Proceedings of the Korean Nuclear Society Autumn Meeting
Seoul, Korea, October 1998

Experimental Analysis of Specification Language Impact on NPP Software Diversity

Chang Sik Yoo and Poong Hyun Seong

Korea Advanced Institute of Science and Technology
373-1 Kusong-dong, Yusong-gu
Taejon, Korea 305-701

Abstract

When redundancy and diversity is applied in NPP digital computer system, diversification of system software may be a critical point for the entire system dependability. As the means of enhancing software diversity, specification language diversity is suggested in this study. We set up a simple hypothesis for the specification language impact on common errors, and an experiment based on NPP protection system application was performed. Experiment result showed that this hypothesis could be justified and specification language diversity is effective in overcoming software common mode failure problem.

1. Introduction

In safety-critical systems like nuclear power plants, extremely high reliability is required of its computer system. As the means of enhancing software reliability, software fault tolerance has been adopted in some industrial computer systems. The fundamental principle of fault-tolerant software system is the software diversity to cope with common mode failures through the efficient system composition of diverse but functionally equivalent software systems. To avoid the possibility of common mode failure in software system, it has been suggested that the software development process should be diversified in development teams[1], programming languages[2], development environments and tools, implementation algorithms[3] and requirements specifications[4,5]. However, the efficient method for software diversity achievement is not established yet and questions have been raised on what factor in software development has dominant efficacy on product software diversity improvement.

From the fact that software product quality is mainly determined at the early development stage, it was expected that the diverse requirement specifications should affect dominantly on software diversity. Some studies[4, 5] reported that the use of diverse specification languages enhanced software diversity, but in these studies how the specification language act on software diversity was not uncovered clearly.

In our study we proposed a hypothesis about specification language impact on software diversity. To

verify this hypothesis an experiment was designed and conducted. Result analysis by simple product code test shows that our hypothesis could be justified.

2. Hypothesis

Regarding the programming process as a translation of requirement specification into product code, specification language plays the role of delivering requirement information to programmers. However this information delivery process tends to be accompanied by some misinterpretations, and the consequence would come out as logical coding errors.

Thus it can be expected that when two specification languages describe a system in different ways, and two programmers are informed of the system through these two specification languages, the two programmers seem to misunderstand the system in different ways and make different logical errors. The way of misunderstanding would be affected by expressional characteristics of the specification languages.

3. Experiment

To verify the hypothesis that the different specification languages induce different logical implementation errors, an experiment was designed and conducted. The author, in three different specification languages wrote three different requirement specifications. Three specification languages are chosen that specifications tend to express the requirements in different ways: natural language (English), SCR (originates in the project name Software Cost Reduction) and Colored Petri Nets (Colored Petri Nets). Whereas SCR describes system requirements with mathematical relations represented as tabular forms, CPN depicts system with states and transitions through graphical notations. Besides this notational and expressional difference between the two formal languages, the successful experience of these two formal languages in industrial projects was considered for selection. Especially, SCR has been successfully applied in requirement analysis for NPP protection system software.

The target system is the variable over power trip function (VOPT) in the core protection calculator software for Ulchin 3&4 NPP. The system monitors reactor operation variables such as hot leg and cold leg temperatures, pressure, loop pump speeds, and excore neutron flux levels. Out of the monitored operation variables the setpoint for reactor trip is calculated, and if current reactor power is greater than or equal to this setpoint, the system raises trip signal. The operation variables monitoring and trip decision is conducted by the system every regular time intervals.

The equality and consistency across three specifications were checked through exhaustive comparison of SCR and CPN specifications with the natural language specification. The specification in SCR was nearly a product code itself. Due to its condition-event table notations, there was no room for misinterpretations or mistakes. The specification in CPN showed the advantage over the others in handling variables without missing, because CPN describes data flow with token movement. But some

difficulty was found in describing system behavior related to time steps with CPN syntax. We attribute this difficulty to CPN specification language characteristics. Natural language permitted comparatively flexible description of system, but many ambiguous statements found to be revised.

