Defect Data Analysis Based on Extended Association Rule Mining

Shuji Morisaki, Akito Monden, Tomoko Matsumura,
Haruaki Tamada and Ken-ichi Matsumoto
Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara 630-0192, Japan

Abstract

This paper describes an empirical study to reveal rules associated with defect correction effort. We defined defect correction effort as a quantitative (ratio scale) variable, and extended conventional (nominal scale based) association rule mining to directly handle such quantitative variables. An extended rule describes the statistical characteristic of a ratio or interval scale variable in the consequent part of the rule by its mean value and standard deviation so that conditions producing distinctive statistics can be discovered. As an analysis target, we collected various attributes of about 1,200 defects found in a typical medium-scale, multi-vendor (distance development) information system development project in Japan. Our findings based on extracted rules include: (1)Defects detected in coding/unit testing were easily corrected (less than 7% of mean effort) when they are related to data output or validation of input data. (2)Nevertheless, they sometimes required much more effort (lift of standard deviation was 5.845) in case of low reproducibility, (3)Defects introduced in coding/unit testing often required large correction effort (mean was 12.596 staff-hours and standard deviation was 25.716) when they were related to data handing. From these findings, we confirmed that we need to pay attention to types of defects having large mean effort as well as those having large standard deviation of effort since such defects sometimes cause excess effort.

1. Introduction

Software defects such as bugs, specification changes, and design changes, are major sources of excessive cost (effort) and delivery slippage of a software project. In order to identify any principles, patterns and conditions associated with effort required to correct defects, this paper focuses on association rule mining [1].

So far, association rule mining has been used to discover rules hidden amongst software engineering data. For example, Amasaki et al. [2] mined preconditions (combinations of risk assessment values) for software projects to fall into disorder using a dataset consisting of a large number of risk assessment variables. Song et al. [8] identified rules related to defect association (types of defects occur with others). Song et al. also identified rules related to four categories of defect correction effort (1 hour or less, 1 hour to 1 day, 1 day to 3 days, and more than 3 days).

There is, however, a serious limitation in association rule mining when applying it to software engineering data repositories. That is, it cannot directly handle quantitative (ratio scale or interval scale) variables such as size, effort and duration that are commonly recorded in the repositories. To apply association rule mining to such repositories, we need to translate ratio and interval scale variables into nominal or ordinal ones beforehand just as Song et al. did [8]; however this causes great loss of information of original variables, such as variance, mean and median of values.

In this paper we propose an extended association rule mining method that takes advantage of interval and ratio scale variables, instead of simply replacing them into nominal or ordinal variables. In the proposed method, an extended rule describes the statistical characteristic of quantitative variables (e.g. mean and standard deviation) in the consequent part together with related metrics (e.g. "lift of mean" and "lift of standard deviation") so that conditions producing distinctive statistics can be discovered as rules. For example, we could discover a condition associated with greater defect correction efforts by focusing on rules having large mean effort in consequent parts. Similarly, conditions associated with large variance of effort could be also discovered. Such rules are expected to contribute to process improvement by making a plan to avoid falling into the conditions specified in the antecedent part of the rules.

Based on the proposed mining technique, this paper describes an empirical study to reveal rules associated with defect correction effort. This paper targets a distance development (multi-vendor development), which is most typical type of information system development in Japan today. Specifically, the target was an information system development project consisting of about 330K lines of C/C++ source code,

Table 1 An Example of Defect Data Corrected via An Issue Tracking System

ID	Description	Priority	Status	Assigned to	Reproducibility	Detected date	Closed date	…
001	Authorization is failed after changing password	High	Resolved	Jane Smith	Always	06/12/21	07/1/10	…
002	Response timed out when huge request is received from some web browsers.	Low	Confirmed	John Smith	Seldom	06/12/21	(Not yet)	…
003	Debug message appears when empty message is received.	Middle	Resolved	John Smith	Seldom	06/12/10	06/12/11	…
...	…	…	…	…	…	…	…	…

carried out with the support of Japan's Ministry of Economy, Trade and Industry (METI). In this project, several leading Japanese software companies carried out development, while Nara Institute of Science and Technology's EASE (Empirical Approach to Software Engineering) Project and Japan's Software Engineering Center (SEC) of Information-Technology Promotion Agency collaboratively created a data-collection scheme. Development was carried out using a waterfall process; and various attributes (metrics) of about 1,200 defects found during the development were collected over a six-month period. In the development, a user company defined requirements, and development companies each developed subsystems under the supervision of a project management company. After each company conducted intra-company unit testing and integration testing, inter-company integration testing was performed, followed by inter-company system testing.

