2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

An Optimal Software Maintenance Policy Based on Reliability and Risk

WANG Xiaoping, ZHOU Fang, SHEN Yi

***Abstract*—Software maintenance is assuming ever more a crucial role in the lifecycle of software due to the increase of software requirements and the high variability of software environment. Common approaches of studying software maintenance are to consider them as a static by-product of software operation and only the maintenance cost is covered. In this paper, software maintenance policies are studied with the consideration of software reliability and risk. An optimization model is defined to drive the choice of a maintenance schedule. The solution of the model provides the best maintenance policy and the choice of actual actions that will minimize the average maintenance cost while the software reliability and risk are acceptable. Finally, a numerical example is given to show the analysis process of our proposed policy.**

I. INTRODUCTION

With the rapid development of information technology and the popularity of computer, software has been used in every aspect of people's work and life which indicates a sharp rise trend in its importance and scale. As we can see from the lifecycle of software, before delivered to users, most of the software will experience a relatively short time of designing and developing. Then the software will enter the operation and maintenance phase until the end of the whole lifecycle. According to statistics, software maintenance accounts for more than 70% of work in its entire lifetime typically. It is necessary to modify and upgrade the software constantly during the long time of the operation phase for correcting the new coming errors, adapting to the new environment and meeting the users' needs. The work will take much resources such as human, material and financial resources and bring new errors sometimes [1].

The discipline of software maintenance began in the late 1970s. Because the early development of software itself was not mature, most of the academic research focused on the software development and the research about maintenance was very rare. The proportion of maintenance cost throughout the software lifecycle increases gradually. The method to reduce the risk and cost effectively, make maintenance strategy to ensure a higher reliability that has become an issue of concern in the Software Industry increasingly. Besides various conferences, the number of academic works and organizations on software maintenance is also increasing. Some researchers focus on the basic structure of the software and compare the impacts of different design patterns on software maintenance [2], whereas others invest different maintenance policies [3, 4] including the comparison and optimization of these different maintenance policies [5, 6]. A model is built to determine the optimal point for maintaining a software application in [5]. Tan and Mookerjee propose a model and operating policy that reduce the sum of maintenance and replacement costs in the useful life of a software system in [6].

Currently, most of the literatures on maintenance policy focus on the maintenance cost. There are also some researchers who take software reliability into account to make the optimal policy. Xiong, Xie and Hui propose an optimal maintenance policy which aims at minimizing the average maintenance time cost in [7]. An optimization model is defined to drive the choice of a maintenance plan (i.e. a set of maintenance actions to be taken) in correspondence of a certain change scenario in [8]. The solution of such model provides the set of actions that minimize the maintenance cost while guaranteeing software reliability. In [9], Tian, Lin and Wu develop an approach related with the physical programming of software to deal with the multi-objective conditions based maintenance optimization problem which includes maximizing reliability and minimizing maintenance costs. The critical point is that the two main optimization objectives are often conflicting. With the proposed approach, the decision maker can systematically and efficiently make good tradeoff between the cost and reliability. As we can see from these related studies, maintenance cost is usually the objective in the model of software maintenance policy. But in the real maintenance work, it is far from enough for the decision maker to take the cost into consideration only. We also need to consider the state of the software, the environment of the maintenance and so on. Based on the analysis above, we set up an optimal model which considers the reliability of software and the risk of software maintenance in this paper. The aim of the policy is to minimize the maintenance cost while ensuring the reliability and risk are under the certain ranges.

The remainder of the paper is organized as follows. Section II first introduces the scenario of software maintenance and the model of the process of software maintenance. The optimal software maintenance policy based on reliability and risk is introduced in Section III in details. In Section IV we

The authors are with the School of Automation and the Key Laboratory of Ministry of Education for Image Processing and Intelligent Control, Huazhong University of Science and Technology, Wuhan 430074 China (e-mail: wangxiaoping@hust.edu.cn, zf1141@163.com, yishen64 @163.com).

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

apply our approach to an example. Section V concludes the paper with the summary of the research and directions for future research.

