Hindawi
Wireless Communications and Mobile Computing
Volume 2017, Article ID 4020162, 10 pages
https://doi.org/10.1155/2017/4020162

Research Article

Greening Software Requirements Change Management Strategy Based on Nash Equilibrium

Zhixiang Tong,[1] Xiaohong Su,[1] Longzhu Cen,[2] and Tiantian Wang[1]

[1] *School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China*
[2] *School of Science, Harbin Institute of Technology, Harbin 150001, China*

Correspondence should be addressed to Xiaohong Su; sxh@hit.edu.cn

Received 13 July 2017; Revised 4 September 2017; Accepted 19 September 2017; Published 10 December 2017

Academic Editor: Zhi Liu

Recently, green computing has become more and more important in software engineering (SE), which can be achieved by effectively recycling the software system and utilizing the computing resources. However, the requirement change may lead to unnecessary labor and time cost. Moreover, it may also result in the waste of hardware and computing resources once unreasonable requirements are realized. Thus, to perform green computing in SE, it is necessary to propose effective strategies to manage the requirement change. For this decision-making problem, game theoretical methods can be feasible solutions. In this paper, we propose a novel requirement change management approach based on game theory. Specifically, we model the problem as a game between the stakeholders and the developer and devise the payoff matrix between different strategies of the players. We then propose a Nash equilibrium-based game theoretical algorithm to manage requirement change. The evaluation results show that, compared to the exhaustive algorithm, our method not only can achieve almost the same optimal results but also can significantly reduce the computational time complexity. Thus, our method is feasible for a lot of requirement changes and can facilitate the green computing targets from the perspective of software engineering.

1. Introduction

Software is the fundamental part of the computing system. Greenness in the software is an emerging quality attribute that needs to be taken into account in each phase of the software development process involving requirement engineering (RE), the system implementation, testing, and maintenance. To support efficient computing paradigms, it is necessary to design effective software requirements management methods. Requirement engineering is one of the most important phases of the software development process and is able to reduce software errors at its early stage. "Variability" is an important feature in the software requirements, and a lot of software system development issues arise from requirement changes. Existing researches indicate that before the software products are formally put into use over 50% of the software requirements will be changed [1]. Requirement change results from adding, deleting, or updating requirements will lead to a series of serious problems in techniques, quality, management, and so forth, which may cause software failure. Thus, how to respond to requirement changes is essential for the survival of the software products. When the software requirement changes, if the project proceeds without judgments or evaluation on the changes, it may disrupt the control and management. In the software developing process, it is necessary to formulate feasible solutions to evaluate the impacts of the requirement change from various aspects such as cost, developing period, quality, and user satisfaction, aiming to provide evidence on making a proper decision. These solutions are essential to ensure the overall success of the software development project.

In the past decades, a number of models have been proposed to describe and analyze the requirement change management [1–9]. On one hand, some studies focus on formal descriptions of requirement changes, such as Kobayashi and Maekawa [1] who came up with an NBM (Need-Based requirement change Management) model, which describes system requirements in 4W (Where, Who, Why, and What).

This model monitors and processes requirement changes during the entire software development cycle by the way of validation and verification. Ramzan and Ikram describe the software requirement change management process as a process model, which consists of activities, roles, and artifacts [2]. Ahmad et al. present a model to minimize the impact of requirement change through classifying the requirements into three different categories: fixed, less likely to change, and most likely to change [9]. On the other hand, some studies have proposed specific methods and procedures for the management of changes. Bhatti et al. propose a change management approach that includes six basic phases [3]. According to this approach, Change Control Board (CCB) evaluates the effects of requirement changes in various aspects of the system; then CCB, QA (Quality Assurance) team, the requirement change proposer, and project group members should reach a collective agreement on whether to accept or reject this change. The Spiral of Change model is a change management method that includes change proposing, decision-making, and technology implementation, and whether to implement the change is determined by the change owner [4]. Recently, Tomyim and Pohthong propose a requirement change management model especially for object-oriented software engineering using unified modeling language [10]. Oertel and Rettberg present a change management approach that measures the effects of a change by integrating the verification and validation activities and the propagation of changes [11]. However, most of the existing models lack the impact analysis and decision-making process for requirement changes. The impact analysis means determining the impact of requirement changes on the software developing processes, such as the impact on the cost or benefit for the developer and the satisfaction for the stakeholders, which will help to provide decision-making on implementing a change scientifically, reasonably, and accurately. In addition, most of the previous methodologies depend a lot on the human role which relies on the interest and experience of the decision-makers, and they also lack detailed descriptions of the decision methods. These problems will affect the reproducibility of the results of a change. Some researchers emphasize that using the agent-oriented process to improve software productivity may be a novel model in requirement change management (RCM) [12–14]. Bendakir et al. develop an agent-oriented approach based on the definition of the process covering all the necessary activities for an effective RCM process [15]. The adaptability, perception, and cooperation features of the model allow managing the change requirements in a controlled manner. In summary, the requirement change impact measurement, the change management automation, and the adjustment strategy for the system after accepting the changes are still crucial problems in RCM [16–18].

Although a set of requirement change management mechanisms have been proposed, in this study, we solve the RCM problem from a different point of view. The core issue of the RCM is to decide whether to accept or reject a change to the system. In game theory, this is a constraint equilibrium problem, that is, to find Nash equilibrium point in a game. Game theoretical methods have been successfully applied in various fields such as optimizing the spectrum allocation among different mobile users [19], and modeling the socially aware utility maximization problem, and deriving the distributed decision [20]. Inspired by these works, in this paper, we propose a novel requirement change decision-making method based on game theory. This method transforms the requirement change decision problem into the mutual game between the software system developer and the stakeholders that put forward the change requests. Through effective RCM, the development of unnecessary requirements can be prevented and green computing is achieved by using the computing resources efficiently. In addition, through implementing effective requirement changes, the existing software system can be recycled, which plays an important role in green computing.

