Applying PNZ Model in Reliability Prediction of Component-Based Systems and Fault Tolerance Structures Technique

Pham Binh[1(✉)], Huynh Quyet-Thang[1], Nguyen Thanh-Hung[1], and Nguyen Hung-Cuong[2]

[1] Department of Software Engineering, School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Vietnam
binh.pham92@gmail.com, {thanghq,hungnt}@soict.hust.edu.vn
[2] Math - Technology Faculty, Hung Vuong University, Viet Tri, Vietnam
cuongnh@hvu.edu.vn

Abstract. Reliability is the chief quality that one wishes for in anything. Reliability is also the main issue with computer systems. One of the purposes of system reliability analysis is to identify the weakness in a system and to quantify the impact of component failures. However, existing reliability prediction approaches for component-based software systems are limited in their applicability because they either neglect or do not support modeling explicitly several factors like error propagation, software fault tolerance mechanisms. In this paper, we evaluate reliability prediction of component-based system and fault tolerance structures technique by applying Pham Nordmann Zhang (PNZ) model, one of the best models based on non homogeneous Poisson process. Our approach uses a reliability modeling schema whose models are automatically transformed by a reliability prediction tool into PNZ models for reliability predictions and sensitivity analyses. Via these our case studies, we demonstrate its applicability and introduce how much reliability of software system can be improved by using fault tolerance structures technique.

Keywords: Software reliability prediction · Software reliability growth model

1 Introduction

Software reliability is one of eight main quality characteristics of software system [1]. This measure has a big number of applications in many phases of software life cycle: analysis, design, coding and testing. There are two approaches to work with this characteristic: evaluating [2–4] and predicting [5–7]. Trung et al. [7] introduced prediction scenario for component-based architecture, a modern technique of software engineering, with six steps.

Software reliability modelling is a mathematics model to evaluate some reliability properties of software system. There are more than hundred introduced

P.C. Vinh and V. Alagar (Eds.): ICCASA 2015, LNICST 165, pp. 272–281, 2016.
DOI: 10.1007/978-3-319-29236-6_27

models based on many mathematics techniques and work with many areas of project resources. One of the most developed group based on non-homogeneous Poisson process (NHPP) to build a time dependent function to present expected number of faults detected by time t. From this function, a practitioner can calculate some reliability measures of system as: the total number of errors, the predicted time of next failure. Pham [8] shows that Pham Nordman Zhang (PNZ) model is one of the best models in this group.

Fault tolerance structures technique (FTS) is a part of Software Fault Tolerance Mechanisms (FTMs). Avizienis et al. [9] describe in detail the principle of FTMs, and Trung et al. [7] introduced some basic concept of FTS. FTMs are often included in a software system and constitute an important means to improve the system reliability. FTMs mask faults in systems, prevent them from leading to failures, and can be applied on diffierent abstraction levels (e.g. source code level with exception handling, architecture level with replication) [10]. FTS only provides RetryStructure and MultiTryCatchStructure. Because in an FTMs, error detection is a prerequisite for error handling and not all detected errors can be handled. Therefore, at most, a RetryStructure or a MultiTryCatchStructure can provide error handling only for signaled failures, which are consequences of errors that can be detected and signaled by error detection.

Based on a good evaluation of PNZ model in NHPP group, we try to apply it into reliability prediction of component-based system and fault tolerance structures technique. Our study is organised as follows: after this introduction section, next section presents about software reliability modelling and PNZ model, the used model. Section 3 presents a reliability-prediction scenario to component based system and Sect. 4 introduces fault tolerance structures to improve software reliability. The last section shows some experimental results when apply those theoretical methods in real system.

2 Software Reliability Modelling and PNZ Model

Let's use some functions to describe characteristic of system when model it by non-homogeneous Poisson process in Table 1.

