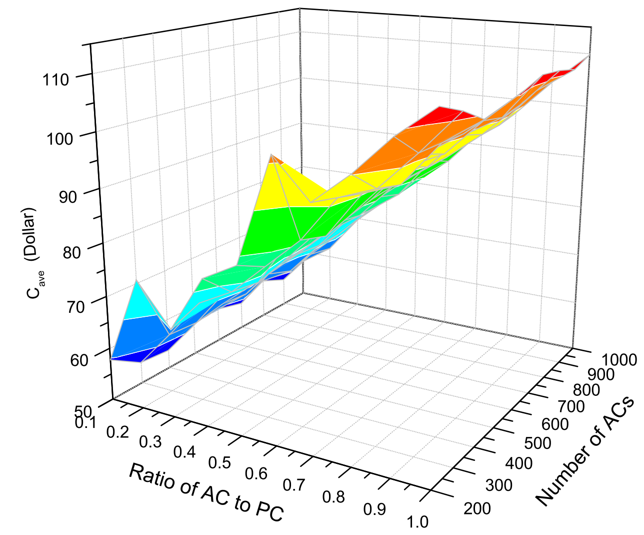


Fig. 8. Effect of adaptability and number of ACs.

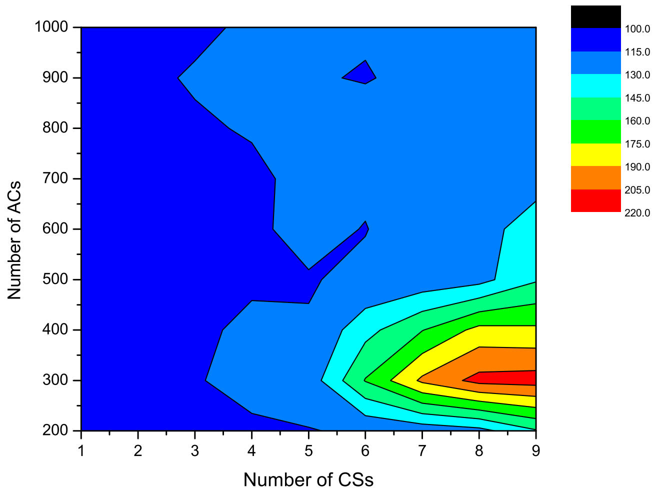


Fig. 11. Contour of C_{ave} while the number of CSs and ACs changing.

shrinking into bottom right corner of contour map is definitely a "dangerous region". It is very hard for software developers to reduce cost rapidly in that area. If a component market falls into other regions except the "dangerous zone", the total cost can be reduced drastically by software developers through using effective component selection methods.

6.5. Effect of the number of RCs

When the number of undertaken applications is constant, the number of RCs increasing represents the size of applications grows accordingly. To observe impact of the size of applications on C'_{ave}, in this experiment, the number of RCs changes from 20 to 80 in steps of 5, and the number of ACs varies from 200 to 1000 in steps of 100. The reusability probability is kept at 50%, and the number of CS is set at 3. Procurement costs and adaptation costs are generated randomly from the distribution $U(40,80)$ and $U(10,20)$, respectively. The number of application is set as 5. The simulation results are shown in Figs. 12 and 13.

Similar to the experiment as described in Section 4.3, there also exist peaks (where the number of RCs is larger than 35 and the number of ACs is more than 700) as shown in Fig. 13, It implies that cost will increase when there are too many RCs need to be implemented and a few ACs are provided in market. It implies that it would be a very smart choice for software developers to analyze whether component market resides in this zone or not. By the way, another phenomenon can be observed in Fig. 13 is that, although regions out of the "dangerous region" are smooth and flat, the region in bottom left of contour map are a little better than the one in up right. That means undertaking applications of smaller size are more slightly effective in reducing cost than selecting components in a more abundant component market.