Table 1: The payoff matrix for stakeholders' requirement change requests and developer's responses.

Stakeholders	Developer	
	Overall adjustment	Local adjustment
Propose	$k + i - f, K + I + F - C - L$	$k, K - L$
Not propose	$i, I - C$	0, 0

2. Game Theoretical Model for the Requirement Change

2.1. Model Components and the Payoff Matrix. The components of our proposed game theoretical model include the following:

(1) Participators: the stakeholders and the developer. Stakeholders are persons who care about the software system and may come up with requirements for the system. The developer is the executor of the software system development and is responsible for the benefit and cost consideration of the system

(2) Strategy: it refers to the decision variable that the participant is playing at some point in the gaming (action plan). In the game, stakeholders' strategy space is A_1 = {propose requirement change, not propose requirement change}, while the developer's strategy space is A_2 = {overall adjustment, local adjustment}

The payoff matrix of the requirement change is shown in Table 1. The benefit and cost for stakeholders directly correspond to the increase or decrease of their satisfactions. In the game process, CCB will first announce two strategies of the developer to stakeholders: local adjustment and overall adjustment. The local adjustment policy will directly implement all change requests made by the stakeholders. The overall adjustment is to release an existing adjustment scheme given by the developer. Upon knowing the strategies of the developer, stakeholders may decide whether to propose their requirement change request or not.

When stakeholders choose not to propose the request, the developer has two options: (1) if the developer chooses local adjustment strategy, the system will stay the same, and the benefits for both stakeholders and the developer are zero; (2) if the developer chooses to make an overall adjustment, the system will be adjusted at the cost of C according to the scheme announced before. Meanwhile, the stakeholders will gain a satisfaction of i and the developer will receive a benefit of I. The corresponding element in the payoff matrix is $(i,\ I - C)$. Because the original system development plan is determined after optimizing the requirements, generally $I < C$.

When stakeholders choose to propose the request, the local adjustment of the developer will cost L to implement the requirement, which increases the satisfaction of stakeholders by the amount of k. The developer will get a benefit of K accordingly. So the corresponding element in the payoff matrix is $(k,\ K - L)$. If the developer chooses to make an overall adjustment, this can be divided into two situations:

(1) In one case, if the overall adjustment scheme already contains the stakeholders' changing requirements, then the existing adjustment scheme does not need to be changed, which makes the element in the payoff matrix $(i,\ I - C)$.

(2) In the other case, the overall adjustment scheme does not contain the stakeholders' changing requirements. Then on the basis of the existing adjustment scheme, it will cost L for the developer to make the stakeholders have a satisfaction improvement of i by implementing their requirements. At the same time, CCB will announce that some of their original requirements will be canceled, serving as the cost for the modification of existing adjustment scheme. So a satisfaction decrease of f will be informed to the stakeholders if they insist on the requirement change request. Therefore, the developer will get a benefit of F by recycling that part of the developing resources. As a result, the corresponding element in the payoff matrix is $(k + i - f,\ K + I + F - C - L)$.

Because the proposed change requests by the stakeholders will be accepted in both cases, the payoff matrix will contain k, K, and L for each case. Based on this, we change the payoff matrix of the first case as $(k + i' - f',\ K + I' + F' - C' - L)$, in which $i' = i - k,\ I' = I - K,\ f' = F' = 0$. In order to facilitate the subsequent analysis, we have unified the expression of the payoff matrix in both cases as $(k + i - f,\ K + I + F - C - L)$; the values of i, f, I, F, and C change in different situations.

2.2. Game Theory Analysis

2.2.1. Situation A. When $I + F < C$, this case indicates that the overall adjustment cannot recycle too much originally planned resources for the stakeholders, so that F is small. At this point, the local adjustment is the strictly dominant strategy for stakeholder and CCB will announce that the overall adjustment is abolished, which induces stakeholders to propose requirement change. Therefore, the Nash equilibrium point is (local adjustment, propose).

2.2.2. Situation B. When $k > f$, this case indicates that gained satisfaction is larger than the cost if the stakeholder chooses to propose when the overall adjustment was to be made. At this point, the stakeholders have a strictly dominant strategy, that is, proposing the requirement change. The strategy of the developer depends on the comparison results between $I + F$ and C: if $I + F < C$, the local adjustment is chosen; otherwise, overall adjustment is chosen. So the Nash equilibrium is an alternative in (local adjustment, propose) and (overall adjustment, propose).

2.2.3. Situation C. When $I + F > C$ and $k < f$, in this case, the game does not have pure strategy Nash equilibrium. Thus, when the developer is inclined to make the overall adjustment, the stakeholders can get the largest returns without making any requests. When the stakeholder has no more requests, the returns can reach a larger point with the developer making a local adjustment. At this point, the game leads to the mixed strategy equilibrium. A mixed strategy is an assignment of a probability to each pure strategy. Since probabilities are continuous, there are infinite mixed strategies available to a player. And the mixed strategy equilibrium is a phenomenon where the participants stay in particular mixed strategies. The existence of mixed strategy equilibrium provides a basis for how to manage the change of requirement. The procedure of resolving the equilibrium point is shown as follows.