In Section 2, we describe conventional association analysis and the issues arise when applying it to defect data. In Section 3 we propose the extended rule mining method. Section 4 describes an empirical study to identify extended rules. Section 5 presents related research. Section 6 summarizes our findings.

2. Association Analysis and Its Issues

2.1 Association Analysis

Researchers have used association analysis to discover associations hidden amongst data in the POS product-purchasing logs of retail stores [1], access logs of website [9], proteins [7], and the like. For example, in the case of POS logs, researchers have mined rules about products purchased together, such as "purchases product A $\wedge$ purchases product B $\Rightarrow$ purchases product C." There are a number of possible uses for the rule in this example: the retailer could place products A, B, and C near to each other in the store so that customers can find them easily; or, it could ensure revenues by setting the prices of antecedent products A and B to make up the discounts on the sale price of consequent product C.

Association analysis is defined as follows [1].

Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of binary attribute values, called items. A set $A \subset I$ is called an item set. Let a database D be a multi-set of I. Each $T \in D$ is called a transaction. An association rule is denoted by an expression $A \Rightarrow B$, where $B = I_k (1 \le k \le m)$, $A \cap B = \phi$

With data like POS logs, however, which have huge numbers of items, it is not realistic to mine all rules: it takes inordinate amounts of computer processing time, and it is not feasible to interpret the huge number of mined rules manually. For this reason, conditions are placed on rule mining, setting minimum values for one or all of three key indicators of rule importance (support, confidence, and lift). Rules that are not likely to be important are generally pruned.

Support:

Support is an indicator of rule frequency. It is expressed as $support(A \Rightarrow B)$, and is $support(A \Rightarrow B) = a/n$, where $a = \left|\{T \in D \mid A \subset T \wedge B \subset T\}\right|$ and $n = \left|\{T \in D\}\right|$.

Confidence:

Confidence is the probability that consequent B will follow antecedent A. It is expressed as $confidence(A \Rightarrow B)$, and is $confidence(A \Rightarrow B) = a/b$, where a is defined as in Support and $b = \left|\{T \in D \mid A \subset T\}\right|$.

Lift:
Lift is an indicator of the contribution antecedent A makes to consequent B. It is expressed as $lift(A \Rightarrow B)$, and is $lift(A \Rightarrow B) = confidence(A \Rightarrow B)/c$, where $c = \left|\{T \in D \mid B \subset T\}\right|$.

For example, assume that the number of defects n = 20, the number of defects that contain A is 10, the number of defects that contain B is 8, and the number of defects that contains both A and B is 6. For $A \Rightarrow B$, the support is 0.3 (6/20), the confidence is 0.6 (6/10), and the lift is 1.5 (0.6/8/20).

2.2 Issues with Association Analysis for Defect Data

This paper envisions collecting defect data as a project progresses, and assumes that defect attributes, such as staff effort required to correct a defect, which are collected via a common issue tracking tool. Table 1 shows an example of defect data. In Table 1, each row corresponds to a single defect found in a software development project. Many attribute values are measured and logged for each defect. As shown in Table 1, a major characteristic of defect data is the existence of nominal scale variables such as description of a defect's description, priority to fix a defect and its status, as well as interval and ratio scale variables such as defect correction effort and duration (between detected date and corrected date.)

Association analysis normally is applied to qualitative variables (nominal or ordinal scale variables); interval and ratio scale variables are generally converted into ordinal scale variables before applying association rule mining. For example, it would be possible to convert the correction effort into an ordinal scale variable consisting of three categories – large, medium, and low – depending on its value, but the optimum partition must be determined via trial and error, and it is a nontrivial task to discover the optimum partition points for multiple variables. Moreover, this scale conversion causes loss of information such as variance, mean and median of original values.