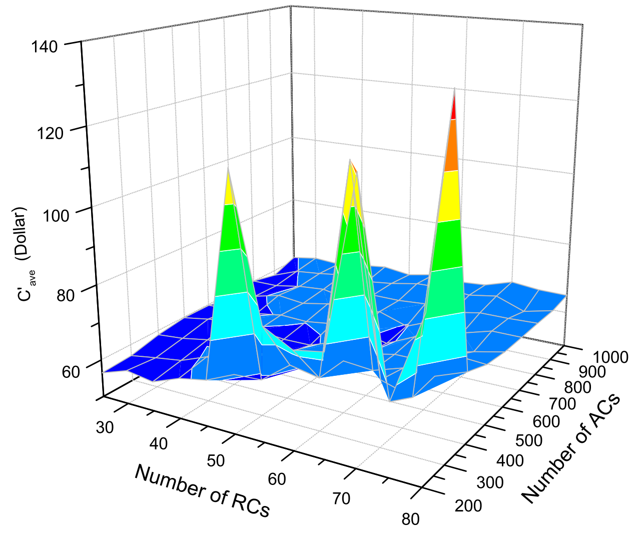


Fig. 12. Effect of the numbers of RCs and of ACs.

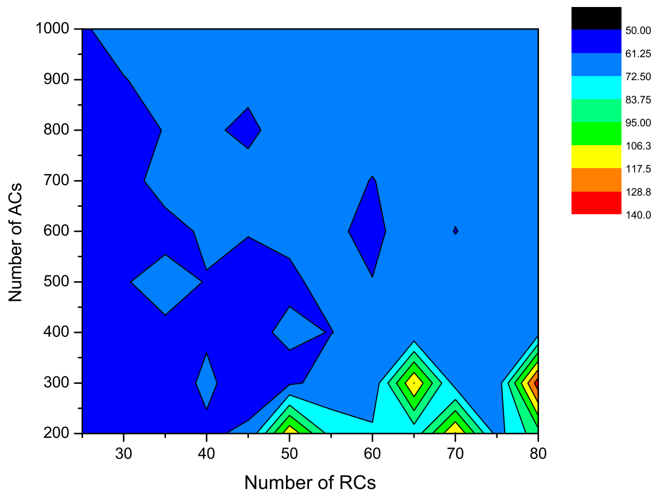


Fig. 13. Contour of C'_{ave} while the number of ACs and RCs varying.

7. Discussion and conclusion

A practical software engineering problem, i.e., a software developer undertakes multiple development tasks of enterprise application concurrently, is discussed in the paper. An optimization model is proposed to assist software developers in selecting components. Compared with the previous studies, the proposed model as described in this paper considers components' compatibility and cost simultaneously.

Several management implications are provided through experiments and sensitive analysis. In the first experiment, a simulation case is used to illustrate and discuss how to use the proposed model to estimate the different effect of component reusability and the number of ACs upon average development cost. In can be concluded that average cost decrease while $P(r)$ is increasing and the number of ACs is decreasing simultaneously, vice versa, cost will increase. In the second experiment, the influence upon average cost come from ratio of adaptation cost to procurement cost and the number of ACs is discussed. $R(a/p)$ of ACs has more substantial effect upon than the volume of ACs in a market. In the third case, how to estimate average cost while the number of compatibility set and ACs changing is explained. In the last experiment, how to analyze variance in average cost while the number of RCs and ACs vary is discussed.

Software product development process is a complex one involving many factors. Functional or non-functional requirements, e.g. development times cycle, quality, reliability, etc. should be considered simultaneously, as objectives or constraints, which will be addressed in the near future as an extension of this paper. In addition, the customized GA could not guarantee the optimal solution.

Acknowledgements

This research is financially supported by Natural Science Foundation of China (NSFC 71021061), the National Key Basic Research Project of China (2009CB320601), and 111 project of Ministry of Education (MOE) in China with number B08015, the Hong Kong Polytechnic University, China (Project No. G-YG40).