Suppose that the probability that the stakeholders choose to change the requirements is x and the probability of the developer taking an overall adjustment is y. The profit expectations of the stakeholders and the developer $U(x, y)$ and $V(x, y)$ are expressed as formula (1) and formula (2), respectively:

$$U(x, y) = x\left[y\left(k + i - f\right) + \left(1 - y\right)k\right] + (1 - x)\,yi \tag{1}$$

$$\begin{aligned} V(x, y) = {} & y\left[x\left(K + I + F - C - L\right) + (1 - x)\left(I - C\right)\right] \\ & + \left(1 - y\right)x\left(K - L\right). \end{aligned} \tag{2}$$

Upon achieving the mixed strategy equilibrium, none of the participators can increase profits by changing the probability distribution of strategy choice. This can be expressed in equations as

$$\begin{aligned} \partial_x U(x, y) &= 0 \\ \partial_y V(x, y) &= 0. \end{aligned} \tag{3}$$

Resolve (3) resulting in $x^* = (C - I)/F$ and $y^* = k/f$. x^* and y^* actually clarify the mixed strategies adopted by stakeholders and the developer when the mixed strategy equilibrium is reached. We shall see this by the following analysis.

When stakeholders select to propose or not by the probability of x^* and $1 - x^*$, respectively, the profit expectations

of the developer to make overall or local adjustment are expressed by formula (4) and formula (5), respectively:

$$V_O = \frac{C-I}{F}(K+I+F-C-L) + \left(1-\frac{C-I}{F}\right)(I-C) = \frac{C-I}{F}(K-L) \tag{4}$$

$$V_L = \frac{C-I}{F}(K-L). \tag{5}$$

Thus, we have $V_O = V_L$.

Similarly, when the developer conducts overall adjustment and local adjustment by the probability of y^* and $1-y^*$, respectively, the return expectations of stakeholders to propose or not are calculated according to formula (6) and formula (7), respectively:

$$U_P = \frac{k}{f}(k+i-f) + \left(1-\frac{k}{f}\right)k = \frac{ki}{f} \tag{6}$$

$$U_{\mathrm{NP}} = \frac{ki}{f} \tag{7}$$

Thus, we have $U_P = U_{\mathrm{NP}}$.

The above analysis shows that the expected payoff of both the stakeholders and the developer is irrelevant to the probability distribution of random choice; that is, neither of the two can improve their return expectation in the way of changing the probability distribution. As a result, in the condition of $I+F>C$ and $k<f$, the mixed strategy equilibrium point in the game is that the stakeholders choose the strategy of proposing or not proposing by a probability of $(C-I)/F$ and $1-(C-I)/F$, respectively, and the developer chooses the strategy of overall or local adjustment with a probability of k/f and $1-k/f$, respectively.

To see the Nash equilibrium from the perspective of the developer, when the probability that the stakeholders tend to propose requirement changes is relatively large (larger than $(C-I)/F$), it means that the system has obvious irrationality and should make the overall adjustment. On the contrary, when the probability is relatively small (smaller than $(C-I)/F$), it shows that the stakeholders are satisfied with the system so that only the local adjustment is required according to specific customers' requests. The actual meaning of Nash equilibrium for the stakeholders is that there is no need to ask for a requirement change in order to obtain a larger benefit when the developer makes the overall adjustment with a larger probability ($>k/f$). On the contrary, changes should be initiated to urge the developer to make an improvement.

The analysis of the game process can provide guidance for the strategy selection upon requirement change requests. But there are still difficulties in how to perform the overall adjustment. The overall adjustment is established based on the recycling and redistribution of the development resources. Thus, it is necessary to apply the game theory to establish an adjustment scheme that could evolve dynamically coping with these changes. Besides, the dynamically performing process can be described with the idea of game theory. In the following section, we will put forward a method to formulate the overall adjustment plan according to this idea.

3. The Method of Strategy-Making for Overall Adjustment Plan

In the process of determining the overall adjustment scenario, there always exists an irreconcilable conflict between controlling the development cost and improving the stakeholders' experience. Based on this situation, how to find an overall adjustment program that can balance the two becomes a key issue to consider for the developers. Based on this situation, how to find an overall adjustment scheme that can balance the two becomes a key issue for the developers. In this section, we quantify the stakeholders' experience in the form of satisfaction. And a mathematical model is built to look for the optimal overall adjustment strategies and analyze the game mechanism in the process of overall adjustment under this model. Finally, under the guidance of Nash equilibrium principle, an efficient method of determining the overall adjustment scheme has been proposed.

3.1. The Analysis of Game Mechanism in the Process of Making an Overall Adjustment Program. The mathematical model we have chosen is based on the following basic assumptions:

(1) The number of stakeholders (m) and the requirements for each stakeholder are provided. Each stakeholder has distinct requirements through pruning the redundancy. The total number of requirements is given as n.

(2) The implementation of each requirement is set to bring a satisfaction c to the stakeholder. The value range of c is $(0, 100]$. And the satisfaction score for a stakeholder will reach 100, after all the requirements of the stakeholder are implemented.

(3) The development cost for each requirement is evaluated using a variable t. A bigger value of t represents a larger development cost for the requirements.