3. Extension of Association Rule Mining

3.1 Preliminary Definitions

Each value in Table 1 is expressed as an <*attribute, value*> pair. Let defects be a set $D = \{D_1, D_2, ..., D_n\}$, and $D_i = \{< attr_1, d_{i1} >, < attr_2, d_{i2} >, ..., < attr_m, d_{im} >\} (1 \le i \le n)$, where $attr_k$ is the kth attribute and d_{ik} corresponds to the value of the kth attribute. Using Table 1 as an example, the second row in the table (the item with ID 0001) is D_1, and D_1 = {<ID, 0001>, <description, "Authorization is failed after changing password">, <priority, high>, <severity, major >,…}. $attr_1$ is ID, d_{11} is "0001".

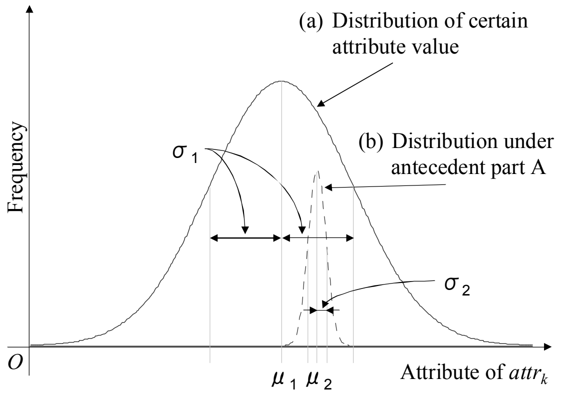
(a) Distribution of certain attribute value
(b) Distribution under antecedent part A
Frequency
σ_1
σ_2
O
μ_1 μ_2
Attribute of $attr_k$

Figure 1 Distributions of attribute value

3.2 Handling Quantitative Variables

The proposed mining method uses the attribute (correction effort or durations), the mean value μ, and the standard deviation σ of a quantitative variable in the consequent part B to create an extended association rule expressed as $A \Rightarrow attr_k(\mu, \sigma)$,

where $\mu = \frac{1}{l}\sum d_{ik} (1 \le i \le l)$, $\sigma = \sqrt{\frac{1}{l}\sum (d_{ik} - \mu)^2} (1 \le i \le l)$, $l = \left|A \subset D\right|$.

The analyst specifies $attr_k$ to be mined according to attributes of defect data. Rules are mined by calculating the mean μ and standard deviation σ of $attr_k$ in defects that meet the antecedent A. An example would be "<severity, major> ⇒ correction effort (9.25, 23.16)."

We define the indicators below (lift of mean and lift of standard deviation) by comparing the mean and standard deviation of all defects.

Lift of mean
The lift of mean is μ divided by the mean of the kth attribute (correction effort) of all defects.

lift of mean $= \frac{\mu}{\frac{\sum d_{ik}}{n}} (1 \le i \le n)$

Lift of standard deviation
Similarly, the lift of standard deviation $= \frac{\sigma}{\sqrt{\frac{\sum (d_{ik} - \mu)^2}{n}}} (1 \le i \le n)$

For example, given a quantitative rule "<detected phase, coding phase> ⇒ correction effort (2.0, 0.864)," if the mean correction effort of all defects is 0.5, then the lift of mean is 2.0 / 0.5 = 4.0. The higher this value, the greater the effect of the antecedent is on the consequent in this rule.

Table 2 Defect Attributes of The Empirical Study

Attribute	Description	Categories
Cause of detection delay	Cause/reason why the defect was not detected in the phases when the defect was introduced or should be detected	Not reviewed Overlooked in the review Overlooked in checking the revised program Insufficient communication Lack of test cases Test cases not executed Testing was carried forward to later phase due to a testing environment Misjudged result of test Others (Left blank)
Correction Effort	Staff hours for correcting effort	N/A (Numeric value)
Corrected phase	The project phase in which a defect was corrected	Coding/unit testing Integration testing System testing (Left Blank)
Detected phase	The project phase in which a defect was detected	
Defect type	Type of defect causes	Logic Computation Interface/timing Data handling Scope of data incorrect Data problem Incorrect problem Accuracy of document Enhancement Performance Interoperability Standards conformance Misoperation Misjudgement (not a defect) Unknown Others (Left blank)
Function type	Function type of a module in which a defect was detected	Validation of input data Computation Data handling File update Data output Linked processes Border processing Sensing external anomalies Others
Introduced phase	The development phase in which the cause of defect was introduced	Architecture design Detail design Coding / unit testing Integration testing System testing (Left blank)
Priority	Priority of correcting a defect	High Medium Low
Project activity	The activity in which a staff reporting a defect was involved	Analysis Testing Review
Reproducibility	How easy to reproduce the failure	Always Sometimes One time occurrence Unknown
Severity	How severe the impact of the failure is on the program operation	High Medium Low

