Journal of Software Engineering and Applications, **2012, 5, 823-835**
http://dx.doi.org/10.4236/jsea.2012.510095 Published Online October 2012 (http://www.SciRP.org/journal/jsea)

A Test Case Prioritization through Correlation of Requirement and Risk

Miso Yoon, Eunyoung Lee, Mikyoung Song, Byoungju Choi*

Department of Computer Science, Ewha Womans University, Seoul, South Korea.
Email: misoyoon@ewhain.net, ddonggurami@ewhain.net, smk1916@gmail.com, *bjchoi@ewha.ac.kr

Received August 20th, 2012; revised September 22nd, 2012; accepted October 1st, 2012

ABSTRACT

Test case prioritization techniques have been focused on regression testing which is conducted on an already executed test suite. In fact, the test case prioritization for new testing is also required. In this paper, we propose a method to prioritize new test cases by calculating risk exposure value for requirements and analyzing risk items based on the calculation to evaluate relevant test cases and thereby determining the test case priority through the evaluated values. Moreover, we demonstrate effectiveness of our technique through empirical studies in terms of both APFD and fault severity.

Keywords: Test Case Priority; Risk-Based Testing; Requirement Analysis; Risk Exposure

1. Introduction

Exhaustive testing is typically not feasible, except in extremely trivial cases. In one case, execution of the entire test suite requires seven weeks for a single software product comprised of 20,000 lines of code [1,2]. Thus, a technique of selecting only a subset of all possible test cases is required based on one or more coverage criteria. Among the test case selection techniques which have both efficiency and effectiveness to reduce errors, "Test Case Prioritization Techniques" provide a more effective test execution by allowing testers to determine the priority of test cases and select test cases with the highest priority, according to the scope of a test suite depending on budget situation, which eventually leads the test cases with the highest priority to be executed earlier than lower priority test cases [1,2]. This paper presents a test case prioritization technique and demonstrates its effectiveness and efficiency through experiments.

Existing prioritization techniques are mainly based on regression testing. Therefore, they tend to prioritize test cases on the basis of previously executed test results. These methods, however, require a prior knowledge of the existence of faults and of which test cases expose which faults. A case in point in these methods is optimal prioritization. The optimal prioritization [3] is an ideal method in theory but it requires considering all possible test case orderings and therefore, must have a worst case runtime exponential in test suite size. Also, Total fault-exposing-potential prioritization which determines priority of test cases based on mutation analysis has constraints since it must have fault information obtained through past experiences [1,4]. Thus, the abovementioned methods cannot be applied to initial testing.

This paper proposes a Risk-based test case prioritization technique which can be applied to cases with no additional information obtained through execution results of test cases. Our Risk Based Testing (RBT) uses product risks derived from requirements as assessment criteria to prioritize test cases.

Existing RBT studies can be divided into two types; one is to employ risk exposure values derived from risk analysis results as seen in Risk-based Test case Derivation And Prioritization (Rite DAP) [5], and the other is to use severity and probability evaluated from test cases such as safety test [6]. These methods have subjective evaluation of risk exposure values and can be performed only when prior fault information exist. Thus, they have constraints given both need the existence of test results. In order to maintain objectivity in evaluation of risk exposure values, this paper presents a method to calculate risk exposure values based on weight given to each requirement item and employ these values to prioritize test cases.

Our Risk-based test case prioritization method is advantageous to prioritize test cases when there is no information gathered in previous runs of existing test case. We conduct empirical evaluation on our proposed technique in comparison with previously mentioned existing techniques, namely, optimal prioritization, safety test, FEP-Total in order to validate our method's effectiveness

*Corresponding author.

by comparing each method's the frequency of fault detection, Average Percentage of Fault Detection (APFD) and Risk Severity in terms of Percentage of Fault Detection.

The next section of this paper describes the existing studies. Section 3 presents a test case prioritization technique employing risk exposure values, and Section 4 discusses the results and analysis of our empirical studies. Section 5 presents overall conclusions of our study.

2. Related Work

Optimal prioritization [3] is a technique to determine priority of test cases that offer the greatest fault detection ability under the assumption that testers know fault information of a program and the existence of all faults that can be detected in the test cases. This method is the most optimal in theory but has a worst case runtime exponential in test suite size since testers must understand precise fault information and which test cases expose which faults.

As for methods using software coverage techniques, there are total statement coverage prioritization total branch coverage prioritization and total fault-exposing-potential prioritization (FEP) [1]. Total statement coverage prioritization instruments a program to prioritize test cases in terms of the total number of statements covering faults by counting the number of statements covered by each test case, then sorting the test cases in descending order of that number. While the total branch coverage prioritization uses test coverage measured in terms of program branches rather than statements to determine priority of test cases. FEP determines the priority based on mutation analysis. FEP (s, t) is the ratio of mutations, s, killed by test case, t, to the total number of mutations of s. The test case priority is determined in terms of the total number of FEP (s, t) covering fault statements. All of these methods are feasible only under the premise that fault information exist.

There are existing test case prioritization techniques employing software risk information. Rite DAP [5] inserts risk information to Activity Diagram to automatically generate priority of test case scenarios. This approach adds a new risk related stereotype "reaction" to the activity diagram. The priority of test case scenarios is determined in order of an entity's "reaction" having the highest sum of risk values. This method has disadvantages of having subjective evaluation of risk values.

Meanwhile, Safety test [6] determines test case priority by evaluating cost and severity probability of test cases. The cost of test cases will be taken by the conesquences of a fault as seen by a customer or a vendor. The severity probability is calculated by multiplying the Number of Defects N by Average Severity of Defects S $(\mathrm{N}\times\mathrm{S})$. This approach, however, has weakness in that the evaluation of cost is subjective and fault information should exist.

This paper presents an approach to systematically identify risk items from requirements and determine test case priority by using risk items, rather than fault information. We report the results of our empirical studies by comparing our method to above mentioned techniques.

3. Risk-Based Test Case Prioritization Technique

Risk is defined as a probability of the occurrence of harm or loss. Boehm defined the Risk Exposure [7] as the probability of an undesired outcome times the expected loss if that outcome occurs. The basic concept of the Risk-based Testing is that the more time should be invested in software areas having high risk exposure values. This paper: 1) Defines the product risk items; 2) Estimates the risk exposure values derived from requirements; and 3) Determines test case priority.

3.1. Risk Items

There are three forms of risk that can arise with respect to software; Project risk, Process risk and Product risk [8]. This paper is designed to propose test case prioritization technique needed to validate "product" quality. Thus, we will derive "product risk" items by using various objective documents.