Based on the above assumptions, all stakeholders' requirements can be represented as a set $D = \{d(c,t) \mid 0 < c \le 100,\ t > 0\}$. The set D contains n elements, each representing a stakeholder requirement. The overall adjustment options available to developers can be represented using the set $S = \{s(b_1, b_2, \ldots, b_n) \mid \forall b_i = 0 \text{ or } 1\}$. In the set, each variable b_i refers to one stakeholder requirement, and $b_i = 0$ denotes that the requirement will not be satisfied, while $b_i = 1$ denotes that the requirement will be satisfied. Therefore, the total number of solutions available for the developers is 2^n. In order to find the optimal solution, an evaluation indicator P is given to evaluate each solution. For any adjustment strategy $s \in S$, the evaluation result is given by the following formula:

$$P(s) = A\frac{1}{100m}\sum_{i=1}^{n} b_i c_i + B\frac{1}{2500}\sigma^2 + C\frac{1}{T}\sum_{i=1}^{n} b_i t_i. \tag{8}$$

In the formula, $\sum_{i=1}^{n} b_i c_i$ represents the sum of the satisfaction that is gained by using strategy s to develop the software, with a maximum value of $100m$. σ^2 denotes the variance of all stakeholders' satisfaction resulting from strategy s. The satisfaction values for n stakeholders are expressed by

n random variables $X_1, X_2, X_3, \ldots, X_n$ with an average of Y; the variance σ^2 is represented as follows:

$$\begin{aligned}\sigma^2 &= \frac{\left[(X_1 - Y)^2 + (X_2 - Y)^2 + \cdots + (X_n - Y)^2\right]}{n} \\ &= \frac{\left[X_1^2 + X_2^2 + \cdots + X_n^2 - 2Y(X_1 + X_2 + \cdots + X_n) + nY^2\right]}{n} \\ &= \frac{\left[X_1^2 + X_2^2 + \cdots + X_n^2 - 2nY^2 + nY^2\right]}{n} \\ &= \frac{\left[X_1^2 + X_2^2 + \cdots + X_n^2 - nY^2\right]}{n} \\ &= \frac{\left[n\left(X_1^2 + X_2^2 + \cdots + X_n^2\right) - (X_1 + X_2 + \cdots + X_n)^2\right]}{n^2}.\end{aligned} \tag{9}$$

In the above formula, any X_n should be an upward quadratic function. So, when $X_n = 0$ or 100 (i.e., the two endpoints of the domain), σ^2 reaches the maximum value.

Suppose that there are a random variables with a value of 0 and $n - a$ random variables with a value of 100; then the average can be calculated as $Y = 100(n - a)/n$, so σ^2 is calculated as follows:

$$\begin{aligned}\sigma^2 &= \frac{\left[a(0 - Y)^2 + (n - a)(100 - Y)^2\right]}{n} \\ &= \frac{\left[a(0 - 100(n - a)/n)^2 + (n - a)(100 - 100(n - a)/n)^2\right]}{n} \\ &= \frac{10000\left[a(n - a)^2 + (n - a)a^2\right]}{n^3} = \frac{10000a(n - a)}{n^2} \\ &= 10000\frac{\left[-(a - n/2)^2 + n^2/4\right]}{n^2} \leq 2500.\end{aligned} \tag{10}$$

When $a = n/2$, that is, there are $n/2$ variables with a value of 0 and $n/2$ variables with a value of 100, then the variance will have a maximum value of 2500 (if n is an odd number, then the variance is not going to reach 2500).

Here $\sum_{i=1}^{n} b_i t_i$ represents the development cost required for developing the software using a strategy s, with a maximum value as $T = \sum_{i=1}^{n} t_i$.

A, B, and C are the sum weights, where $A > 0$, $B < 0$, and $C < 0$ and their specific values are dependent on the actual needs of the developers.

Based on the above mathematical model, the determination process of the overall software adjustment strategy can be summarized as follows: for a given set of requirements D, seek to find a strategy $s^* \in S$ which will lead to a maximum evaluation index P. As S contains 2^n elements, the algorithm complexity is at least $O(2^n)$ when applying the exhaustive search method to find s^*. Thus the exhaustive method is not an efficient and practical algorithm. The reason for its low efficiency is that it ignores some of the inherent rules which exist in finding the optimal adjustment strategy. Using the analysis method of the game theory, we can find the game mechanism to obtain the optimal adjustment strategy based on the above model. The existence of such a game mechanism will help to develop a more efficient search method.

In the developer's point of view, there is a competitive and cooperative relationship among all alternative development requirements, competing for the limited development resources and cooperating to improve the stakeholders' experience. It makes that the alternative development requirements have the attributes of game players, so we regard each requirement as a game participant. For a participant i, there are two choices: whether it is developed or not, corresponding to the two cases in (8), where b_i is equal to 1 or 0. The benefits of the two strategies are given by the following formula:

$$p_i(s) = A\frac{1}{100m}b_i c_i + B\frac{1}{2500}\sigma^2 + C\frac{1}{T}\sum_{j=1}^{n} b_j t_j. \tag{11}$$

The meanings of the variables in formula (11) are the same as those of formula (8). Based on the above definitions, it is not difficult to find that the state of the game (the choice of each game player's strategy) has a one-to-one relationship with the overall adjustment scheme. Thus, changing the overall adjustment program is equivalent to changing the game state. In accordance with the game theory, considering that each game player is seeking to maximize their own interests, the game will eventually reach the Nash equilibrium. Based on a simple analysis, we find that looking for a strategy $s^* \in S$ to reach the maximum value of P is equal to finding the Nash equilibrium in a game. The proof is as follows.