By time t, a system has $a(t)$ faults and $m(t)$ faults have been detected so we have $a(t) - m(t)$ remaining faults. With detection rate is $b(t)$, we have a relationship among number of faults detected in period Δt, total remaining faults of system and fault detection rate:

Table 1. Characteristic functions of software system

$a(t)$	Total number of faults
$b(t)$	Fault detection rate
$m(t)$	Expected number of fault detected by time t (mean value function)
$\lambda(t)$	Failure intensity

$$m(t+\Delta t) - m(t) = b(t)[a(t) - m(t)]\Delta t + o(\Delta t) \tag{1}$$

where $o(\Delta t)$ is infinitesimal value with Δt: $\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} = 0$. Let $\Delta t \to 0$:

$$\frac{\partial}{\partial t} m(t) = b(t)[a(t) - m(t)] \tag{2}$$

If t_0 is the starting time of testing process, with initial conditions $m(t_0) = m_0$ and $\lim_{t \to \infty} m(t) = a(t)$, Pham shows that general solution of (2) is [11]:

$$m(t) = e^{-B(t)} \left[m_0 + \int_{t_0}^{t} a(\tau) b(\tau) e^{B(\tau)} d\tau \right] \tag{3}$$

where

$$B(t) = \int_{t_0}^{t} b(s) ds \tag{4}$$

Pham et al. [11] introduce a Non-homogeneous Poisson process (NHPP) software reliability modeling (SRM) with time dependent functions:

$$a(t) = a(1 + \alpha t) \tag{5}$$

$$b(t) = \frac{b}{1 + \beta e^{-bt}} \tag{6}$$

So:

$$m(t) = \frac{a}{1 + \beta e^{-bt}} [(1 - e^{-bt})(1 - \frac{a}{\beta}) + at] \tag{7}$$

Existing publications show that PNZ model is one of the best model in NHPP sub-group.

3 Reliability Prediction for Component-Based System

3.1 Prediction Scenario

Our approach follows repetitively six steps [7] as depicted in Fig. 1.

3.2 Applying PNZ Model in Prediction Scenario

In step 3 of prediction scenario for component-based software system which has been shown in 3.1. After software architects create a system reliability model, the resulting model should be transformed into some kinds of model that can execute reliability, in this paper we use PNZ models.

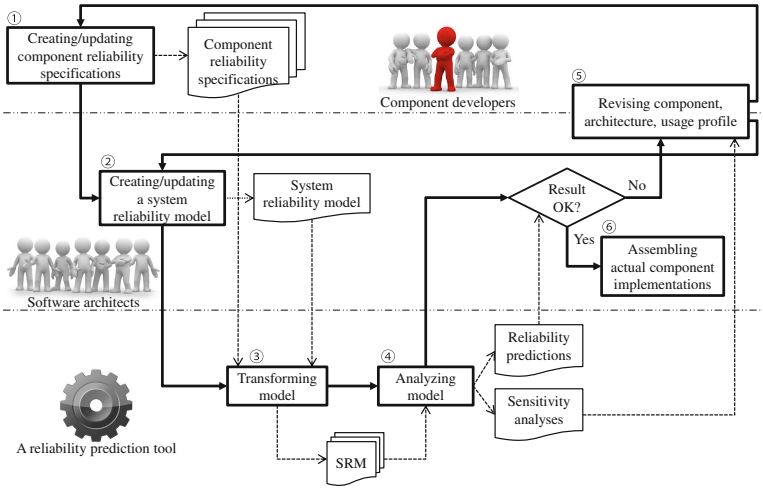

Fig. 1. Prediction scenario for component-based software system

In RetryStructure. For each possible input $I \in$ AIOS (Set of All sets of failure types) of a RetryStructure, the transformation instead of building a PNZ model like original from RMPI tool, we build a PNZ model that reflects all the possible execution paths of the RetryStructure with the input I and their corresponding probabilities, and then build up the failure model for the equivalent IA from this PNZ model.