The followings are product risk items suggested by IEEE1044.1-1995 [9], Wallace [10], Sullivan [11], Jha [12], Bach [13], Pertet [14].

- IEEE. Std. 1044.1-1995: Standards with regard to software anomalies. Refers to user-related risk symptoms and risk types.
- Wallace: Refers to risk items related to medical equipment software's failure.
- Sullivan: Refers to risk items with regard to database-related error type, error trigger and defect type.
- Jha: Risk catalog on known problems or potential problems in mobile application is derived based on Bach's Heuristic Test Strategy Model (Bach, 2006). The risk catalog is classified into Product Elements, Operational Criteria, Development Criteria, and Project Environment. Among them, see the product-related risk items.
- Bach: See the risk catalog analyzed through heuristic analysis.

Pertet: See risk items with regard to software failure in web applications.

This paper categorizes product risk items into software, data, interface, enhancement, platform groups and subcategories, based on the above mentioned risk items, as seen in, to develop test case prioritization technique. For the relationships between the risk items suggested in **Figure 1** and referred data, see the Appendix attached in this paper.

3.2. Calculation of Risk Exposure Values from Requirements

The way to estimate risk exposure values from requirements is conducted in the following steps.

3.2.1. Step 1: Determine Risk Weight of Requirements

Risk weight of requirement i, RE (Req_i) is determined by multiplying the Risk Threat Likelihood by Risk Impact. Risk Threat Likelihood means the probability that the requirement will meet failure and Risk Impact is the size or cost of that loss if the requirement turns into a problem. Namely, If Req_i = Requirement i, WL_i = Weight of Risk Threat Likelihood of Requirement i, WI_i = Weight of Risk Impact of Requirement i, then

$$\mathrm{RE}\left(\mathrm{Req}_i\right) = \mathrm{WL}_i * \mathrm{WI}_i * 10000 \qquad (1)$$

In order to estimate relative importance in terms of risk probability and impact of requirements, Analytic Hierarchy Process (AHP) Technique [15] is employed. The AHP is a process of decision making to consistently determine weight by selecting factors that can have impact on the decision making criteria, gradually dividing them into smaller factors to establish a hierarchy and make judgments based on pair wise comparison of the importance of these factors. This AHP approach makes it easy to compare between different characteristics regardless of units and determine preferences through the comparison. Thus, it is useful to determine priority of requirement items in terms of risk. However, the importance of requirements can vary by persons. To adjust this problem, we use average values of risk exposure values provided by a risk expert and settle pair wise comparison values in terms of risks among the requirement items.

For instance, print_tokens have total 18 requirement items. **Table 1** represents the requirements of print_tokens and the risk weights of each requirement. The risk threat likelihood (WL_i) and risk impact (WI_i) are estimated by applying AHP technique. The risk threat likelihood and impact values for the first requirement, produced by using AHP, are 0.080, 0.102, respectively. The risk weight RE (Req_i) calculated based on the formula suggested in Equation (1) is 82.

3.2.2. Step 2: Estimate the Risk Exposure Values

Next, risk exposure values are estimated by reflecting risk weight of requirements by risk item. Amland have defined risk exposure as the probability of a fault occurring times the cost if a fault occurs [16]. In order to calculate the probability of a fault occurring, we determine how many risks are related to requirements and assume that the more the risk is related to requirements, the higher probability of the fault occurs. Also, to calculate the cost when fault occurs, we use the risk weight of the requirement and assume that the cost is high when risk is related to the requirements with higher weight occur.

Table 2 is the metric to estimate risk exposure values of risk items. RM_{ij} indicates 1 when the risk item i, r_i, correlates with requirement j and shows 0 when the risk item i does not correlate with requirement j. RE (r_i) represents risk exposure values of r_i The values of RE (r_i) are estimated by using the following mathematical formula.

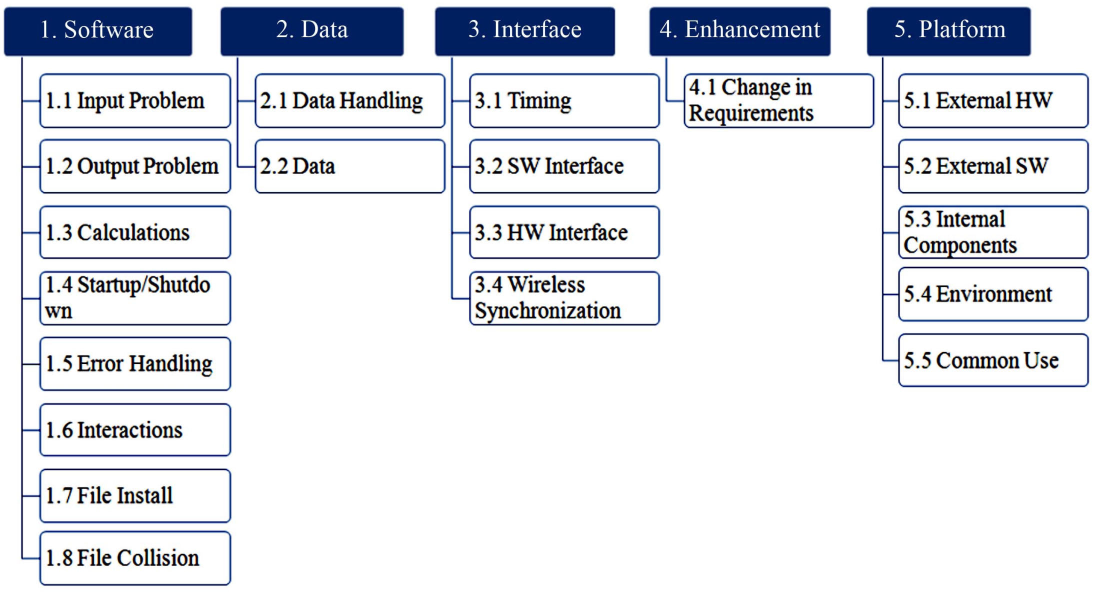

Figure 1. Product risk items.

Table 1. Risk weight values for requirements of print_tokens.