Suppose that there is a non-Nash equilibrium solution that maximizes $P(s)$; then there is at least one requirement x, let us say the kth, which is willing to improve the benefit by changing its strategy. $P(s)$ could be represented as

$$P(s) = A\frac{1}{100m}\sum_{i=1, i\neq k}^{n} b_i c_i + p_k(s). \tag{12}$$

Consider that only the requirement x has changed its strategy to get a new requirement change plan and improve its benefits; that is,

$$p_k(s') > p_k(s). \tag{13}$$

Under the new requirement change plan s',

$$P(s') = A\frac{1}{100m}\sum_{i=1, i\neq k}^{n} b_i' c_i + p_k(s'). \tag{14}$$

Since from strategy s to s' only the development state of x is changed, $b_i = b_i'$ $(i \neq k)$; then

$$\begin{aligned}P(s') - P(s) &= A\frac{1}{100m}\sum_{i=1, i\neq k}^{n}(b_i' - b_i)c_i + p_k(s') \\ &\quad - p_k(s) = p_k(s') - p_k(s).\end{aligned} \tag{15}$$

Based on formula (13), we know that $p_k(s') - p_k(s) > 0$, so $P(s') - P(s) > 0$, which is a contradiction with the hypothesis. Therefore, there is no non-Nash equilibrium change strategy that could maximize $P(s)$.

The Nash equilibrium is the equilibrium state that exists in the game. Based on the above analysis, we know that the

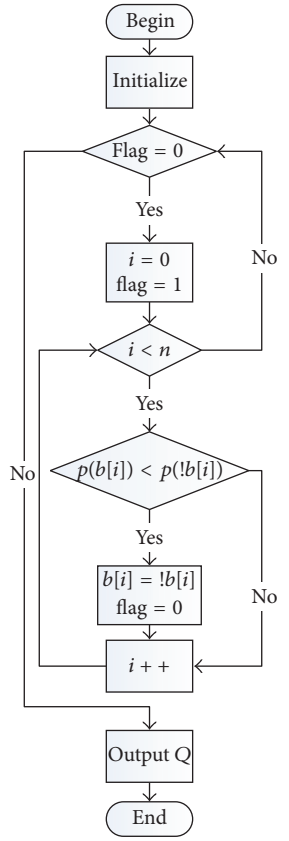

Figure 1: The flowchart of the game theoretical algorithm.

Nash equilibrium point is correlated to the optimal change strategy. Thus, when it is difficult to determine whether a change plan is optimal, one can use the game between the requirements to choose a better strategy.

3.2. Algorithm Design and Parameters Selection. Since the evaluation index P can only reach the maximum value when the game reaches Nash equilibrium, we designed the algorithm to optimize the overall adjustment scheme by looking for the Nash equilibrium in the game. The algorithm is simply referred to as the game theoretical algorithm. The flowchart of the algorithm is shown in Figure 1. The initialization step is to initialize all $b[i]$ to 1 and the flag to 0, respectively, which corresponds to the fact that all requirements are developed and initially are not the best overall adjustment strategy. When the flag equals 0, the algorithm will enter the first loop. In the loop body, the flag is firstly assigned a value of 1, indicating that it has not found that the Nash equilibrium condition is not satisfied. Then the algorithm goes into the second loop, which is an n times iteration, where the value of i is changed from 0 to $n - 1$. In each iteration, it will be determined whether the $i + 1$ game participant can improve his own income by unilaterally changing his own strategies. If so, game player's strategy will be updated and the flag is set to 0. Otherwise, it will enter the next iteration. So only when in the secondary loop and none of the game players could improve their benefits by changing their strategies, the value of flag can be maintained as 1. It indicates that the current game state has reached Nash equilibrium, so the algorithm can jump out of the primary loop. After jumping out of the primary loop, the evaluation index P is calculated using the $b[i]$ value and gets the output. Then the algorithm ends.

In principle, all the values of $b[i]$ can represent the found overall adjustment plans. In the practice, the output of $b[i]$ is the symbol of achieving the purpose of determining the overall adjustment program. Here the reason why we choose to output the evaluation index P is to verify the effectiveness of the game theoretical algorithm by comparing the result to the maximum value obtained from the exhaustive algorithm in the small-scale case. The set D of requirements used in the test is randomly generated from two to six stakeholders with five requirements per stakeholder. The number of stakeholders is randomly generated from 2, 3, 4, 5, and 6. In the system evaluation formula (8), the selection of parameters has two basic principles:

(1) The optimal solution that violates the objective facts is avoided. For example, in the two cases where all the requirements are developed or none is developed, although they can make the variance of stakeholder's satisfaction become 0, it is obvious that they both are unreasonable strategies. Therefore, the corresponding system scores should be bottomed up. If the difference between $|A|$ and $|C|$ is too

Table 2: Comparison of the results between the exhaustive algorithm and the game theoretical algorithm in a small case.

Number of requirements	Exhaustive algorithm	Game theoretical algorithm	Ranking
10	0.28082	0.264493	$2/2^{10}$
15	0.268151	0.253433	$6/2^{15}$
20	0.207848	0.184625	$10/2^{20}$
25	0.30348	0.267092	$32/2^{25}$
30	0.314467	0.287621	$134/2^{30}$

large, it will result in either of the two solutions getting a high system score. To avoid this issue, $|A| \approx |C|$ should be guaranteed when selecting the parameters.