In their way of expressing requirement information, we thought that natural language and CPN are most contrastive. In the expressional and notational characteristics SCR-CPN pair and SCR-NAT pair are not so contrastive as natural language - CPN is.

Twelve programmers with similar career made out twelve product codes from these three requirement specifications. All programmers recruited were junior or senior students enrolled in department of Computer Science, KAIST. Three groups were made out of the twelve programmers. Each group worked with the same requirement specification. Some education for the relevant formal languages was given to the two groups on SCR and CPN specifications before product code implementation. A supervisor carefully controlled programmers' coding. Each group was required to program the code in the specified place in the department of Nuclear Eng. Building. No kind of communications between programmers was permitted. Programmers were permitted to ask only questions about formal language syntax and semantics, and the questions concerned with requirement itself or specification context were rejected. The natural language specification group was not allowed in ask any question.

4. Result analysis

Twelve product codes were inspected in desk checking for the moment. A checklist containing fifty-nine checkpoints was prepared. First the checklist was written on the basis of specification hard points, and as errors came to be found in codes, new checkpoints relevant to the errors were added.

The desk checking results are summarized in Table 1. After the thorough desk check, all program errors found were identified, labeled and arranged in tabular form. Table 2 is the list of the errors that appears more than once through twelve product codes, and the description for these common errors is given in Table 4. The specification contents was divided into sixteen parts, and labeled as A ~ P. Characters in error ID correspond to the label of the parts where the error appeared. CPN3, CPN5, etc. denotes product code ID. P.S. (partial sum) represents the number of the common errors within a group.

Most parts of the reactor trip function consist of simple comparative logic and computational statements. Thus target requirement was context-free on the whole, but errors revealed themselves in context-sensitive particular parts of specification. Errors concentrate in the two areas in program at large, i.e. the parts "I" and "N+O" of the specification. In the part "I", CPN group made eighteen errors but NAT (natural language) group made nine errors. This seems to attribute to CPN handicap to system time-behavior description. The part "I" describes the successive update of 5 variables relevant to the measured cold leg temperatures in current and past 4 time steps. CPN syntax is neither appropriate nor effective for describing such system behaviors, and the CPN specification was difficult to understand. Of the ten common errors (the errors that appeared more than once through twelve product codes), six errors occurred in this part. Most of them seem to be traced back to CPN

ineffectiveness on time-behavior description. Table 3 presents the contents and inferred origin of the common errors. From the results mentioned above, it could be said that CPN language characteristics induced common errors in the programs.

Seven errors were found in the "N+O" part of four codes that NAT group produced, but only three detected in that of CPN group. The part "N+O" plays the role of calculating and updating the trip setpoint at each time step. In this part a variable should be set initially when the program is executed for the first time. This initialization is described by simple token initial marking in CPN specification, but the natural language specification failed in describing this process clearly in efficient way. Another description in "N+O" part is about a simple comparative logic, but natural language specification used somewhat ambiguous expression in the description. Three NAT programmers made identical errors concerning this expression. These two kinds of common error in NAT codes show the impact of specification language prominently.

SCR specification induced few errors. We think that the condition macros and structured decision tables in SCR specification minimized programmer's confusions. Only eight errors were found in four codes from SCR specification. Moreover, we detected no common errors in the four SCR product codes. Thus we deduced that the programmers were not affected by expressional characteristics of SCR specification. If only one specification language should be selected among the three languages, the best one would be SCR.

5. Conclusion

The experiment results indicate that the codes from the same specification language are prone to common errors. On the contrary, the codes from the different specification languages contain few common errors. In addition, the expressional characteristics of the specification language induced errors on their own way. The hypothesis that the diverse specification languages enhance the logical coding error diversity could be justified within the scope of our experiment and result analysis.