	Requirement R_i	WL_i	WI_i	RE (Req_i)
Req_1	Takes a file name as an input and analyzes all tokens in a file.	0.080	0.102	82
Req_2	If the file name doesn't exist, it will exit from the program.	0.073	0.094	68
Req_3	Takes character stream as an input return one character. If the stream is empty then it reads the next line from the file and returns the character.	0.066	0.094	62
Req_4	Checks whether it is end of character stream or not and check whether the last read character is end file character or not and returns the value according to it.	0.0354	0.0500	18
Req_5	The location of the read letter should be identified.	0.0354	0.0560	20
Req_6	Takes file name as an input and it gets character stream and returns the token stream.	0.0460	0.0690	32
Req_7	Returns the next token from the token stream.	0.0730	0.0980	72
Req_8	Checks whether the token is numeric token.	0.0210	0.0300	6
Req_9	Checks whether it is EOF or not.	0.0489	0.0200	10
Req_{10}	Checks whether the character is alphabet and number.	0.1620	0.0400	65
Req_{11}	Identify a keyword, if it is not a keyword, output an error message.	0.0386	0.0360	0.0014
Req_{12}	Identify a special character, if it is not a keyword, output an error message.	0.0386	0.0370	14
Req_{13}	Skip the characters until EOF or EOL found.	0.0247	0.0310	8
Req_{14}	Check whether token of string is constant.	0.0891	0.0400	0.0036
Req_{15}	Returns the next state in the transition diagram.	0.0368	0.0520	0.0019
Req_{16}	Checks whether the token is the end of token.	0.0386	0.0560	0.0022
Req_{17}	In the case of keyword, special character, print the type of token and ID of token.	0.0524	0.0620	0.0032
Req_{18}	In the case of identifier, numeric, string and character, prints the actual token and it removes the leading and trailing spaces and prints the token.	0.0412	0.0340	0.0014

Table 2. Risk exposure metric.

Risk item	Requirements					RE (r_i)
	Req_1		Req_j		Req_n	
	WL_1		WL_j		WL_n	
	WI_1		WI_j		WI_n	
	RE (Req_1)	⋯	RE (Req_j)	⋯	RE (Req_n)	
r_1	RM_{11}	⋯	RM_{1j}	⋯	RM_{1n}	RE (r_1)
⋯	⋯	⋯	⋯	⋯	⋯	⋯
r_i	RM_{i1}	⋯	RM_{ij}	⋯	RM_{in}	RE (r_i)
⋯	⋯	⋯	⋯	⋯	⋯	⋯
r_m	RM_{m1}	⋯	RM_{mj}	⋯	RM_{mn}	RE (r_m)

$$\mathrm{RE}\left(\mathrm{r_i}\right)=\sum\nolimits_{j=1}^{n}\left(\mathrm{RE}\left(\mathrm{Req_j}\right)*\mathrm{RM_{ij}}\right) \tag{2}$$

Table 3 shows the risk exposure values of print_tokens. For example, the first requirement of print_tokens, Req_1, as seen in **Table 3**, correlates with Input problem, Output problem, Startup/Shutdown, Error Handing among the software related risk groups. The risk exposure values of risk items are calculated by using the Equation (2), and the risk exposure values of input problem risk are $(82*1)+(68*1)+\cdots+(14*0)=150$, when calculated based on the Equation (2).

Table 3. Example of risk exposure values of print_tokens risk items.

Risk item r_i		Requirements				RE (r_i)
		Req_1	Req_2	⋯	Req_{18}	
		0.080	0.073		0.0412	
		0.102	0.094		0.0340	
		82	68	⋯	14	
Software	Input problem	1	1	⋯	0	150
	Output problem	1	1	⋯	0	182
	Calculations	0	0	⋯	1	191
	Startup/Shutdown	1	1	⋯	0	196
	Error handing	1	1	⋯	0	228
	Interactions	0	0	⋯	0	365
	File install	0	0	⋯	0	0
	File collision	0	0	⋯	0	0
⋯	⋯	⋯	⋯	⋯	⋯	⋯

3.3. Test Case Prioritization Evaluation

Test Case Priority (TCP) is estimated by using the risk exposure values of risk items. The criteria of TCP measurement in this paper considers that how many fault can be detected by test case with how much each fault is severe. Indicator that indicates how many faults can be detected by test case is evaluated by how many times the area related to risk item is executed by test case. Also, risk exposure value of risk item is used for the indicator that represents how severe the detected fault is. In other words, in this paper we measure TCP by adding the product of the number of risk items executed by test case and the risk exposure value of risk item in terms of each risk item.

Table 4 is the metric of the evaluation of test case priority. TM_{ji} indicates the number of occurrences of the risk item j when test case i (TC_i) was executed. TCP is the metric representing the number of risk items executed by test case. The ith test case priority value, TCP_i, is as follows:

$$TCP_i = \sum_{j=1}^{m} \left(RE\left(r_j\right) * TM_{ji} \right) \quad (3)$$

Test case priority is decided in order of test cases having the highest TCP values. For instance, **Table 5** is part of the example of a test case prioritization evaluation for print_tokens.

The total number of test cases in **Table 5** is 4130. Test cases are arranged in order of having the highest TCP values. TC1 reflects the number of executed risk items, namely, 4 output problem related risks, 7 calculation risks, 3 startup/shutdown risks, 5 error handing risks and 50 interaction risks, and thus TCP_1, based on the Equation (3), accounts for

$$(150*0)+(182*4)+(191*7)+(196*3)+(228*9)$$
$$+(265*50)+\cdots = 25477 .$$

4. Empirical Studies

4.1. Subjects and Goal of the Experiment

4.1.1. Subjects

Siemens programs [17], selected as the subjects of our empirical study, are C programs developed to study fault detecting effectiveness of coverage criteria. As seen in **Table 6**, the Siemens programs consist of 7 C programs. The programs are widely used as a subject in the comparative experiments for test coverage and test case prioritization technique evaluation.

Table 4. TCP value metric.

Risk item	RE (ri)	Test case				
		TC_1	⋯	TC_i	⋯	TC_n
r_1	RE (r_1)	TM_{11}	⋯	TM_{1j}	⋯	TM_{1n}
		⋯	⋯	⋯	⋯	⋯
r_j	RE (r_j)	TM_{j1}	⋯	TM_{ji}	⋯	TM_{jn}
		⋯	⋯	⋯	⋯	⋯
r_m	RE (r_m)	TM_{m1}	⋯	TM_{mj}	⋯	TM_{mn}
TCP		TCP_1	⋯	TCP_i	⋯	TCP_n

Table 5. An example of test case prioritization evaluation.