II. SOFTWARE MAINTENANCE POLICY ANALYSIS

A. The Model of Software Maintenance Process

A variety of failures may occur after the officially operation of software. Some failures are due to the changes of data or processing environment in the operation process. It's essential to modify the software to adapt to the changes. Some users and data processing staffs often suggest improving the existing features, adding new features and improving the overall performance. It is necessary to modify the resource code so as to incorporating these requirements into the software. Others' target is correcting the potential bugs or design flaws which will expose under certain conditions. The software maintenance can be divided into adaptive maintenance, perfective maintenance, and corrective maintenance according to the three types of the failures' causes. The work needed in the three types of maintenance is different. In this paper, we only focus on corrective maintenance.

Currently, most of the models about software failure time are based on Markov chains or non-homogeneous Poisson process. Gokhale who unified most of the models described the software failure process as Non-homogeneous Continuous Time Markov Chain (NHCTMC) [10, 11]. Because software failure time and maintenance time have similar probability nature, so Gokhale and Lyu suppose that software maintenance time can also be modeled with NHCTMC. In this paper, the NHCTMC is adopted to model the behaviors of maintenance events.

As we can see, a NHCTMC can be represented by the transition probability uniquely. We define the maintenance process as $X(t)$, the repair rate as $\lambda(n,t)$, and the mean function of maintenance process as $m(t)$. n is the number of failures up to the present time t. We obtain $m(t)$ by (1).

$$m(t) = E\left[X(t)\right] = \int_0^t \lambda(n,s)ds \qquad (1)$$

We can get $\lambda(n,t)$ from historical data and use (1) to analyze the software maintenance events. Assuming the repair rate can be described as the non-homogeneous Poisson process (NHPP), the G-O classic model is adopted to calculate the repair rate in (2).

$$\lambda(n,t) = \alpha\beta\exp(-\beta t) \qquad (2)$$

The parameter α is defined as the total number of corrected failures while β means the efficiency of the correction. So $m(t)$ can be expressed by (3).

$$m(t) = \alpha\{1-\exp(-\beta t)\} \qquad (3)$$

The method of SLE or MLE can be adopted to estimate the values of α and β if we have effective historical data [12, 13].

B. Software Maintenance Policy Problems

It's necessary to spend time on finishing the corrective maintenance. In corrective maintenance, the corresponding errors can be tracked by the source code which is mapped and fixed. All the resources of software systems are required for the maintenance so the systems can not keep running until the maintenance is finished. The service quality of system is degraded because of the unavailable time. Once a failure occurs, we will not construct the maintenance at once but restore the system that is called software rejuvenation. The system will resume the normal operation after the rejuvenation. Maintenance will not start until the number of failures meets a certain value or certain threshold criteria are met [7]. The Fig.1 shows a simple example of operation and maintenance process of software systems.

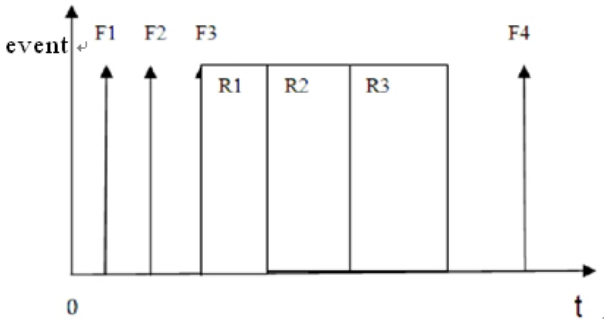

Fig. 1. Example of operation and maintenance of software systems

From Fig.1, maintenance will start after three failures occurred. F1, F2, and F3 are defined as software failures sequentially while R1, R2, and R3 represent the corresponding maintenance actions.

An important issue of software maintenance policy in this paper is how to determine the time point of constructing the maintenance which also means the cumulative number of failures in each maintenance schedule. The maintenance time point is closely related with environment and the state of software. With failures being resolved, the reliability of the software is increasing continuously. But the failures encountered in later phase will be more complex and it will need more time and high cost to complete the maintenance. Our goal in the paper is obtaining the time point of maintenance.

III. SOFTWARE MAINTENANCE POLICY BASED ON RELIABILITY AND RISK

A. Software Maintenance Cost

Software maintenance cost consists of three parts usually: setup cost, work cost, and loss cost [14].

1) Setup cost: It incurs when arranging and preparing maintenance resources before performing maintenance activities. Setup cost is unavoidable in the maintenance and very high for some large software systems. It is relatively stable when comparing with the other two parts. Usually we set it as a known constant. In this paper, it is a known constant and expressed as C_1 .