Figure1 shows an example to explain the lift of standard deviation. Solid line (a) is distribution of d_{ik} of all defects ($1 \le i \le n$). Dotted line (b) is distribution of d_{ik} of defects that meets antecedent part A ($A \subset D$). Lift of standard deviation is the ratio of σ_2 to σ_1. In this case, lift of standard deviation smaller than 1 ($\sigma_2 / \sigma_1 < 1$) indicates that situations expressed by the antecedent part A are drivers for smaller deviation. Enhancement of situations expressed by A may lead to smaller deviation of values of k^{th} attribute. For more detailed explanation about our extended rule mining, see our technical report [6].

Table 3 Top 5 Rules Having Large Lift of Mean

ID	Rule	Support	Lift of mean	Lift of Std. Dev.
LLM1-a	(Detected phase = System testing) $\wedge$ (Severity = High) $\wedge$ (Priority = High) $\Rightarrow$ Correction effort (mean: 21.818, std deviation: 15.420)	0.011	10.246	3.444
LLM1-b	(Detected phase = System testing) $\wedge$ (Priority = High) $\Rightarrow$ Correction effort (mean: 21.818, std deviation: 15.420)	0.011	10.246	3.444
LLM1-c	(Detected phase = System testing) $\wedge$ (Severity = High) $\Rightarrow$ Correction effort (mean: 19.231, std deviation: 15.428)	0.013	9.031	3.446
LLM1-d	(Detected phase = System testing) $\wedge$ (Severity = High) $\wedge$ (Defect cause = Coding error) $\Rightarrow$ Correction effort (mean: 19.231, std deviation: 15.428)	0.011	8.539	3.689
LLM2	(Introduced phase =Coding / unit testing) $\wedge$ (Reproducibility = Always) $\wedge$ (Corrected phase = System testing) $\Rightarrow$ Correction effort (mean: 19.231, std deviation: 15.428)	0.011	6.024	3.629

Table 4 Top 3 Rules Having Small Lift of Mean

ID	Rule	Support	Lift of mean	Lift of Std. Dev.
SLM1	(Detected phase = Coding / unit testing) $\wedge$ (Function type = Data output) $\wedge$ (Corrected phase = Coding / unit testing) $\wedge$ (Priority = Low) $\Rightarrow$ Correction effort (mean: 0.100, std deviation: 0.000)	0.013	0.047	0
SLM2	(Detected phase = Coding / unit testing) $\wedge$ (Function type = Data output) $\Rightarrow$ Correction effort (mean: 0.129, std deviation: 0.049)	0.016	0.060	0.011
SLM3	(Detected phase = Coding / unit testing) $\wedge$ (Function type = Checking input data) $\wedge$ (Defect cause = Coding error) $\Rightarrow$ Correction effort (mean: 0.147, std deviation: 0.101)	0.025	0.069	0.023

Table 5 Top 3 Rules Having Large Lift of Standard Deviation

ID	Rule	Support	Lift of mean	Lift of Std. Dev.
LLS1	(Detected phase = Coding / unit testing) $\wedge$ Reproducibility = One time occurrence) $\wedge$ (Corrected phase = Coding / unit testing) $\Rightarrow$ Correction effort (mean: 9.333, std deviation: 26.166)	0.011	4.383	5.845
LLS2	(Introduced phase = Coding / unit testing) $\wedge$ (Defect type = Data handling) $\wedge$ (Defect cause = Coding error) $\Rightarrow$ Correction effort (mean: 12.596, std deviation: 25.716)	0.011	5.915	5.744
LLS3	(Cause of detection delay = Overlooked in the review) $\wedge$ (Reproducibility = One time occurrence) $\wedge$ (Severity = Middle) $\Rightarrow$ Correction effort (mean: 8.109, std deviation: 22.655)	0.013	3.808	5.060

4. Empirical Study

4.1 Data Collection

We analyzed a project to develop a probe-information system carried out by members of the COSE (COnsortium for Software Engineering), with the support of METI. Six companies participated in the development: one company was tasked with project management, and the other five with development.