Step 1: The transformation builds a PNZ block for each retry. The PNZ Block for the i^{th} retry (MB(I, RP_i)) reflects its possible execution paths for signaled failures (Fig. 2). It includes a state labeled "I, RP_i" ($[I, RP_i]$, for short) as an initial state, states $[RP_i, F]$ for all $F \in$ AFS as states of signaled failures. The probability of reaching state $[RP_i, F]$ from state $[I, RP_i]$ is $\Pr_{RP}(I, F)$ $\forall F \in$ AFS.

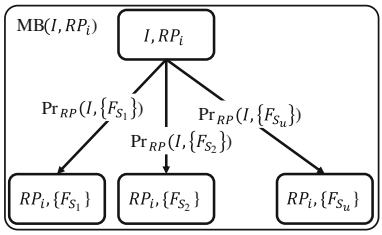

Fig. 2. PNZ block for i^{th} retry

Step 2: The transformation assembles these PNZ blocks into a single PNZ model that reflects all the possible execution paths of the RetryStructure with the input $I \in$ AIOS as follows:

- Add a state $[I, \text{START}]$.
- Add states $[F]$ for all $F \in \text{AFS}$.
- Add states $[O]$ for all $O \in \text{AIOS}$.
- Add a transition from state $[I, \text{START}]$ to state $[I, RP_0]$ with probability 1.0.
- For all PNZ block $\text{MB}(I, RP_i)$ with $i \in \{0, 1, \cdots, rc\}$, let rc be the retry count, add transitions from state $[I, RP_i]$ to state $[O]$ with probability $\text{Pr}_{PR}(I, O)$ for all $O \in \text{AIOS}$. This is because a correct (resp. erroneous) output of the RetryPart's execution leads to a correct (resp. erroneous) output of the whole RetryStructure.
- For PNZ block $\text{MB}(I, RP_{rc})$ (i.e. the PNZ block of the last retry), add transitions from state $[RP_{rc}, F]$ to state $[F]$ with probability 1.0 for all $F \in \text{AFS}$.
- For other PNZ blocks, i.e. $\text{MB}(I, RP_i)$ with $i \in \{0, 1, \cdots, rc-1\}$, add transitions from state $[RP_i, F]$ to
  (1) state $[I, RP_{i+1}]$ with probability 1.0 if $F \in F_H$, or otherwise to
  (2) state $[F]$ with probability 1.0 for all $F \in \text{AFS}$.

Step 3: After the transformation generated the PNZ model, the failure model for the equivalent IA is built up as follows:

- For all $F \in \text{AFS}$: $\text{Pr}_{IA}(I, F)$ is the probability of reaching absorbing state $[F]$ from transient state $[I, \text{START}]$.
- For all $O \in \text{AIOS}$: $\text{Pr}_{IA}(I, O)$ is the probability of reaching absorbing state $[O]$ from transient state $[I, \text{START}]$.

The transition matrix for the generated chain of PNZ blocks has the following format:

$$P = \begin{pmatrix} Q & R \\ O & I \end{pmatrix} \tag{8}$$

where the upper left transition matrix Q is a square matrix representing one-step transitions between transient states $[I, \text{START}]$, $[I, RPi]$, and $[RPi, F]$ for all $F \in \text{AFS}$ (with $i \in \{0, 1, \cdots, rc\}$), the upper right transition matrix R represents one-step transitions from the transient states to absorbing states $[F]$ for all $F \in \text{AFS}$ and $[O]$ for all $O \in \text{AIOS}$, I is an identify matrix with the size equal to the number of the absorbing states. Let $B = (I - Q)^{-1}R$ be the matrix computed from the matrices I, Q and R. Because this is an absorbing chain of PNZ Blocks, the entry b_{ij} of the matrix B is the probability that the chain will be absorbed in the absorbing state s_j if it starts in the transient state s_i. Thus, the failure model of the equivalent IA can be obtained from the matrix B.

In MultiTryCastStructure. Similar to the case of RetryStructures, for each possible input $I \in \text{AIOS}$ of a MultiTryCatchStructure, the transformation builds a PNZ model that reflects all the possible execution paths of the MultiTryCatchStructure with the input I and their corresponding probabilities, and then builds up the failure model for the equivalent IA from this PNZ model.