2) Work cost: It incurs in maintenance activities. Work cost is mainly the cost of human resource based on the characteristics of software maintenance. With the increase of maintenance time, work cost will increase linearly. We set it as C_2 and calculate it by (4).

$$C_2 = c_2 t \quad (4)$$

In (4), c_2 represents the maintenance cost unit time and t is the time that maintenance needs (It's also expressed as maintenance time in the paper).

3) Loss cost: It is caused by unavailable time in the maintenance phase. An apparent feature of loss cost is that its initial value is 0 and it will increase rapidly with the increase of maintenance time. But it can not increase to infinity because decision makers will take certain measures such as switching to a new system before the threshold is met. Loss cost is set as C_3 and expressed with a compound linear-exponential function as follows:

$$C_3 = c_3 t\{1-\exp(-\gamma t)\} \quad (5)$$

In (5), c_3 is the loss coefficient, γ is the form factor, and t is the maintenance time. As it shows in (5), the loss cost will increase steadily when maintenance time is short and it will increase rapidly when maintenance time is long.

From the analysis above, the total cost can be expressed as follows:

$$C(t) = C_1 + C_2 + C_3 = C_1 + c_2 t + c_3 t\{1-\exp(-\gamma t)\} \quad (6)$$

We obtain the value of C_1 , c_2 , and c_3 from the testing records or prior releases [14, 15].

B. Software Reliability

IEEE Computer Society of United States made a clear definition about "software reliability" in 1983. Then the definition was accepted as the national standard by American National Standards Institute. It was also accepted as the national standard by China in 1989. The definition includes two aspects of meaning [16]:

(1)The probability of that the failure won't occur under prescribed condition and time;

(2)The ability that procedure performs the required function under prescribed condition and within a predetermined time period;

The probability is a function not only of the system input and use but also of the errors in software. The system input will determine whether it will meet the failure (if the error exists).

Software reliability is the nature that whether software can meet the demand function. Software can not meet the requirement because of the software failure caused by software errors. Software reliability measurement refers to the quantitative evaluation of the degree of reliability. Software reliability index (refer to software reliability parameter) is the basis for describing software reliability, which is set as $R(t)$. $R(t)$ represents the probability of that the software performs the required function under prescribed condition and time or the failure will not happen within a specified time period. The parameter is the probability description of the behavior of software failures and the basic definition of software reliability. It can be identified as: $R(t) = P(T \le t)$. There are other indexs such as probability of failure, failure strength, failure rate, mean failure time and mean time between failures and so on.

In this paper, we choose the mean time between failures (MTBF) to represent the reliability. The MTBF is the average value of the failure time between two adjacent failures. Assuming ξ is the interval time, $F(t) = P(\xi \le t)$ is the function which means the cumulative probability density of software failure. So the function of reliability is $R(t) = 1 - F(t) = P(\xi > t)$, and the function of MTBF is as follows:

$$T_{BF} = \int_0^\infty R(t)dt \quad (7)$$

$$R(t) \ge R_0 \quad (8)$$

In (8), R_0 is the threshold value of the reliability.

C. Software Risk without Maintenance

TABLE I
THE CRITERIA TO QUANTIFY THE POSSIBILITY OF RISK

Possibility score	Quantitative description	Qualitative description
1	Less than 10%	Extremely low; Do not occur under normal circumstances.
2	10%-30%	Low; Occur only in rare cases.
3	30%-70%	Medium; Occur under certain circumstances.
4	70%-90%	High; Occur in many cases.
5	More than 90%	Extremely high; Often happen or almost happen certainly.

TABLE II
THE CRITERIA TO QUANTIFY THE IMPACT OF RISK

Impact score	Qualitative description	Impact on business
1	Very slight	Unaffected; A user can't use the system to construct the business properly.
2	Slight	Affected mildly; A business stops working.
3	Medium	Moderate impact; A application system is unavailable.
4	Considerable	Serious impact; One application system have failures and all or most of the businesses associated with the failure system are unavailable./ All or most of the businesses are unavailable in one area.
5	Catastrophic	Significant impact; Multiple (more than one) application systems have failures and all or most of the businesses associated with the failure systems are unavailable./ All or most of the businesses are unavailable in multiple (more than one) areas.