The defect attributes to be collected were determined through pre-project discussions, referring to failure reports actually used by the participating companies and the classification for software anomalies in IEEE Standard 1044-1993 [5].

The phases of development for which data were collected - that is, the phases where defects were detected - were those from coding and unit testing phase to system testing (in this project, coding and unit tests are considered parts of the same phase). In this project, testing was carried out in the following stages: first, each site conducted the unit testing and integration testing internally; next, inter-company integration and system testing were conducted, integrating the products of all companies in an integrated environment.

Defects introduced after the architecture-design phase were analyzed. For this project, errors and changes to the requirements specification were excluded from analysis, since they are managed by a different management procedure.

Table 2 lists the collected defect attributes used for this paper. Correction effort of each defect was collected in staff hours as a unit (based on staffs' declarations). The number of defects was 1,225. There were also free-description reports, such as descriptions of the symptoms as well as details on a defect's cause, correction and confirmation, but these do not appear in Table 2 because they were not used for the analysis.

4.2 Extracted Rules

About 17,000 rules were mined using a prototype implementation of the proposed method with parameter minimum support 0.01. Mined rules are sorted into Table 3, 4, and 5. Table 3 lists top five rules in descending order of lift of mean. Table 3 includes all resembling rules such as LLM1-a, LLM1-b, LLM1-c and LLM1-d. Table 4 lists top three rules in ascending order of lift of mean excluding resembling rules. Table 5 lists top three rules in descending order of lift of standard deviation excluding resembling rules because of space limitations.

Figure 2 illustrates extracted rules in 2-dimension space (mean and standard deviation of defect correction effort). From Figure 2, characteristics of rules are easily recognizable, i.e. rules LLM1-a,…d and LLM2 are in the large mean effort area; rules LLS1, 2 and 3 are in the large std. dev. effort area; rules SLM1, 2 and 3 are in the small mean effort area.

Rules related to greater effort

Rules LLM1-a, LLM1-b, LLM1-c, and LLM1-d in Table 3 indicate that high severity (or high priority) defects detected in the system testing phase took about 10 times greater correction effort than that of all defects. The support values of LLM1-a, LLM1-b, LLM1-c, and LLM1-d show that 1.1%-1.3% of all defects satisfy the conditions described by the rules.

On the other hand, rule LLM2 indicates that even without the condition "high severity/priority," defects required greater effort (6.024 times greater than the mean) when they were corrected in system testing. Our further analysis revealed that detected phase almost correspond to the corrected phase, i.e. most defects were corrected in their detected phases.

Rules related to smaller effort

Rules SLM1 to SLM3 in Table 4 are all related to defects detected in coding/unit testing. SLM1 and SLM2 indicate that defects related to data output took much less effort (0.047 and 0.06 times as much correction effort as all defects did.) Similarly, SLM3 indicates that defects related to validation of input data took 0.069 times as much effort as that of all defects.

Rules related to greater variance of effort

The rule LLS1 in Table 5 indicates that correction effort of defects satisfying the antecedent ((Detected phase = Coding/unit testing) $\wedge$ (Reproducibility = One time occurrence) $\wedge$ (Corrected phase = Coding/unit testing)) had 5.845 times greater standard deviation than that of all defects. Interestingly, although many defects detected in coding/unit testing took smaller staff effort to correct (as shown by rules SLM1, SLM2, and SLM3), this rule shows that if such defects had low reproducibility, then they sometimes took greater staff effort to correct (as shown by high standard deviation.)

Rule LLS2 indicates that defects introduced in coding/unit testing often required large correction effort if they were related to data handing.

Similarly, LLS3 indicates that if defects were overlooked in the reviews at design phase and they had low reproducibility, they often required greater correction effort.

4.3 Discussion

Firstly, from rules in Table 3, we found that high severity/priority defects detected in system testing required remarkably greater correction effort (about 10 times greater than that of all defects). The number of defects that satisfied the rule conditions was 14 for LLM1-a, b and d, and 16 for LLM1-c. Considering that there were 44 defects found in system testing phase in all, about 30% of them required very large correction effort. Based on the interviews with developers and the questionnaires they completed, we confirmed the following reasons for increased effort in system testing.