References

[1] R.P. Sundarraj, An optimization approach to plan for reusable software components, European Journal of Operational Research 142 (1) (2002) 128–137.
[2] P. Herzum, O. Sims, Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise, John Wiley & Sons, Inc., New York, NY, USA, 2000.
[3] W. Kozaczynski, G. Booch, Component-based software engineering, IEEE software 15 (5) (1998) 34–36.
[4] P. Vitharana, H. Jain, Design, retrieval, and assembly in component-based software development, Communications of the ACM 46 (11) (2003) 97–102.
[5] C. Szyperski, D. Gruntz, S. Murer, Component Software Beyond Object-Oriented Programming, Addison-Wersley, 2002.
[6] A.W. Brown, Large-Scale, Component Based Development, Prentice Hall, PTR Upper Saddle River, NJ, USA, 2000.
[7] F.P. Brooks, No silver bullet: Essence and accidents of software engineering, IEEE Computer 20 (4) (1987) 10–19.
[8] A.W. Brown, K.C. Wallnau, Component-based software engineering, IEEE Computer Society, 1999. pp. 714–715.

[9] D.G. Messerschmitt, C. Szyperski, Industrial and economic properties of software: technology, processes, and value, University of California at Berkeley Computer Science Division Technical Report, 2001. 18: pp. 2001–11.
[10] B.Z. Abraham, J.C. Aguilar, Software Component Selection Algorithm Using Intelligent Agents, in: Agent and Multi-Agent Systems: Technologies and Applications, 2007, pp. 82–91.
[11] N. Haghpanah, S. Moaven, J. Habibi, M. Kargar, S.H. Yeganeh, Approximation algorithms for software component selection problem, in: Software Engineering Conference, 2007, APSEC 2007, 14th Asia-Pacific. 2007, Aichi, pp. 159–166.
[12] J. Bhuta, C.A. Mattmann, N. Medvidovic, B. Boehm, A framework for the assessment and selection of software components and connectors in cots-based architectures, IEEE Computer Society, 2007. p. 6.
[13] C. Geisterfer, S. Ghosh, Software component specification: a study in perspective of component selection and reuse, in: Proceedings of the Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems. 2006, IEEE Computer Society, Washington, DC, USA, 2006, pp. 100–108.
[14] P. Ardimento, A. Bianchi, G. Visaggio, Maintenance-oriented selection of software components, in: Proceedings of the Eighth European Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA, 2004, p. 115.
[15] R.P. Sundarraj, S. Talluri, A multi-period optimization model for the procurement of component-based enterprise information technologies, European Journal of Operational Research 146 (2) (2003) 339–351.
[16] H.W. Jung, B. Choi, Optimization models for quality and cost of modular software systems, European Journal of Operational Research 112 (3) (1999) 613–619.
[17] E. Armstrong, J. Ball, S. Bodoff, D.B. Carson, I. Evans, D. Green, K. Haase, E. Jendrock, The J2EE 1.4 tutorial. Sun Microsystems, 2005.
[18] S.H. Kan, Metrics and Models in Software Quality Engineering, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
[19] J.A. McCall, P.K. Richards, G.F. Walters, Factors in software quality, The National Technical Information Service, Springfield, VA, USA, 1977.
[20] D.C. Craig, W.M. Zuberek, Verification of Component Behavioral Compatibility, in Dependability of Computer Systems, 2007. DepCoS-RELCOMEX '07, in: second International Conference on 2007, pp. 294–304.
[21] W.T. Chan, C.Y. Tan, Genetic-algorithm programming of road maintenance and rehabilitation, Journal of Transportation Engineering 122 (1996) 246.
[22] R.M. Marian, L. Luong, K. Abhary, A genetic algorithm for the optimisation of assembly sequences, Computers & Industrial Engineering 50 (4) (2006) 503–527.
[23] Y. Tsujimura, M. Gen, Entropy-based genetic algorithm for solving TSP (1998).
[24] S. Luke, L. Panait, G. Balan, S. Paus, Z. Skolicki, J. Bassett, R. Hubley, A. Chircop, ECJ: A Java-based evolutionary computation research system. Downloadable versions and documentation can be found at the following url: <http://cs.gmu.edu/eclab/projects/ecj>, 2006.
[25] W. Frakes, C. Terry, Software reuse: metrics and models, ACM Computing Surveys (CSUR) 28 (2) (1996) 415–435.