TABLE III
THE RISK LEVEL

Impact	Possibility		Low			Medium		High		
			Very low 1-1.5	Lower		Medium		Higher		Very high 4.5-5
				1.5-2	2-2.5	2.5-3	3-3.5	3.5-4	4-4.5	
High	Very high	4.5-5	Medium	High	High	High	High	High	High	High
	Higher	4-4.5	Medium	Medium	Medium	High	High	High	High	High
		3.5-4	Medium	Medium	Medium	Medium	High	High	High	High
Medium	Medium	3-3.5	Medium	Medium	Medium	Medium	Medium	Medium	High	High
		2.5-3	Low	Medium	Medium	Medium	Medium	Medium	Medium	High
	Lower	2-2.5	Low	Low	Low	Medium	Medium	Medium	Medium	High
Low		1.5-2	Low	Low	Low	Low	Medium	Medium	Medium	Medium
	Very low	1-1.5	Low	Low	Low	Low	Low	Medium	Medium	Medium

In this paper we analyze the risk quantitatively from two dimensions of the possibility (M) and impact (P). The first thing is to quantify the possibility and impact of risk according to different levels. The criteria of score are shown in Table I and Table II.

Software risk (F) = possibility (M) ×impact (P). We build a two-dimensional risk matrix with possibility (M) as the abscissa and impact (P) as the vertical axis. The risk matrix is shown in Table III.

According to the method mentioned above, we make qualitative and quantitative analysis of risk. The risk must be controlled within a certain range during the maintenance phase that can be shown as follows：

$$F \le F_0 \tag{9}$$

In (9), F_0 is the threshold value of the risk.

D. Model of Optimal Maintenance Policy

The aim of this research about optimal maintenance policy is to decrease the total cost of the whole operation phase as much as possible. The analysis about cost described above is the cost of one maintenance schedule and it's not a good choice for the objective function. In order to achieve the goal, we change the objective to maintenance cost unit time (C_U). If the maintenance cost unit time is minimal in every maintenance schedule, the total cost throughout the entire lifecycle is the smallest certainly. Therefore, the minimum maintenance cost unit time is set to be the objective of the model that can be expressed as follows:

$$C_U(t) = C(t)/t \tag{10}$$

Currently the cumulative failures are expressed as F_1 , F_2 ,… F_i . The maintenance starts when $i = N$,where N is the decision variable and a positive integer. Obviously N is interrelated with maintenance time t .

The constraints of the model are software reliability and risk. Software reliability will decrease gradually with failures occurring continuously. We should ensure the reliability within a certain range. Reliability is expressed with mean time between failures. Assuming the interval time between the ith failure and the $(i+1)th$ failure is t_{BFi} and T_{BF0} is the minimum acceptable threshold, the constraint of reliability can be expressed by (11).

$$\frac{1}{N-1}\sum_{i=1}^{N-1} t_{BFi} \ge T_{BF0} \tag{11}$$

Similarly, the risk should be also kept within an acceptable range. Assuming that the failures are independent with each other, the possibility of the risk caused by the ith failure is m_i and the impact is p_i , the constraint of risk can be expressed by (12).

$$\max(m_i p_i) \le F_0; i = 1,2,...N \tag{12}$$

The model of software maintenance policy can be expressed as follows:

$$\begin{aligned} &\text{minimize:} \quad C_U(t) = C(t)/t \\ &\text{subject to:} \quad m(t) = N \\ &\qquad \frac{1}{N-1}\sum_{i=1}^{N-1} t_{BFi} \ge T_{BF0} \\ &\qquad \max(m_i p_i) \le F_0; i = 1,2,...N \\ &\qquad N \ge 1, N \text{ is a positive integer.} \end{aligned} \tag{13}$$

Firstly, we let the first-order derivative of $C_U(t)$ 0. Then the minimum value can be obtained. Because N is a positive integer， the optimal solution can be expressed as $\lfloor m(t) \rfloor$ or $\lfloor m(t) \rfloor + 1$, where $\lfloor m(t) \rfloor$ is the maximum positive integer which is smaller than $m(t)$. Since a few constraints still exist in the model, the smallest solution of the objective function can not be chose as the best solution. The Fig.2 shows the analysis process after the failure has happened.