A) When the developer found a defect in the system testing phase, s/he had to locate the vendor at which the cause of the defect was introduced. Sometimes this took much time because the engineers were often unfamiliar with other vendors' (sub) system.
B) If the defect was related to more than one vendor's program, it took much time to modify the programs due to the need for excessive communication between vendors in distance.
C) In the system testing phase, correction had to be confirmed by regression testing in an environment where all venders' (sub) systems were integrated. The preparation for regression testing in the system testing phase required much more time than for intra-company unit/integration testing because the developers ask other vendors developers in distance to execute and appear the same situation as the defect was detected by using e-mail or telephone.

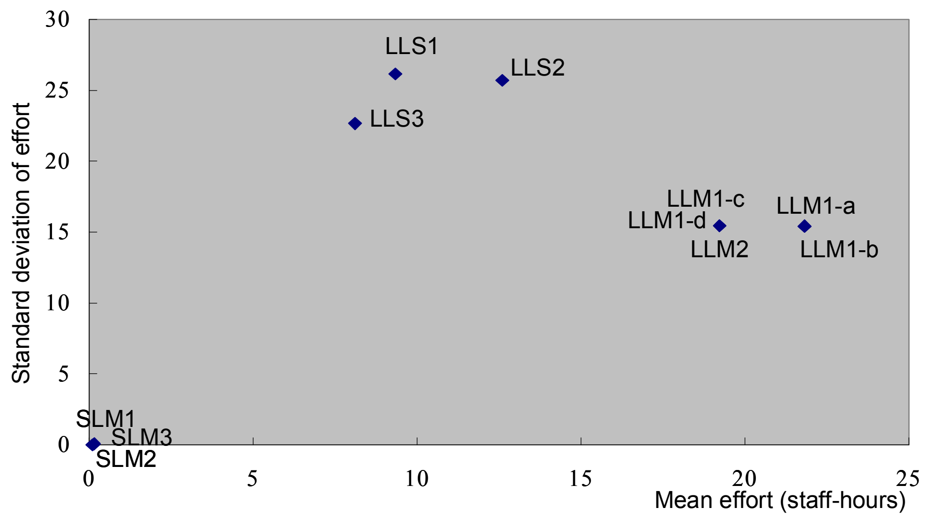

Figure 2 Extracted Rules in 2-dimension Space

Next, from rules in Table 4, we found that defects detected in coding/unit testing (i.e. before integration testing) took much smaller correction effort (less than 7% of mean effort) when they are related to data output or validation of input data functions. Nevertheless, from the rule LLS1 in Table 5, even defects were detected in coding/unit testing, they sometimes required much more effort (lift of standard deviation of effort was 5.845) in the case of low reproducibility.

From the rule LLS2, we also found a remarkable suggestion. Generally, in this project, defects introduced in the coding/unit testing phase required relatively *less* effort than those introduced in the design, integration testing or system testing phase on average. Nevertheless, according to LLS2, defects introduced in coding/unit testing often required large correction effort when they were related to data handing. This suggests defects in data handing function are quite serious ones in this project.

Above all, we have confirmed that paying attention only to the mean effort is insufficient. We also need to be aware of defects that satisfy rule having large standard deviation of effort since such defects potentially cause excessive effort and may trigger the delivery slippage.

5. Related Research

Fukuda et al [4] have proposed a method for handing quantitative variables in association rule mining. This method derives a category from a given quantitative variable by defining an interval in the variable; for example, given a quantitative variable *age*, this method calculates the values x_1, x_2 for which a rule "age interval $[x_1, x_2] \Rightarrow$ purchased a given service X" has the highest support. The article [3] extended this method so that it can handle two quantitative variables. Although this method does not provide mean and standard deviation values, the method seems useful to mine software engineering data repositories since they usually contain quantitative variables. We could use this method in the antecedent part of rules together with our extended association rule mining.

A number of case studies have reported association-analysis methods for software engineering repositories. Amasaki et al [2] evaluated risk items for each development phase based on questionnaires to project managers, and conducted an association analysis to reveal conditions leading to project overrun (excess development budgets or delivery slippage). Their analysis target dataset, however, did not contain any quantitative variables, and rules were mined within the scope of conventional association analysis.

Song et al [8] mined association rules from defect data logged during software development (type of defect cause, correction effort, etc.) to predict types of defects occur with others and to predict defect-correction effort (staff-hours). In their analysis, defect correction effort was converted into four hard-wired categories: one hour or less, one hour to one day, one to three days, and longer than three days. Although their approach can discover some useful rules (e.g. rules related to large or small effort), it cannot discover rules related to large variance of effort. We believe our extended rule mining is useful not only to our dataset but also to their dataset to discover diverse rules.