(2) The proportion of the stakeholder's satisfaction variance in the system evaluation formula is guaranteed. The requirement selection algorithm based on the cost-value ratio is a simple and clear method to evaluate the requirement optimization result. However, because it does not fully guarantee the satisfaction of as many stakeholders as possible, it may lead to the emergence of particularly dissatisfied or extremely dissatisfied stakeholders, which will lay the hidden troubles that can result in software failures. The game theoretical algorithm adopts formula (8) as the system evaluation formula and introduces the stakeholder satisfaction variance as an important indicator to evaluate the change strategy, which could effectively avoid the above problem. As variable B is the summation weight of the stakeholder satisfaction variances in the system evaluation formula, in order to determine the reasonable range of B, we compare the optimal solutions that could be found by the game theoretical algorithm under different values of B. The cost-value ratio algorithm is to sort all the requirements based on their cost-value ratio (c/t) and then calculate the system evaluation index from the development of the first requirement to the first k requirements using formula (8). The value of k is from 1 to the total number of requirements (n), and the final output can reach the maximum P value. The comparison results between the cost-value ratio algorithm and the game theoretical algorithm are shown in Figure 2. The relative score $\widetilde{P}$ in the figure is defined as

$$\widetilde{P} = \frac{P_{\text{Gmax}} - P_{\text{Cmax}}}{P_{\text{Gmax}}} \times 100\%. \tag{16}$$

In the formula, P_{Gmax} and P_{Cmax} denote the system scores of the optimal solutions given by the game theoretical algorithm and the cost-value ratio algorithm, respectively. The results indicate that when $|B| = 0$, the game theoretical algorithm and the cost-value ratio algorithm will give the same result $(\widetilde{P} = 0)$. As $|B|$ increases, the game theoretical algorithm gives better results. Until $|B|$ reaches 5, the system score got by the game theoretical algorithm is nearly 50% higher than the cost-value algorithm. It demonstrates that the game theoretical algorithm will show its superiority when the proportion of stakeholder's satisfaction variance is large in the system evaluation formula.

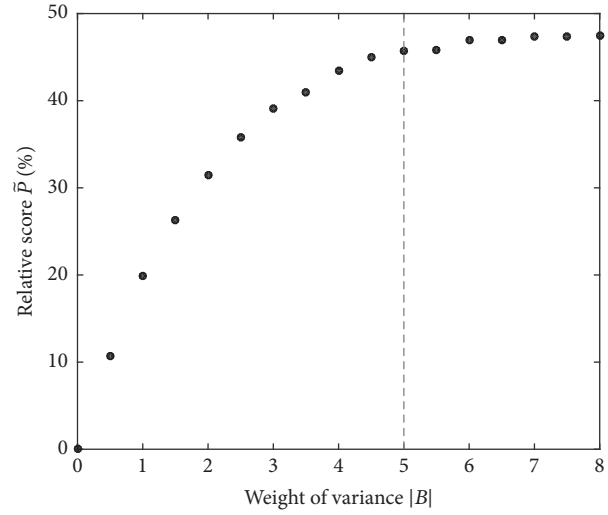

Figure 2: Comparison results between the cost-value ratio algorithm and the game theoretical algorithm.

4. Results Analysis

In order to analyze the performance of the game theoretical algorithm, we use the exhaustive search as a baseline method. The exhaustive search will go through all the possible solutions and calculate the system evaluation indices of these solutions. After that, it will select the solution that can maximize the P value. We then evaluate the performance of the game theoretical algorithm based on the advantages and computability of the proposed change schemes and finally demonstrate the practicability of the game theoretical algorithm through a case study.

4.1. Superiority Analysis of the Results by Game Theoretical Algorithm. Based on the above two basic principles for parameter selection, we select $A = 1$, $B = -5$, and $C = -1$ as parameters to conduct the experiment, in which the results of the exhaustive algorithm and the game theoretical algorithm are shown in Table 2. The first column shows the number of requirements. The second column and the third column show the maximum values of the evaluation indices P_{max} and P'_{max}, which can be found by the exhaustive algorithm and the game theoretical algorithm, respectively. The fourth column is the ranking of P'_{max} in all development strategies. It can be observed that P'_{max} obtained by the game

Table 3: The list of satisfaction and cost for each requirement in the case study.

	Requirement 1	Requirement 2	Requirement 3	Requirement 4	Requirement 5
Emp1	(11, 884)	(19, 319)	(36, 971)	(9, 943)	(25, 564)
Emp2	(6, 579)	(33, 859)	(16, 447)	(43, 620)	(2, 646)

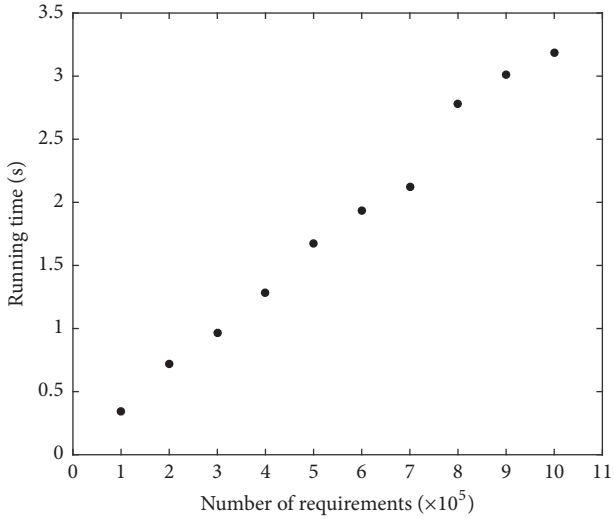

Figure 3: The running time of the game theoretical algorithm in a large-scale situation.

theoretical algorithm does not reach the true maximum. The reason is that there is more than one Nash equilibrium in the game, but after finding a Nash equilibrium point, the game theoretical algorithm will not continue looking for other Nash equilibrium states, leading to failure in finding a larger evaluation index. Even so, if we sort the evaluation indices corresponding to all possible development strategies in descending order, the ranking of $P'_{\max}$ found by the game theoretical algorithm is still very high. Therefore, the development strategy that is found by the game theoretical algorithm is superior to most of the other strategies, and the superiority of the overall adjustment scheme can be verified by the game theoretical algorithm.