Assuming the number of the accumulated failures is i currently, the first thing is to analysis that whether the values of reliability and risk are beyond the thresholds. Once one of the thresholds is exceeded, we need to construct the maintenance immediately and set $N = i$. If the values are all under the threshold ranges, we should calculate the best maintenance time and obtain the optimal value of N according to NHCTMC of the maintaining process which can be set as N_o . If $i = N_0$, we set $N = i$ and start the maintenance at once. If $i < N_0$, we will not start the maintenance and just wait for next failure.

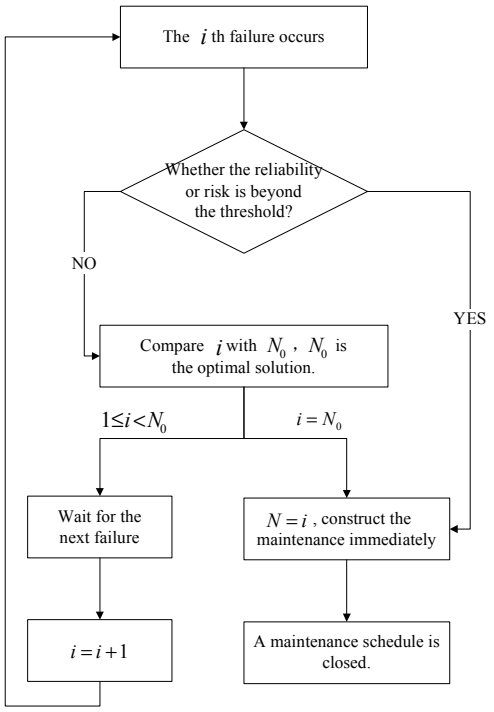

Fig. 2. Analysis process after the failure has happened

IV. EXAMPLE OF MAINTENANCE POLICY

We obtain the best maintenance policy easily if there are enough information and an effective model. A major problem in quantitative analysis of maintenance policy is the lack of information and data. It's difficult to get maintenance cost data of the actual company by public means. In this section, we use the preset values of cost parameters from [7] as $C_1 = 100, c_2 = 10, c_3 = 1000, \mathrm{r} = 0.5$. According to the data in operation and maintenance phase of the Apache server, we get the derived parameters: $\alpha = 66.81, \beta = 0.2139$. Maintenance cost unit time $C_U(t)$ can be calculated by (14).

$$\begin{aligned} C_U(t) &= \frac{C(t)}{t} = \frac{C_1 + c_2 t + c_3 t\{1 - \exp(-\gamma t)\}}{t} \\ &= C_1 / t + c_2 + c_3\{1 - \exp(-\gamma t)\} \\ &= \frac{100}{t} + 10 + 1000\left(1 - e^{-0.5t}\right) \end{aligned} \tag{14}$$

We get the smallest value of $C_U(t)$ by (15).

$$\frac{\partial C_U(t)}{\partial t} = 0 \tag{15}$$

In this case, the value of t is 0.5078 and the theoretical value of corresponding N can be calculated by (16).

$$\begin{aligned} N_E &= \alpha\{1 - \exp(-\beta t)\} \\ &= 66.81 \times \left(1 - e^{-0.2139t}\right) = 6.8747 \end{aligned} \tag{16}$$

Since N is a positive integer, we get the optimal solution of 6 or 7 without considering reliability and risk. When $N = 6$, we get the results of $t = 0.4399$ and $C_U(t) = 434.72$ .When $N = 7$, we get the results of $t = 0.5175$ and $C_U(t) = 431.22$ .Finally, we obtain the optimal solution is 7 by comparing the results calculated above.

Assuming that the number of accumulated failures is six currently, we should make quantitative analysis of software reliability and risk in the current state according to the method described previously. If the values of reliability and risk are beyond the accepted ranges, we should construct the maintenance immediately and set $N = 6$ at this same time. If the values do not exceed the threshold, we will not start the maintenance until the 7th failure occurs.

V. CONCLUSION

In this paper, we built a model of software maintenance policy based on reliability and risk. The target of maintenance policy is minimizing the total cost of the entire operation and maintenance phase while guaranteeing the software reliability and risk within the threshold. From the model we can see that N is the key parameter of the maintenance strategy and obtain the optimal number of cumulative failures N in section III. After software failure occurs, we should analyze the software at first. If the values of the software reliability and the risk without maintenance is within the accepted ranges, we will not begin the maintenance until the number of accumulated failures reaches the value of N . At this time the whole maintenance cost is the lowest. If the value of software reliability or risk exceeds the threshold, we should start the maintenance immediately regardless of whether the number of cumulative failures achieves the optimal value of N .