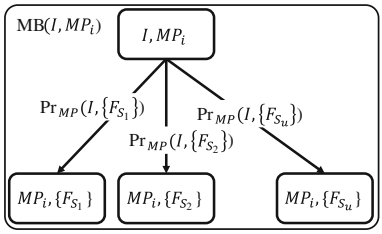

Fig. 3. PNZ block for MultiTryCatchPart i

Step 1: The transformation builds a PNZ block for each MultiTryCatchPart. The PNZ Block for the MultiTryCatchPart i (MB(I, MP_i)) reflects its possible execution paths for signaled failures (Fig. 3). It includes a state $[I, MP_i]$ as an initial state, states $[MP_i, F]$ for all $F \in$ AFS as states of signaled failures. The probability of reaching state $[MP_i, F]$ from state $[I, MP_i]$ is $\Pr_{MP_i}(I, F)$ for all $F \in$ AFS.
Step 2: The transformation assembles these PNZ blocks into a single PNZ model that reflects all the possible execution paths of the MultiTryCatchStructure with the input $I \in$ AIOS as follows:

- Add a state $[I, \text{START}]$.
- Add states $[F]$ for all $F \in$ AFS.
- Add states $[O]$ for all $O \in$ AIOS.
- Add a transition from state $[I, \text{START}]$ to state $[I, MP_i]$ with probability 1.0.
- For all PNZ blocks MB(I, MP_i) with $i \in \{1, 2, \cdots, n\}$, let n be the number of MultiTryCatchParts, add transitions from state $[I, MP_i]$ to state $[O]$ with probability $\Pr_{MP_i}(I, O)$ for all $O \in$ AIOS. This is because a correct (resp. erroneous) output of a MultiTryCatchPart's execution leads to a correct (resp. erroneous) output of the whole MultiTryCatchStructure.
- For PNZ block MB(I, MP_n) (i.e. the PNZ block of the last MultiTryCatchPart), add transitions from state $[MP_n, F]$ to state $[F]$ with probability 1.0 for all $F \in$ AFS.
- For other PNZ blocks, i.e. MB(I, MP_i) with $i \in \{1, 2, \cdots, n-1\}$, add transitions from state $[MP_i, F]$ to
  (1) state $[I, MP_x]$ with probability 1.0 where $x \in \{i+1, i+2, \cdots, n\}$ is the lowest index satisfying $F \in F_{Hx}$, or to
  (2) state $[F]$ with probability 1.0 if no such index $x \in \{i+1, i+2, \cdots, n\}$ satisfying $F \in F_{Hx}$ for all $F \in$ AFS.

Step 3: Because the resulting PNZ model is an absorbing chain of PNZ blocks, the failure model for the equivalent IA is built up as follows. For all $F \in$ AFS, $\Pr_{IA}(I, F)$ is the probability of reaching absorbing state $[F]$ from transient state $[I, \text{START}]$. For all $O \in$ AIOS, $\Pr_{IA}(I, O)$ is the probability of reaching absorbing state $[O]$ from transient state $[I, \text{START}]$.

4 Improving Reliability of Software System by Fault Tolerance Structures

Avizienis et al. [9] describe in detail the principle of Software Fault Tolerance Mechanisms (FTMs). An FTM is carried out via error detection and system recovery. Error detection is to identify the presence of an error. Error handling followed by fault handling together form system recovery. Error handling is to eliminate errors from the system state, e.g. by bringing the system back to a saved state that existed prior to error occurrence. Fault handling is to prevent faults from being activated again, e.g. by either switching in spare components or reassigning tasks among non-failed components.

To support modeling FTMs, our reliability modeling schema provides Fault Tolerance Structures (FTSs) [12], namely RetryStructure and MultiTryCatch-Structure. Because in an FTM, error detection is a prerequisite for error handling and not all detected errors can be handled. Therefore, at most, a RetryStructure or a MultiTryCatchStructure can provide error handling only for signaled failures, which are consequences of errors that can be detected and signaled by error detection.