4.2. Computability Analysis. The game theoretical algorithm determines the optimal overall adjustment strategy by looking for the Nash equilibrium states, and the superiority of the adjustment scheme has been verified in the small-scale test. Now we evaluate our proposal with large-scale cases, and the results are shown in Figure 3. It can be observed that as the number of requirements increases from 100,000 to one million, the running time of the program grows linearly, indicating that the time complexity of the algorithm is $O(n)$. Thus, our proposed algorithm is practical.

4.3. Case Study. The actual performance of the game theoretical algorithm is illustrated using a case study with two stakeholders, each of whom has five requirements. The two stakeholders are denoted as Emp1 and Emp2, and their requirements are listed in Table 3. Each requirement in the table is characterized by two parameters: the first parameter represents the satisfaction gain by implementing the requirement, and the second parameter represents the development cost for the requirement with a maximum value of 1000. The list of requirements is randomly generated according to the above conditions.

The optimization process is shown in Figure 4 when the system evaluation parameters A, B, and C are set to be $(1, -5, -1)$ for the overall adjustment scheme. The square matrix of 32×32 in the figure is the adjustment strategy space of the developer, and each element represents an adjustment strategy. After traversing the entire strategic space, the system evaluation score P for each strategy is represented in a color, with a darker color indicating a greater P value. All development strategies having a negative P value are treated as $P = 0$. Finally, the global optimal solution s in this example is to use the strategy $s1(01101)$ for Emp1 and $s2(01010)$ for Emp2.

Compared with the exhaustive method that requires traversing the whole solution space, the game theoretical algorithm based on Nash equilibrium is much more efficient. The process of finding the optimal solution from the lower right corner of the square matrix is shown in Figure 4. We can see that the Nash equilibrium point is achieved through five steps, and, in each step, the P value is increased, and the solution of the Nash equilibrium point is exactly the optimal solution in the whole strategy space. In practice, the game theoretical algorithm only looked up 20 requirement changing strategies to get the globally optimal solution, compared to 32×32 times of query with the exhaustive algorithm, which shows the efficiency of our proposal.

5. Conclusions

Effective software requirement change management methods can facilitate the green computing by efficiently using the computing resources and recycling the software system. In this paper, we propose an algorithm based on the game theory and treat the requirement change management as the game between the stakeholders and the developer. We construct the payoff matrix for the stakeholders' requirement requests and the developer's responses. We analyze the situations of rejecting or accepting the requirement change and propose the process to obtain the equilibrium point for the mixed game strategy. We also propose the mathematical model to find the optimal scheme of the overall adjustment by using the Nash equilibrium principle. The evaluation results show that, in the small-scale cases, the overall adjustment program determined by Nash equilibrium is almost close to the optimal result obtained by the exhaustive algorithm. And

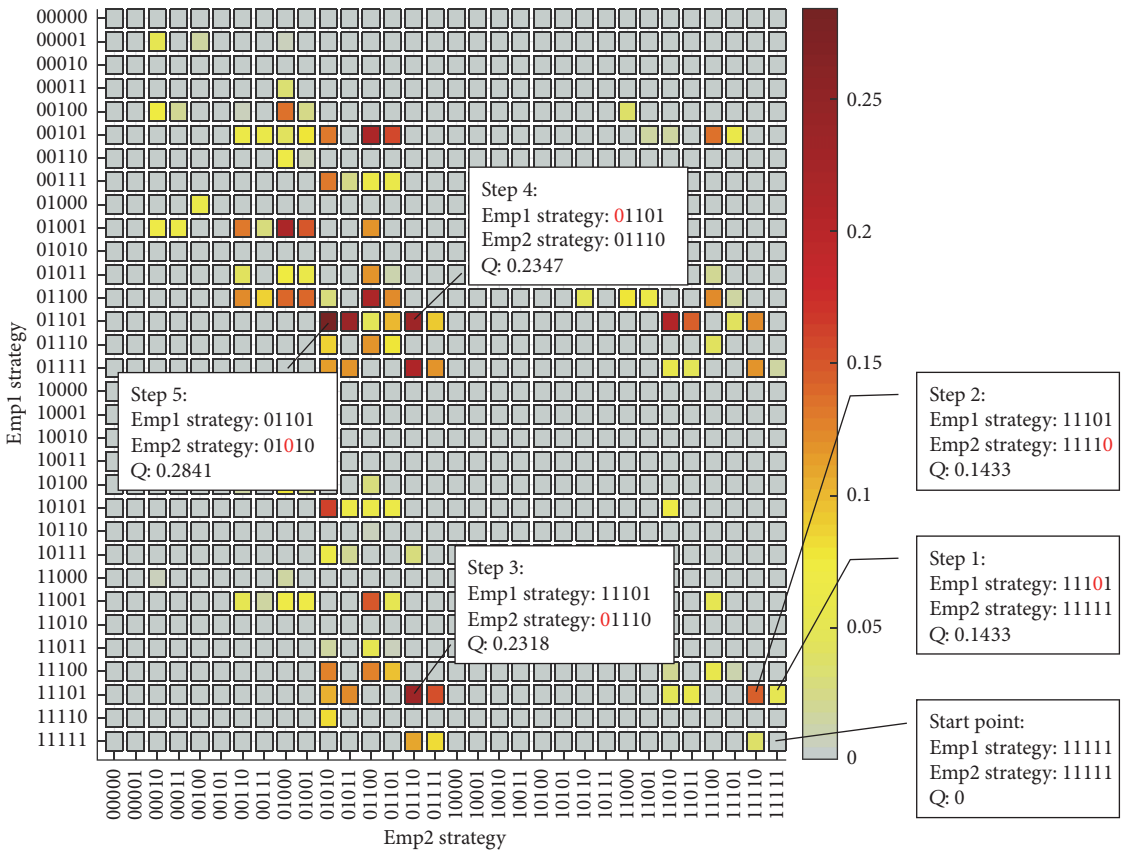

Figure 4: Optimization of the overall adjustment scheme based on the game theoretical algorithm.