Risk item		RE (r_i)	Test case			
			TC_1	TC_2	⋯	TC_{4130}
Software	Input	150	0	0	⋯	1
	Output	182	4	2	⋯	1
	Calculations	191	7	4	⋯	0
	Startup/Shutdown	196	3	3	⋯	1
	Error handling	228	9	2	⋯	1
	Interactions	365	50	16	⋯	0
	File install	0	0	0	⋯	0
	File collision	0	0	0	⋯	0
⋯	⋯	⋯	⋯	⋯	⋯	⋯
TCP			25,477	8494	⋯	⋯

Each program has an original version and a faulty version. Each faulty version was designed to precisely estimate how many specific faults are detected in test cases. **Table 6** reflects the number of faulty versions of Siemens programs, the number of functions possessed by each program and the number of test cases.

4.1.2. Experiment Goal

The followings are test case prioritization techniques used as the comparative subjects including the test case prioritization technique that we propose.

- No prioritization: Select according to the order of generating test cases.
- RE (r_i): The technique this paper proposes.
- Safety Tests: Select in order of test cases with the highest risk exposure values.
- Optimal prioritization: Select test cases that can increase fault detection rates.
- FEP-Total: Select in order of having high fault likelihood through mutation analysis.

Usually, the purposes of testing are to detect serious faults as early as possible, fix faults before product release and find faults as many as possible. In short, how swiftly faults are detected and how fast the locations of severe faults are identified is one of the most critical purposes of testing. We want to present our experiment results in two aspects to demonstrate the effectiveness of our proposed method in terms of fault detection.

- Fault detection rate based on "Frequency of Fault Detected" and "Average Percentage of Fault Detected".
- Severity of Fault.

First, we analyze experiment results on faults detected when testing in the order of test cases generated by test case prioritization technique. To do that, "Frequency of Fault Detected" in the execution order when executed according to test case priority is estimated and then "Average Percentage of Fault Detected" (APFD) [18] is measured to be compared with other techniques. If the number of faults included in the program to be tested is *m*, the number of the total test cases is *n*, and the first test case to expose fault i in the test case pool is TF_i, then

$$APFD = 1 - \frac{\sum_{i=i}^{m} TF_i}{n*m} + \frac{1}{2n} \quad (4)$$

Second, we analyze our experiment results in terms of severity of faults detected. Fault types may vary ranging from severe faults leading to system shutdown or function halt to faults that cause just slowdown of the system. The faults by program were categorized according to six fault types suggested by [19].

If test cases are executed in an order that can detect severe faults earlier than less severe faults, when testing in a given period, the test would be significantly efficient. This paper employs AHP approach to estimate the fault

Table 6. Subject programs.

Programs	# of version	# of fault	Fault severity						# of function	# of test case	Description
			1	2	3	4	5	6			
print_tokens	7	10	1	0	3	1	1	4	18	4130	Lexical analyzer
print_tokens2	10	10	0	3	1	3	3	0	19	4115	Lexical analyzer
Schedule	9	10	0	2	2	2	2	2	18	2650	Priority scheduler
Schedule2	10	10	0	1	0	6	1	2	16	2710	Priority scheduler
Tot_info	23	23	8	1	8	1	1	4	7	1052	Information measure
Replace	32	36	1	9	1	9	10	5	21	5542	Pattern replacement
Tcas	41	54	14	6	14	13	1	4	9	1608	Altitude separation

severity and calculate the "Average Severity of Faults Detected, ASFD" by using Equation (4).

4.1.3. Experiment Method

Applying test case prioritization techniques means to select test case pools that can effectively detect unknown faults. To facilitate this experiment, we established experiment designs to diversely modify known fault information.

For instance, the number of faulty versions of print_ tokens, seen in **Table 7**, is 7. We evaluate test case priority by using fault information on two faulty versions 1 and 2 and estimate whether faults of the remaining 5 faulty versions are detected.

We randomly selected two as known faulty versions out of 7 faulty versions to establish the total 7 sets. We created 7 sets as gradually expanding the number of faulty versions. In the case of print_tokens, a total of 28 experiment sets were created. We also established experiment sets for the rest Siemens programs and conducted experiments relative to 947 sets, as suggested in **Table 7**.

4.2. Experiment Result and Analysis

4.2.1. Frequency of Fault Detection—Status of Number of Faults Detected in the Order of Test Cases with the Highest Priority

If the number of faults detected in the first test suite is high, in order to evaluate whether test cases detect many faults as fast as possible, it indicates the testing has a high effectiveness.

Figure 2 indicates the number of faults detected according to the test case execution sequence by each program by using each test prioritization technique on 947 experiment sets.

The X axis represents test cases in order of having the highest test case priority, while Y axis indicates the number of faults detected by test cases. Since the total number of test cases by each program was 1052~5542, we drew

Table 7. Experiment sets.

Program	# of versions	# of experiment sets
Print_tokens	7	28
Print_tokens2	10	7
Schedule	9	54
Schedule2	10	70
Tot_info	23	460
Replace	32	110
Tcas	41	155

the graphs by dividing the test cases by 100.

The first graph in **Figure 2** represents the mean value of faults detected according to the sequence of test case execution in the entire Siemens programs by each test case prioritization technique. Our proposed method RE (r_i) is represented with a thick solid line, which suggests this method outperforms the other techniques. Especially when we see the first graph, test cases having the higher test case priority of RE (r_i) detect more fault than other technique and the number of fault detected by test cases that are executed later due to lower test case priority decrease.

4.2.2. Average Percentage of Fault Detected

For each subject program, we applied prioritization techniques to each of the 947 test suits. **Table 8** depicts the APFD values of prioritized test cases by each technique and by programs. **Figure 3** is a diagram indicating the fault detection rate according to the test progress. The X axis indicates the execution rates of test cases and Y axis represents rates of fault detection. RE (r_i) is the thick solid line. RE (r_i) technique shows a lower APFD than optimal prioritization. However, when compared to Safety Tests, FET-total, which evaluate test case priority by using fault information, RE (r_i) technique exhibits relatively good fault detection rates even if it only employs exposure values of risk category without using fault information.

$$\text{ASDF}=\frac{\text{sum of undetcted severity}-\text{sum of detected fault severity}}{\text{sum of undetected fualt sevirity}} \qquad (5)$$