RetryStructure. An effective technique to handle transient failures is service re-execution. A RetryStructure is taking ideas from this technique. The structure contains a single RetryPart which, in turn, can contain different activity types, structure types, and even a nested RetryStructure. The first execution of the RetryPart models normal service execution while the following executions of the RetryPart model the service re-executions.

MultiTryCatchStructure. A MultiTryCatchStructure is taking ideas from the exception handling in object-oriented programming. The structure consists of two or more MultiTryCatchParts. Each MultiTryCatchPart can contain different activity types, structure types, and even a nested MultiTryCatchStructure. Similar to try and catch blocks in exception handling, the first MultiTryCatch-Part models the normal service execution while the following MultiTryCatch-Parts handle certain failures of stopping failure types and launch alternative activities.

After obtaining the results of system reliability, as well as the rate of occurrence of the error by using the tools RMPI. We will determine the type of error that has the highest rate of appearance and then follow the 3 steps below to reduce the possibility that errors occur and thereby improve the overall reliability of the entire system.

In system reliability model that system architects have designed from the beginning, we will add a module called fault-tolerant module, this module will be built right before the module that contains the error rates appear most which been identified above. We follow 3 steps:

Step 1. Model the component Fault-Tolerance.
Step 2. Create an instance of this component.
Step 3. Redefine some component connectors.

5 Experimental Results of WebScan Sub-system

5.1 Preparing Reliability Prediction Scenario

We take following steps to build reliability model.

Step 1: modeling services, components and service implementations.

Sub-step 1.1: first of all, we model all the services, here we have six services:

1. serveClientRequest.
2. configureScanSettings.
3. scan.
4. createNewDocument.
5. addPageToDocument.
6. saveDocument.

Sub-step 1.2: after that, we model three components.

Sub-step 1.3: then, we model service implementations for provided services of components.

Step 2: modeling failure models. We model all kinds of failure models include: propagating failure type and stopping failure types.

1. For propagating failure type:
   - ContentPropagatingFailure $\leftrightarrow F_{P1}$.
2. For stopping failure types:
   - ServingRequestFailure $\leftrightarrow F_{S1}$.
   - ConfiguringScanFailure $\leftrightarrow F_{S2}$.
   - ScanningFailure $\leftrightarrow F_{S3}$.
   - CreatingDocumentFailure $\leftrightarrow F_{S4}$.
   - AddingPageFailure $\leftrightarrow F_{S5}$.
   - SavingDocumentFailure $\leftrightarrow F_{S6}$.

Step 3: modeling system architecture and usage profile

Sub-step 3.1: first, we define architecure of the whole system.

Sub-step 3.2: then, we define component instances:

- ClientInteraction $\leftrightarrow$ clientInteraction
- WebScanControl $\leftrightarrow$ webScanControl
- DocumentManager $\leftrightarrow$ documentManager

Sub-step 3.3: after that, we define component connectors.

Sub-step 3.4: we define user interface(s). For WebScan sub-system, we only have one user interface corresponding to serverClientRequest service and called as webScanUI.

Step 4: using tool RMPI to predict WebScan sub-systems reliability. From this, failure F_{S2} "ConfiguringScanFailure" is the most frequent failure type. This failure occurred between 2 modules is "WebScanControl" model and "DocumentManager" model and its corresponding with"configureScanSettings" service.

5.2 Applying Fault Tolerance Structures Method

Follow 3 steps outlined in Sect. 4, we have

Step 1: model the component Fault-Tolerance.

Step 2: create an instance of this component WebScanControlFaultTolerance $\leftrightarrow$ webScanControlFaultTolerance

Step 3: redefine some component connectors.