Results of this paper are based on certain assumptions such as NHCTMC of the maintenance process and so on. It's necessary to adjust the policy according to the specific circumstances during the actual maintenance process which is also a point for further research. In this paper, the type of maintenance is only corrective maintenance. We can take the other two types of maintenance into consideration in the future studies so as to make the research more general. Factors we should also consider in latter work include the maintenance order, the risk caused by the maintenance activities and so on.

ACKNOWLEDGMENT

The authors acknowledge the valuable comments and suggestions by the reviewers and the editors which have created an in-depth paper.

The work is supported by National Science Foundation of China (No.61374150, 61374171), by the State Key Program of National Natural Science of China (No.61134012), by National Basic Research Program of China (973 Program 2011CB710606), by the Fundamental Research Funds for the Central Universities (2013TS126), and by the Research Fund of Yalong River Hydropower Development Company LTD.

REFERENCES

[1] C. McClure, "The Three R's of Software Automation-Reengineering, Repositor, Reusability," New Jersey: Prentice-Hall, 1992.

[2] Schneidewind NF, "Experience report on using object-oriented design for software maintenance, Journal of Software Maintenance and Evolution: Research and Practice," vol. 19, pp. 183-201, 2007.

[3] Ahmed RE, "Software maintenance outsourcing: Issues and strategies," Computers and Electrical Engineering, vol. 32, pp. 449-453, 2006.
[4] Bhatt P, Shroff G, Anantaram C, Misra AK, "An influence model for factors in outsourced software maintenance," Journal of Software Maintenance and Evolution: Research and Practice, vol. 18, pp. 385-423, 2006.
[5] Feng Q, Mookerjee VS, Sethi SP, "Optimal policies for the sizing and timing of software maintenance projects, European Journal of Operational Research," vol. 173, pp. 1047-1066, 2006.
[6] Tan, Y, Mookerjee VS, "Comparing uniform and flexible policies for software maintenance and replacement," IEEE Transactions on Software Engineering, vol. 31, no. 3, pp. 238-255, 2005.
[7] Cheng-Jie Xiong, Min Xie, Szu-Hui Ng, "Optimal software maintenance policy considering unavailable time," Journal of Software Maintenance and Evolution: Research and Practice, vol. 23, pp. 21-33, 2011.
[8] Vittorio Cortellessa, Raffaela Mirandola, Pasqualina Potena, "Selecting optimal maintenance plans based on cost/reliability tradeoffs for software subject to structural and behavioral changes," in *Conf. Rec. 2010 14th European Conference on Software Maintenance and Reengineering,* pp. 21-30.
[9] Zhigang Tian, Daming Lin, Bairong Wu, "Condition based maintenance optimization considering multiple objectives," J Intell Manuf, vol. 23, pp. 333-340, 2012.
[10] Gokhale SS, Lyu MR, Trivedi KS, "Analysis of software fault removal policies using a non homogeneous continuous time Markov chain," Software Quality Journal, vol. 12, pp. 211-230, 2004.
[11] Gokhale SS, Lyu MR, Trivedi KS, "Incorporating fault debugging activities into software reliability models: A simulation approach," IEEE Transactions on Reliability, vol. 55, no. 2, pp. 281-292, 2006.
[12] Xie M, Software Reliability Modeling, Singapore: World Scientific Publisher, 1991.
[13] Lyu MR, Handbook of Software Reliability Engineering, New York NY: McGraw-Hill, 1996.
[14] Koskinen J, Salminen A, Paakki J, "Hypertext support for the information needs of software maintainers," Journal of Software Maintenance and Evolution: Research and Practice, vol. 16, pp. 187-215, 2004.
[15] Zhang XM, Pham H, "Software field failure rate prediction before software deployment, The Journal of Systems and Software," vol. 79, pp. 291-300, 2006.
[16] Zhao Chenggui, Pu Zhaobin, "A software reliability assessment model and its Petri net description," Computer Applications and Software, vol. 29, no. 1, pp. 141-144, 2012.