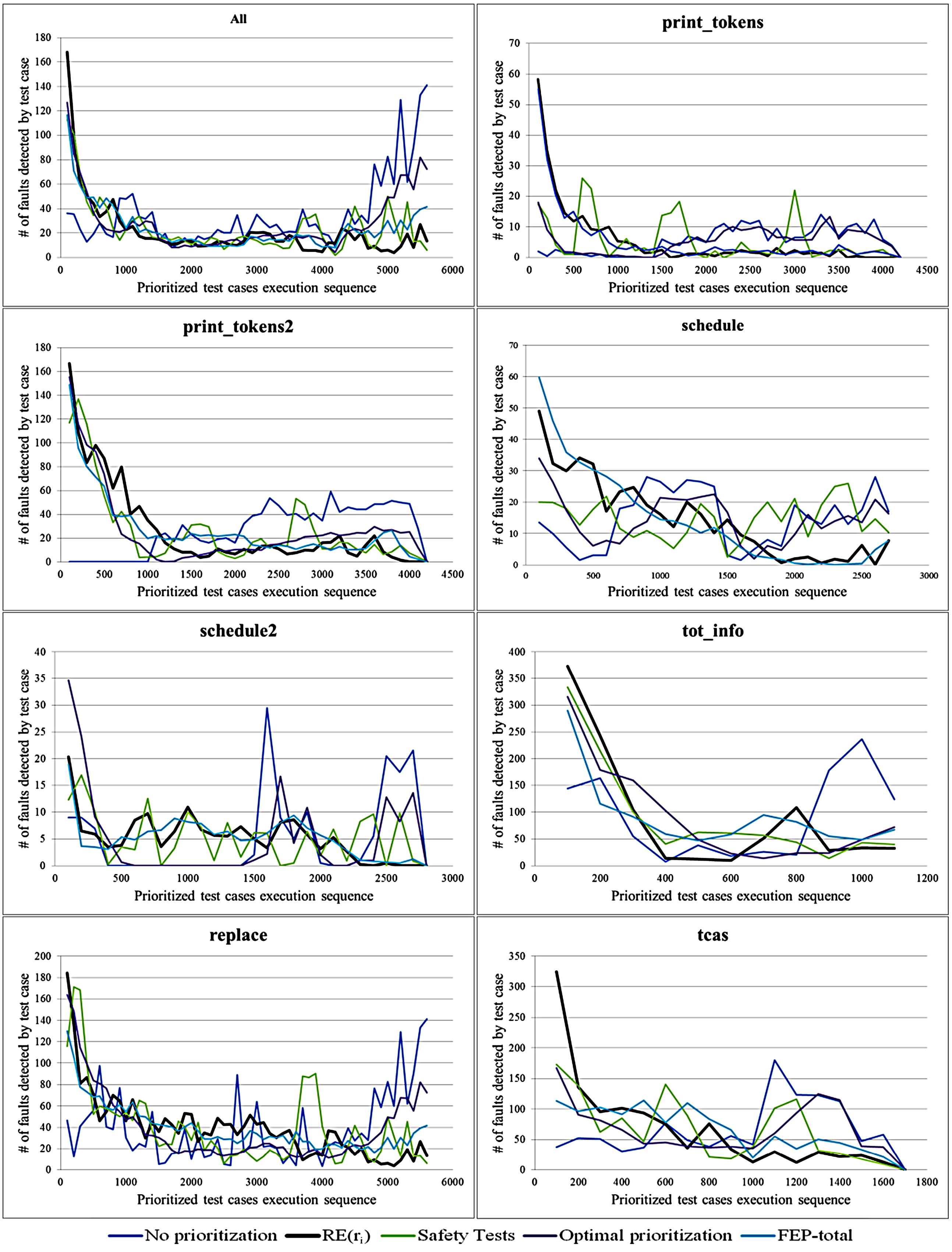

Figure 2. Number of fault detected according to test case execution sequence.

Table 8. APFD (%), data of all in Figure 3.

Programs	No prioritization	RE (r_i)	Safety tests	Optimal prioritization	FET-total
Print_tokens	76	99	99	99	91
Print_tokens2	57	99	99	99	99
Schedule	56	98	86	99	99
Schedule2	30	77	63	90	75
Tot_info	94	95	95	92	93
Tcas	90	88	87	96	81
Replace	89	89	89	89	90
Average	70	92	88	95	90

Figure 4 presents box-plots of the APFD values of the five categories of prioritized test set for each program and an all program total. X axis indicates the test case prioritization techniques and Y axis is fault detection rates.

Table 9 depicts the detailed data of box-plots of the APFD. The scope of fault detection rates of test cases by the upper 30% level is indicated in the dark box, by upper 50% in the lighter box, and by the upper 70% in the white box. The below the vertical line of the inside box-plot represents the minimum fault detection rate when a one test case is run, while the above the vertical line indicates the maximum fault detection rate. For instance, the first graph shows the total average of APFD for all programs.

The minimum fault detection rate of RE (r_i) is high with 19%, following the optimal prioritization technique. The test cases in the upper 30% level indicate RE (r_i) has a significantly high detection rate, which means the high probability of fault detection rates in the early stage of test execution.

4.2.3. Severity of Faulty

Figure 5 is a graph indicating how many severe faults are detected based on the Equation (5) of Chapter 4.1. X axis indicates the prioritized test case and Y axis represents the severity of faults not detected by test cases. The slope of fault severity indicates how many severe faults remain, which means the lower of the slope, the less of the severity of faults remains.

The first graph in **Figure 5** indicates the severity of the entire Siemens programs. RE (r_i) is represented with the thick solid line, which shows a lower slope than other techniques. This means that RE (r_i) can detect serious faults in the early stage of test execution. Moreover, the graphs of RE (r_i) exhibit consistent degrees of slopes for all programs. Thus, the graph analysis of RE (r_i) can allow testers to roughly predict the time to terminate the testing.

4.3. Threats to Validity

Risk is largely divided into Project Risk, Process Risk and Product Risk. We evaluated test case priority by using the risk exposure values of risk items for product risks. Thus, we made the evaluation under the assumption that there exist no project or process risks associated with test cases.

The source size of the Siemens programs which are the subject of our empirical studies is small. In this sense, we need to conduct future works to validate whether the experiment results can be held in other experimental situations including programs which are larger than the source size of the Siemens program or other types of programs. Nevertheless, our experiment could have relative objectivity in terms of evaluating number of fault detected, fault detection rate and the severity of faults since Siemens programs have a set of faulty versions. Moreover, the programs could be advantageous in that they also could be used as a subject in other priority techniques to be compared.

5. Conclusions

We developed a technique to prioritize test cases by employing risk exposure values calculated in each requirement and described the proposed prioritization technique based on comparative analysis between ours and several other existing methods. The characteristics of our method are as follows.

First, our method does not require the pre-executed test results, unlike other existing techniques. Instead, we develop and use a metric for risk item evaluation. This method is feasible to be conducted without the previous test execution results and thus it is expected to have a wide range of applications. In addition, we specifically defined product risk items and it is expected to be useful for risk identification process.