It seems that all pairs of specification languages don't have such effect as NAT-SCR pair has on logical coding errors. In our experiment, natural language and CPN were very contrastive in their expressional characteristics, but in case of "NAT and SCR" or "CPN and SCR" the contrasts between specification languages were uncertain. The question of "What is the best choice for 2 diverse specification languages among many specification languages?" still remains unanswered.

Reference

1. A. Avizienis and L. Chen. On the implementation of N-version programming for software fault tolerance during execution. In proc. IEEE COMPSAC 77, pages 149-155, November 1977.
2. A. Avizienis, M. R. Lyu, and W. Schuetz. In search of effective diversity: a six-language study of fault-tolerant flight control software. In Digest of 18th FTCS, pages 15-22, Tokyo, Japan, June 1988.
3. L. Chen and A. Avizienis. N-version programming: a fault-tolerance approach to reliability of

software operation. In Digest of 8th FTCS, pages 3-9, Toulouse, France, June 1978.

4. J. Kelly and A. Avizienis. A specification-oriented multi-version software experiment. In Digest of 13th FTCS, pages 120-126, Milano, Italy, June 1983.
5. J. Kelly and S. Murphy. Achieving Dependability through the development process: a distributed software experiment. IEEE Transactions on Software Engineering, vol. 16. No. 2, Feb 1990.

	CPN3	CPN5	CPN8	CPN9	NAT3	NAT5	NAT8	NAT9	SCR3	SCR5	SCR8	SCR9	P.S.		
A2					•								0	1	0
A3					•								0	1	0
A4											•		0	0	1
A5										•			0	0	1
A6				•									1	0	0
B1			•										1	0	0
B2					•								0	1	0
B3					•								0	1	0
C1			•			•							1	1	0
C2					•								0	1	0
D1					•								0	1	0
E1								•					0	1	0
E2b							•						0	1	0
E2c											•		0	0	1
E5					•								0	1	0
F1					•								0	1	0
H1a			•										1	0	0
H1b							•						0	1	0
I1	•		•										2	0	0
I2	•		•	•									3	0	0
I3		•											1	0	0
I4a	•			•									2	0	0
I4b				•									1	0	0
I4c							•						0	1	0
I5			•										1	0	0
I6a		•	•		•								2	1	0
I6b		•											1	0	0
I6c			•		•								1	1	0
I6d			•										1	0	0
I7					•		•	•					0	3	0
I8	•												1	0	0
I9a											•		0	0	1
I9b											•		0	0	1
I10							•						0	1	0
I11					•								0	1	0
I12					•								0	1	0
I13										•			0	0	1
I14				•									1	0	0
I15				•									1	0	0
L1a					•								0	1	0
L1b					•								0	1	0
L2									•				0	0	1
M1											•		0	0	1
N1a		•	•										2	0	0
N1b					•	•	•						0	3	0
N2			•										1	0	0
N3					•								0	1	0
O1					•	•	•						0	3	0
O2				•									1	0	0
S1				•									1	0	0
	4	4	9	7	18	3	7	2	1	2	5	0			

Table 1. List of detected errors by desk checking

	CPN 3	CPN 5	CPN 8	CPN 9	NAT3	NAT5	NAT8	NAT9	SCR 3	SCR 5	SCR 8	SCR 9	P.S.
C1			•			•							**(1,1,0)**
I1	•		•										**(2,0,0)**
I2	•		•	•									**(3,0,0)**
I4a	•			•									**(2,0,0)**
I6a		•	•		•								**(2,1,0)**
I6c			•		•								**(1,1,0)**
I7					•		•	•					**(0,3,0)**
N1a*		•	•										**(2,0,0)**
N1b					•	•	•						**(0,3,0)**
O1					•	•	•						**(0,3,0)**
Sum	3	2	6	2	5	3	3	1	0	0	0	0	25

Table 2. List of errors that appears more than once (common errors)

♦Errors ID in bold letter indicate logical common errors induced by spec language, judging somewhat subjectively.