as the number of requirements increases from hundreds of thousands to one million, the running time of the program grows in a linear way, demonstrating the feasibility of our proposed method in practical applications with large-scale requirement changes.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported by the "13th Five-Year" National Science and Technology Major Project of China (Grant no. 2017YFC0702204) and the National Natural Science Foundation of China (Grant no. 61173021).

References

[1] A. Kobayashi and M. Maekawa, "Need-based requirements change management," in *Proceedings of the Eighth Annual IEEE International Conference and Workshop On the Engineering of Computer Based Systems-ECBS 2001*, pp. 171–178, Washington, DC, USA.

[2] S. Ramzan and N. Ikram, "Requirement change management process models: Activities, artifacts and roles," in *Proceedings of the 10th IEEE International Multitopic Conference 2006, INMIC*, pp. 219–223, December 2006.

[3] M. W. Bhatti, F. Hayat, N. Ehsan, A. Ishaque, S. Ahmed, and E. Mirza, "A methodology to manage the changing requirements of a software project," in *Proceedings of the 2010 International Conference on Computer Information Systems and Industrial Management Applications, CISIM 2010*, pp. 319–322, October 2010.

[4] M. Makarainen, "Software change management processes in the development of embedded software," Tech. Rep., VTT Technical Research Center of Finland, ESPOO, 2000.

[5] J. L. Mate and A. Silva, *Requirements Engineering for Sociotechnical Systems*, Information Resources Press, 2005.

[6] A. A. Khan, S. Basri, P. D. D. Dominic, and Fazal-E-Amin, "A process model for requirements change management in collocated software development," in *Proceedings of the IEEE Symposium on E-Learning, E-Management and E-Services, IS3e 2013*, pp. 11-6, 2013.

[7] A. Rozan and M. Zaidi, "Change requirement management issues for a large software development projects," *Journal of Information Systems Research and Innovation*, pp. 63–69, 2013.

[8] S. Ramzan and N. Ikram, "Making decision in requirement change management," in *Proceedings of the 1st International Conference on Information and Communication Technology, ICICT 2005*, pp. 309–312, August 2005.

[9] Z. Ahmad, M. Hussain, A. Rehman, U. Qamar, and M. Afzal, "Impact minimization of requirements change in software project through requirements classification," in *Proceedings of*

the 9th International Conference on Ubiquitous Information Management and Communication, ACM IMCOM 2015, vol. 15, pp. 15:1–15:5, January 2015.

[10] J. Tomyim and A. Pohthong, "Requirements change management based on object-oriented software engineering with unified modeling language," in *Proceedings of the 7th IEEE International Conference on Software Engineering and Service Science, ICSESS 2016*, pp. 7–10, August 2016.

[11] M. Oertel and A. Rettberg, "Reducing re-verification effort by requirement-based change management in," in *Proceedings of the International Embedded Systems Symposium*, pp. 104–115, 2013.

[12] V. Gaur, A. Soni, and P. Bedi, "An application of multi-person decision-making model for negotiating and prioritizing requirements in agent-oriented paradigm," in *Proceedings of the 2010 International Conference on Data Storage and Data Engineering, DSDE 2010*, pp. 164–168, February 2010.

[13] F. Golpayegani, K. Azadbakht, and R. Ramsin, "Towards process lines for agent-oriented requirements engineering," in *Proceedings of the IEEE EuroCon 2013*, pp. 550–557, July 2013.

[14] A. Fleischmann, U. Kannengiesser, W. Schmidt, and C. Stary, "Subject-oriented modeling and execution of multi-agent business processes," in *Proceedings of the 2013 12th IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2013*, vol. 02, pp. 138–145, November 2013.

[15] S. Bendakir, N. Zarour, and P. J. Charrel, "A novel approach to change management in requirements engineering context," *International Journal of Agent Technologies and Systems*, vol. 7, no. 3, pp. 18–44, 2015.

[16] S. M. A. Bokhari and M. Abbas, "A logic-based framework for the management of changing software requirements," *Lecture Notes on Software Engineering*, vol. 3, no. 4, pp. 275–278, 2015.

[17] H. Ahmed, A. Hussain, and F. Baharom, "Current challenges of requirement change management," *Journal of Telecommunication, Electronic and Computer Engineering*, vol. 8, no. 10, pp. 173–176, 2016.

[18] A. AlSanad and A. Chikh, "Software requirements change managementa comprehensive model," in *Proceedings of the World Conference on Information Systems and Technologies*, pp. 821–830, 2017.

[19] H. Hu, Y. Wen, and D. Niyato, "Spectrum allocation and bitrate adjustment for mobile social video sharing: potential game with online QoS learning approach," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 4, pp. 935–948, 2017.

[20] H. Hu, Y. Wen, and D. Niyato, "Public cloud storage-assisted mobile social video sharing: a supermodular game approach," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 3, pp. 545–556, 2017.