Second, we presented an empirical study comparing the effectiveness of our approach with other prioritization approaches. Our empirical study shows our prioritization technique using risk exposure is promising in terms of effectives in detecting severe faults and benefits in terms of time and cost efficiency.

The risk-based test approach we propose somewhat focus on the functional testing. We plan to expand our study on test case prioritization technique by employing

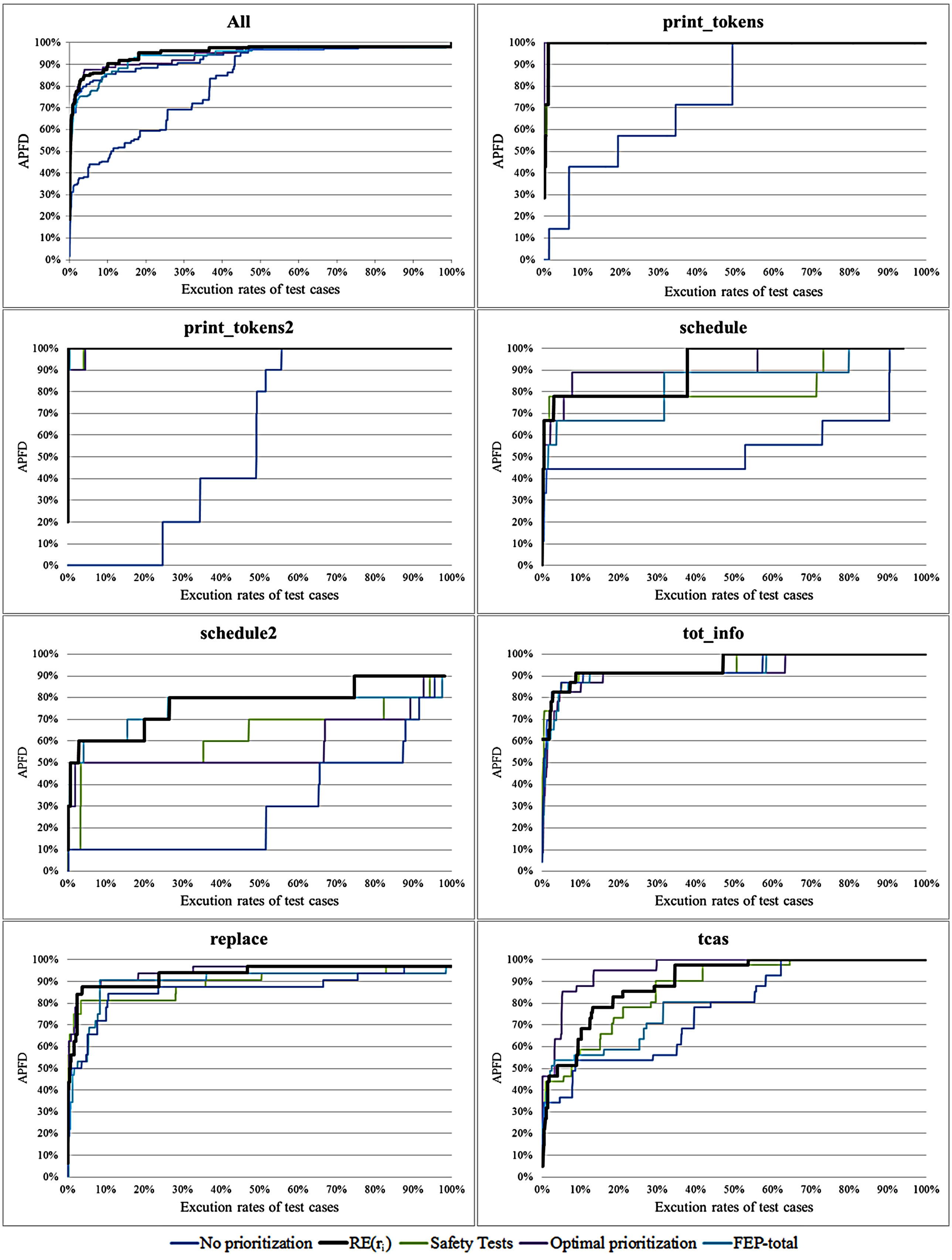

Figure 3. APFD.