5.3 Reliability Comparison

After we have completed the structural changes in the system like above, we use RMPI tool to compare results (before and after changing happened) by:

```
java -jar RMPITool.jar -p WebScan_WithFTS.xml Output.txt
```

Comparing the results of the reliability and rate of occurrence of system failures before and after Webscan sub-system have added fault-tolerant components FTS, we have the following result in Table 2. From this result:

- The predicted reliability has increased 0.042277 %.
- The predicted failure probability for "ConfiguringScanFailure" has decreased 99.84899954 %.

Table 2. Result after applying fault tolenrance structures method

	WebScan	WebScan_WithFTS
Reliability	0.9981865558446795	0.9986085685485016
CreatingDocumentFailure	1.5764455152E-4	1.5764455152E-4
ConfiguringScanFailure	4.22706018E-4	6.382880502600001E-7
ServingRequestFailure	2.25E-4	2.25E-4
AddingPageFailure	2.9063232776269386E-4	2.9063232776269386E-4
SavingDocumentFailure	1.5098630021824724E-4	1.5098630021824724E-4
ScanningFailure	3.085747310766729E-4	3.085747310766729E-4
ContentPropagatingFailure	2.5790022674295393E-4	2.5795525287075817E-4

6 Conclusions and Future Works

The article evaluates reliability prediction of component-based system and fault tolerance structures technique by applying PNZ model based on good evaluation of PNZ model in NHPP group. We presented prediction scenario for component-based architecture, a modern technique of software engineering, with six steps. We also introduced PNZ model and how to apply this model into reliability prediction of component-based system.

To apply our approach, component developers create component reliability specifications and software architects create a system reliability model using provide reliability modeling schema. Then, these artifacts are transformed automatically to PNZ models for reliability predictions and sensitivity analyses by our reliability prediction tool. After all, to improve reliability of software system by FTS, component developers can revise the components and/or software architects can revise the system architecture and the usage profile. Via case studies,

we demonstrated the applicability of our approach, also shown how much reliability of software system can be improved. This kind of helps can lead to more reliable software systems in a cost-effective way because potentially high costs for late life-cycle changes for reliability improvements can be avoided.

We plan to extend our approach with more complex error propagation for concurrent executions, to include more software FTSs, and to validate further our approach. We also plan to continue developing our reliability modeling schema and prediction tool to help component developers automatically provide component reliability specifications. Those future works was sketched and will further increase the applicability of our approach.

Acknowledgement. This research was supported by The National Foundation for Science and Technology Development (NAFOSTED) under Grant 102.03-2013.39: Automated verification and error localization methods for component-based software.

References

1. ISO/IEC-25010:2011: Systems and software quality requirements and evaluation (square) system and software quality models (square) (2011)
2. Rana, R.: Defect prediction & prevention in automotive software development (2013)
3. Roshandel, R.: Calculating architectural reliability via modeling and analysis. Ph.D. thesis, University of Southern California (2006)
4. Chengjie, X.: Availability and Reliability Analysis of Computer Software Systems Considering Maintenance and Security Issues. Ph.D. thesis (2011)
5. Brosch, F.: Integrated Software Architecture-Based Reliability Prediction for IT Systems, vol. 9. KIT Scientific Publishing, Karlsruhe (2012)
6. Larsson, M.: Predicting quality attributes in component-based software systems. Mälardalen University (2004)
7. Pham, T.-T., Defago, X.: Reliability prediction for component-based software systems with architectural-level fault tolerance mechanisms. In: Eighth International Conference on Availability, Reliability and Security, pp. 11–20. IEEE (2013)
8. Pham, H.: System Software Reliability. Springer, Heidelberg (2006)
9. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secure Comput. **1**(1), 11–33 (2004)
10. Pullum, L.L.: Software Fault Tolerance Techniques and Implementation. Artech House, Norwood (2001)
11. Pham, H., Nordmann, L., Zhang, Z.: A general imperfect-software-debugging model with s-shaped fault-detection rate. IEEE Trans. Reliab. **48**(2), 169–175 (1999)
12. Avižienis, A.: Fault-tolerance and fault-intolerance: complementary approaches to reliable computing. In: ACM SIGPLAN Notices, vol. 10, pp. 458–464 (1975)