6. Conclusions

In the former half of this paper, we proposed an extended association rule mining method that takes full advantage of quantitative variables. In the proposed method, an extended rule describes the statistical characteristic of quantitative variables (mean and standard deviation) in the consequent part together with related rule metrics ("lift of mean" and "lift of standard deviation") so that conditions producing distinctive statistics can be discovered as rules.

In the later half of this paper, we presented an empirical study to reveal rules associated with defect correction effort using the defect data collected from a Japanese multi-vendor information system development project. Our findings based on extracted rules include the following:

- High severity/priority defects detected in system testing required remarkably greater correction effort (about 10 times greater than that of all defects).

- Defects detected in coding/unit testing were easily corrected (less than 7% of mean effort) when they are related to data output or validation of input data functions.
- Nevertheless, even defects were detected in coding/unit testing, they sometimes required much more effort (lift of standard deviation of effort was 5.845) in the case of low reproducibility.
- Defects introduced in coding/unit testing required smaller correction effort than average; however, they often required large correction effort (mean was 12.596 staff-hours and standard deviation was 25.716) when they were related to data handing.

Some parts of these findings are generic ones (e.g. defects detected in system testing require greater correction effort), while some other parts are specific to the project we analyzed. Thus, these are especially useful to the succeeding project now being held by the same development organizations. For example, since data handling defects introduced in coding/unit testing are very costly in this project, we recommend adding a source code review process for data handling modules to this project.

From above findings, we have confirmed that paying attention only to the mean effort is insufficient. We also need to be aware of types of defects having large standard deviation of correction effort since such defects potentially cause excessive effort and may trigger the delivery slippage.

Acknowledgements

This work is supported by the Comprehensive Development of e-Society Foundation Software program of the Japanese Ministry of Education, Culture, Sports, Science and Technology.

References

[1] Agrawal R., Imielinski T., Swami A.,: Mining Association Rules between Sets of Items in Large Databases, In Proceedings of ACM SIGMOD Conference on Management of Data, pp. 207-216. (1993)

[2] Amasaki S. , Hamano Y. , Mizuno O. , and Kikuno T. , “Characterization of Runaway Software Projects Using Association Rule Mining,” In Proceedings of 7th International Conference on Product Focused Software Process Improvement, pp.402-407. (2006)

[3] Fukuda T., Morimoto Y., Morishita S., Tokuyama T., : Data Mining Using Two Dimensional Optimized Association Rules: Scheme, Algorithms, and Visualization, In Proceedings of the ACM SIGMOD Conference on Management of Data, pp. 13-23. (1996)

[4] Fukuda T., Morimoto Y., Morishita S., Tokuyama T., : Mining Optimized Association Rules for Numeric Attributes In Proceedings of the 5th ACM SIGACT-SIGMOD SIGART Symposium on Principles of Database Systems, pp. 182-191. (1996)

[5] IEEE Standard 1044-1993 IEEE Standard Classification for Software Anomalies. (1993)

[6] Morisaki, S., Monden, A., Tamada, H., Matsumura, T., and Matsumoto, K.: An Extension of Association Rule Mining for Software Engineering Data Repositories, Information Science Technical Report, NAIST-IS-TR2006008, Graduate School of Information Science, Nara Institute of Science and Technology. (2006)

[7] She R., Chen F., Wang K., Ester M., Gardy J.L., Brinkman F.L.: Frequent-Subsequence-Based Prediction of Outer Membrane Proteins, Proceedings of 9th ACM SIGKDD International conference on Knowledge Discovery and Data Mining, pp. 436-445. (2003)

[8] Song Q. , Shepperd M., Michelle Cartwright, and Carolyn Mair: Software Defect Association Mining and Defect Correction Effort Prediction, IEEE Transaction on Software Engineering, Vol. 32, No. 2, pp. 69-82. (2006)

[9] Yang Q., Zhang H.H., Li T., “Mining Web Logs for Prediction Models in WWW Caching and Prefetching, Proceedings of Seventh ACM SIGKDD International Conference of Knowledge Discovery and Data Mining, pp. 473-478. (2001)