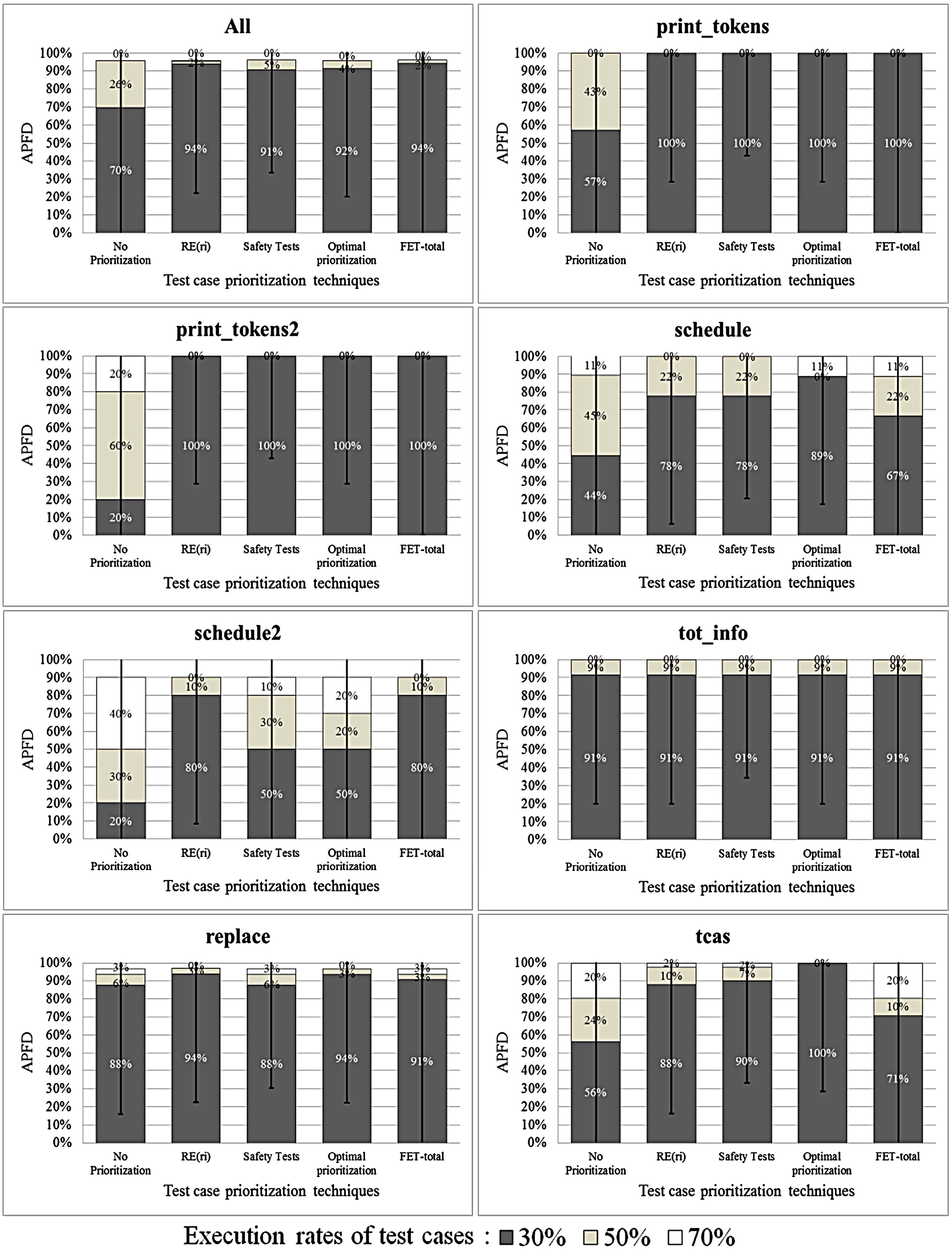

Figure 4. APFD box-plox (vertical axis is APFD score).

Table 9. Data of all in Figure 4.

	No prioritization	RE (r_i)	Safety tests	Optimal prioritization	FET-total
Max.	0.97	0.97	0.97	0.98	0.97
70%	0.96	0.96	0.96	0.96	0.96
50%	0.96	0.95	0.96	0.96	0.96
30%	0.70	0.94	0.91	0.94	0.92
Min.	0.01	0.19	0.17	0.20	0.16

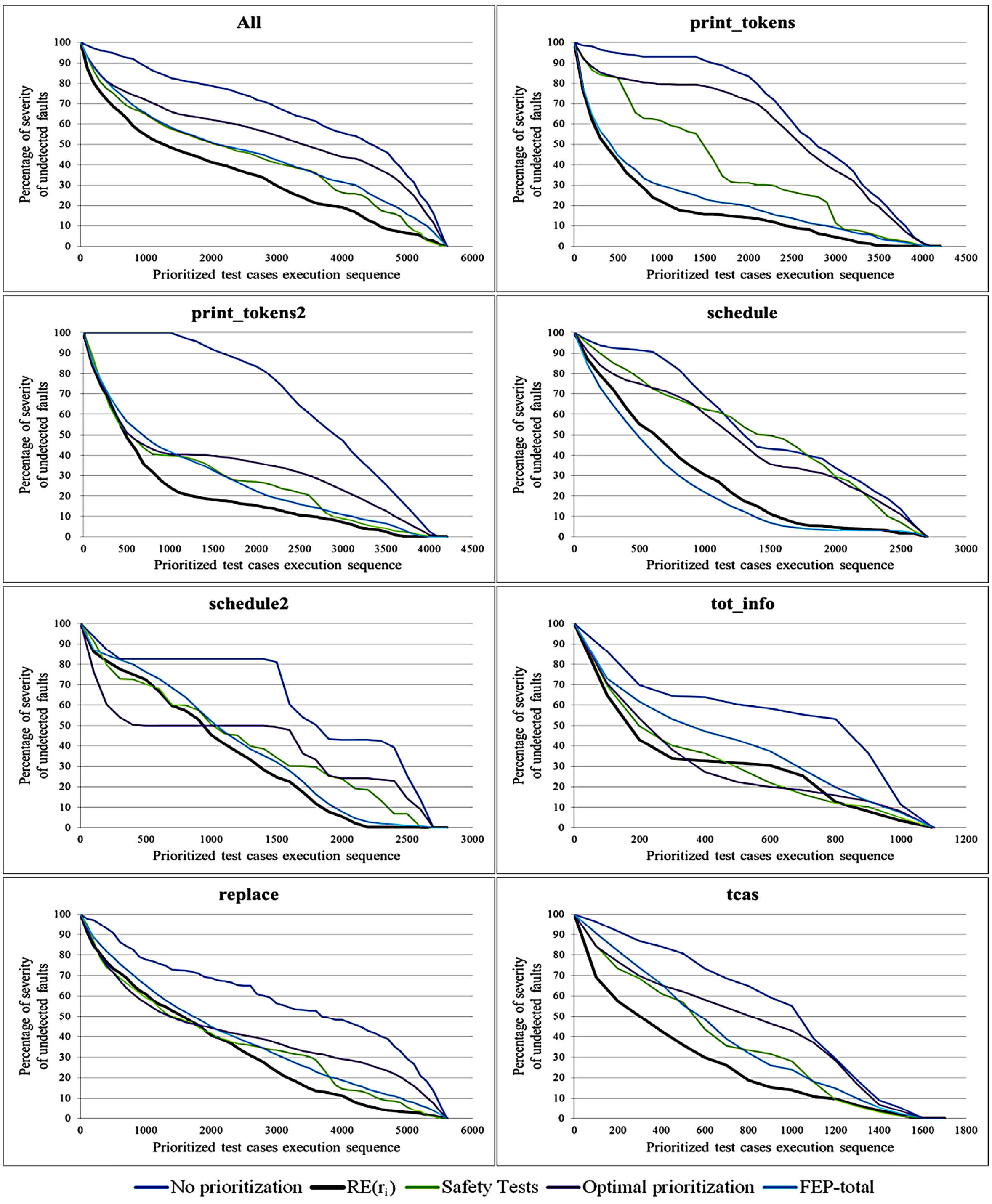

Figure 5. Severity of fault.

risk metric of performance features.

6. Acknowledgements

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency (NIPA-2012-(H0301-12-3004)).