Error ID	Partial Sum	Inferred Origin
C1	(1,1,0)	Trivial miss
I1	(2,0,0)	Misunderstand transition firing sequence in CPN spec P-14
I2	(3,0,0)	Misunderstand transition firing sequence in CPN spec P-14
I4a	(2,0,0)	Trivial miss (?)
I6a	(2,1,0)	Misunderstand CPN initial marking rules, or mistake by CPN spec complexity. Trivial miss for NAT spec
I6c	(1,1,0)	Trivial miss
I7	(0,3,0)	NAT spec ambiguity
N1a	(2,0,0)	Misunderstanding CPN initial marking rules, or mistake by CPN spec complexity.
N1b	(0,3,0)	NAT spec ambiguity
O1	(0,3,0)	NAT spec ambiguity

Table 3. Contents and Inferred Origin of Errors that appears more than 2 times

<table>
<tr><th>Error ID</th><th colspan="2">Description</th></tr>
<tr><td>C1</td><td colspan="2">Use Thraw1 instead of Thraw2 in Th2 calculation statement</td></tr>
<tr><td rowspan="3">I1</td><td colspan="2">Wrong sequence of updating Tc1[x],Tc2[x]</td></tr>
<tr><td>Correct</td><td>Wrong</td></tr>
<tr><td>Tc1[4]=Tc1[3];
Tc1[3]=Tc1[2];
…
Tc1[1]=Tc1[0];
Tc1[0]=Tc1;</td><td>Tc1[0]=Tc1;
Tc1[1]=Tc1[0];
…
Tc1[3] =Tc1[2];
Tc1[4]=Tc1[3];</td></tr>
<tr><td rowspan="3">I2</td><td colspan="2">Update Tc1[0] first, then update Tc1[4]~Tc1[1]</td></tr>
<tr><td>Correct</td><td>Wrong</td></tr>
<tr><td>Tc1[4]=Tc1[3];
Tc1[3]=Tc1[2];
Tc1[2]=Tc1[1];
Tc1[1]=Tc1[0];

Tc1[0]=Tc1;</td><td>Tc1[0]=Tc1;

Tc1[4]=Tc1[3];
Tc1[3]=Tc1[2];
Tc1[2]=Tc1[1];
Tc1[1]=Tc1[0];</td></tr>
<tr><td rowspan="3">I4a</td><td colspan="2">Improper subscript of Tc[x] array in Tc1f~Tc2s calculation</td></tr>
<tr><td>Correct</td><td>Wrong</td></tr>
<tr><td>Tc1f[0]=a1f*Tc1[2]+a2f*Tc1[3]+a3f*Tc1f[1];
Tc1s[0]=a1s*Tc1[3]+a2s*Tc1[4]+a3s*Tc1s[1];</td><td>Tc1f[0]=a1f*Tc1[3]+a2f*Tc1[4]+a3f*Tc1f[1];
Tc1s[0]=a1s*Tc1[4]+a2s*Tc1[5]+a3s*Tc1s[1];</td></tr>
<tr><td>I6a</td><td colspan="2">Tc1[x],Tc2[x] initial marking are performed in each time step, not for the first execution</td></tr>
<tr><td>I6c</td><td colspan="2">Tc1f~Tc2s initial marking are performed in each time step, not for the first execution</td></tr>
<tr><td>I7</td><td colspan="2">Do not update internal variable Tc1f~Tc2s</td></tr>
<tr><td>N1a</td><td colspan="2">FOLLOW initial marking is performed in each time step, not for the first execution</td></tr>
<tr><td>N1b</td><td colspan="2">Do not implement FOLLOW initial marking process</td></tr>
<tr><td>O1</td><td colspan="2">Omit the statement SPVOPT=FOLLOW+DELSPV, for case SPVmin<=FOLLOW+DELSPV<=SPVmax</td></tr>
</table>

Table 4. Description for the common errors