REFERENCES

[1] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, "Prioritizing Test Cases for Regression Testing," *IEEE Transactions Software Engineering*, Vol. 27, No. 10, 2001, pp. 929-948. doi:10.1109/32.962562

[2] R. Krishnamoorthi and S. A. Mary, "Factor Oriented Requirement Coverage Based System Test Case Prioritization of New and Regression Test Cases," *Information and Software Technology*, Vol. 51, No. 4, 2009, pp. 799-808. doi:10.1016/j.infsof.2008.08.007

[3] S. Yoo and M. Harman, "Regression Testing Minimization, Selection and Prioritization: A Survey," *Software Testing*, *Verification and Reliability*, Vol. 22, No. 9, 2012, pp. 67-120. doi:10.1002/stvr.430

[4] H. Do and G. Rothermel, "On the Use of Mutation Faults in Empirical Assessments of Test Case Prioritization Techniques," *IEEE Transactions on Software Engineering*, Vol. 32, No. 9, 2006, pp. 733-752. doi:10.1109/TSE.2006.92

[5] H. Stallbaum, A. Metzger and K. Pohl, "An Automated Technique for Risk-Based Test Case Generation and Prioritization," *Proceedings of the* 3rd *International Workshop on Automation of Software Test*, New York, 11 May 2008, pp. 67-70. doi:10.1145/1370042.1370057

[6] Y. Chen, R. Probert and D. P. Sims, "Specification-Based Regression Test Selection with Risk analysis," *Proceedings of the* 2002 *Conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, 30 September-3 October 2002, pp. 1-14.

[7] B. W. Boehm, "Software Risk Management: Principles and Practices," *Software*, Vol. 8, No. 1, 1991, pp. 32-41. doi:10.1109/52.62930

[8] P. Gerrad and N. Thompson, "Risk-Based E-Business Testing," Artech House, Norwood, 2002.

[9] Institute of Electrical and Electronics Engineers, IEEE. Std. 1044.1-1995, "IEEE Guide to Classification for Soft-Ware Anomalies," 5 August 1996.

[10] D. R. Wallace and D. R. Kuhn, "Failure Modes in Medical Device Software: An Analysis of 15 Years of Recall Data," *Reliability*, *Quality and Safety Engineering*, Vol. 8, No. 4, 2001, pp. 301-311.

[11] M. Sullivan and R. Chillarege, "A Comparison of Software Defects in Database Management Systems and Operating Systems," *Digest of Papers FTCS*-22, *The* 22*nd International Symposium on Fault Tolerant Computing*, Boston, 8-10 July 1992, pp. 475-484.

[12] A. Jha, "A Risk Catalog for Mobile Applications Computer Sciences," Florida Institute of Technology, Melbourne, 2007.

[13] J. Bach, "Heuristic Risk-Based Testing," *Software Testing and Quality Engineering Magazine*, Vol. 1, No. 6, 1999, pp. 96-98.

[14] S. Pertet and P. Narasimhan, "Causes of Failure in Web Applications," Parallel Data Laboratory, Carnegie Mellon University, 2005.

[15] T. L. Saaty, "How to Make a Decision: The Analytic Hierarchy Process," *European Journal of Operational Research*, Vol. 24, No. 6, 1990, pp. 9-26. doi:10.1016/0377-2217(90)90057-I

[16] S. Amland, "Risk-Based Testing: Risk Analysis Fundamentals and Metrics for Software Testing Including a Financial Application Case Study," *Journal of Systems and Software*, Vol. 53, No. 3, 2000, pp. 287-295. doi:10.1016/S0164-1212(00)00019-4

[17] "Siemens Program," 2012. http://pleuma.cc.gatech.edu/aristotle/Tools/subjects/

[18] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies," *IEEE Transactions Software Engineering*, Vol. 28, No. 2, 2002, pp. 159-182. doi:10.1109/32.988497

[19] D. Jeffrey and N. Gupta, "Test Case Prioritization Using Relevant Slices," *Proceedings of the* 30*th Annual International Computer Software and Applications Conference*, Chicago, 17-21 September 2006, pp. 411-420.

Appendix

Comparison between the proposed risk items and referred risk items.

Risk Items			IEEE. Std 1044.1-1995	Wallce, D. R.	Sullivan, M.	Jha, A. K.	Bach, J.	Pertet, S.
1. Software	1-1	Input Problem	**[Symptoms]** -Input Problem	-Input		-User Interface -System Interface		
	1-2	Output Problem	**[Symptoms]** -Output Problem -Spelling/Grammar -Cosmetic	-Display -Output				
	1-3	Calculations	**[Type]** -Computational Problem	-Function		-Calcualations		-Computational/ Logic errors
	1-4	Startup/Shutdown	**[Symptoms]** -Operating System Crash -Program Hung-Up			-Startup/Shutdown		
	1-5	Error Handling	**[Symptoms]** -System Error Message		**[Error Type]** -Unknown	-Error Handling		
	1-6	Interactions	**[Type]** -Logical Problem		**[Error Type]** -Statemnet Logic -Synchroniztion -Undefined State -Wrong Algorithm **[Software Defect Type]** -Function Defect -Data Structure/Algorithm	-Interaction		
	1-7	File Install					-Wrong files installed -Screen saver disrupts install -No detection of incompatible apps	-Failed Software upgrade
	1-8	File Collision			**[Error Type]** -Copying Overrun **[Software Defect Type]** -Build/Package/Merge Defect	-NonExcutable Files	-Files clobbered -Installer silently replaces or modifies critical files or parameters	
2. Data	2-1	Data Handling	**[Type]** -Data Handlin Problem	-Data	**[Error Type]** -Data Error -Pointer Management	-Big and Little(Data Handling)		
	2-2	Data	**[Type]** -Data Problem	-Data	**[Error Type]** -Uninitialized Variable	-Input -Output -Noise		
3. Interface	3-1	Timing	**[Type]** -Interface/Timing Problem	-Timing	**[Error Type]** -Allocation Management -Interface Error **[Software Defect Type]** -Interface Defect -Timing/Synchronization Defect	-Interfaces -Input/Output		
	3-2	SW Interface	**[Symptoms]** -Program Crash			-Software Interface		
	3-3	HW Interface				-Hardware Interface		
	3-4	Wireless Synchronization				-Wireless Synchronization		
4. Enhancement	4-1	Chage in Requirements	**[Type]** -Enhancemnet					
5. Platform	5-1	External HW		-Response		-External Hardware	-Hardware not properly configured	
	5-2	External SW		-Sevice		-External Software	-Other apps clobbered	
	5-3	Internal Components	**[Type]** -Interperability Problem			-Internal Components		
	5-4	Environment				-Environment		
	5-5	Common Use				-Common Use		
Exculding Items			**[Symptoms]** -Incomplete/Missing -Failed Required -Perceived Total Product Failure -Other **[Type]** -Documentation Problem -Unachievable Item -Document Quality Problem Other	-Behavior -Generla -Quality -System -User instruction	**[Error Type]** -Memory Leak -Other	-Code -Hardware	-Install process is too slow -Install precess requires constant user monitoring -Install process is confusing	-Resouce exhastion -Recovery code