http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (2): 235-249, 2006
ISSN 1812-5638
© 2006 Asian Network for Scientific Information

A Stability-oriented Business Component Refactoring Method Using Bayesian Analysis

Zhongjie Wang, Xiaofei Xu and Dechen Zhan
Research Center of Intelligent Computing of Enterprises,
Harbin Institute of Technology,
P.O. Box 315, No. 92 West Da Zhi Street, Harbin, Heilongjiang, China

Abstract: Reusable components should be continuously refactored during the full lifecycle to modify those designs unsuitable to reuse and to improve reuse performance. In this paper we proposed a stability-oriented business component refactoring method using Bayesian Analysis. Before introduction on this method, a unified feature-oriented component model and its reuse mechanism are briefly proposed with the classification of component reuse styles (modification levels). In this method, clear separation of stability is considered as the goal of refactoring to decrease reuse cost, practical reuse data is utilized as the information source of refactoring and Bayesian analysis method is adopted to calculate posterior distribution and estimation of a set of stability parameters. By analyzing variation tendency of each parameter, component designers can take three basic refactoring operations and ten concrete refactoring rules to reconfigure granularity and instantiation degree of components to realize reuse cost optimization. In order to evaluate performance improvement before and after refactoring, five metrics are addressed. A practical case is compendiously shown to validate the effectiveness of this method, with some qualitative comparisons with other refactoring methods in literatures.

Key words: Component refactoring, reuse cost optimization, stability, Bayesian analysis method, statistical reuse data

INTRODUCTION

Component Refactoring (CR) is defined as the process of continuously re-design a component in its whole reuse lifecycle. The purpose of CR is to reallocate responsibilities among components to provide a better and cleaner component architecture design without the redundancy of data and behavior and without extensive interaction and dependencies among components, while keeping stable the contractually defined collaborative behavior of the set of components, under the guarantee that the business functions components provide are not always destroyed[1].

Actually, CR is a continuous and iterative process. After the component design phase and a component has been reused for some time, it is necessary for designers to re-design and optimize it according to practical reuse situations so as to make the component be better fit for reuse[2]. The process is described in Fig. 1.

At present, CR has been gradually paid more and more attentions to by researchers. CR can be decomposed into two key sub-problems, i.e., (1) how to find the deficiencies in component specification that impact component performance and, (2) how to modify and eliminate these deficiencies.

Aiming at the former, there have been many solutions in literatures. Stojanoviæ presented in his doctoral dissertation a method based on 9-interaction matrix[1], constructing an interaction matrix from Content, Contract and Context for arbitrary two components and by analyzing 9 parameters in this matrix to optimize relationships between the two components. Vitharana *et al.*[3] presented a component design model (BusCod Model) based on business strategies, in which an objective function for component refactoring was addressed by synthesis of several parameters that depict reuse performance and by adjusting values of these parameters, the objective function could achieve optimization. Caballero and Demurjian[4] proposed a formal component refactoring framework, in which the dependencies between components were classified into 8 types according to abstraction level and by transforming those undesired types to good ones, component performance may be improved.

Corresponding Author: Zhongjie Wang, Research Center of Intelligent Computing of Enterprises,
Harbin Institute of Technology, P.O. Box 315, No. 92 West Da Zhi Street,
Harbin, Heilongjiang, China Tel: +86 451 86412664 Fax: +86 451 86412664

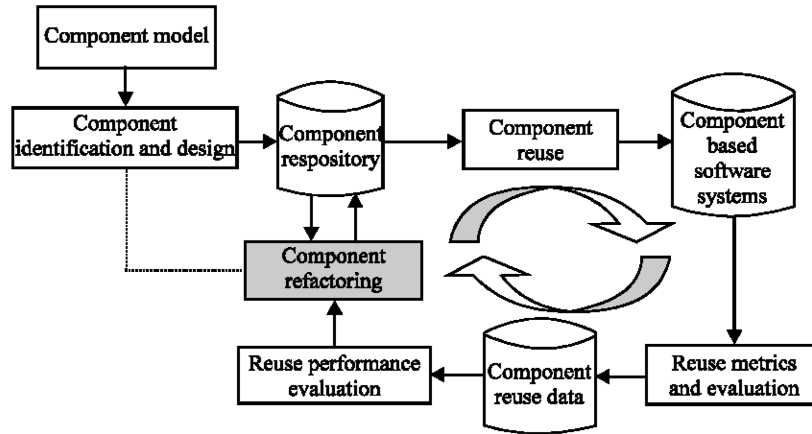

Fig. 1: The full lifecycle of component

Besides these methods above, other methods are also available, such as component refactoring method based on patterns and metamodelling[5,6], Extract Component Refactoring method[6], etc.

These methods realize component optimization by mainly focusing on the following aspects:

- Optimization on the relationships between basic elements in components[7,8];
- Adjusting the inheritance hierarchy between elements[9];
- Adjusting the abstraction level of attributes and methods in components[4].

In these methods, technical metrics of components, such as coupling-cohesion, granularity, total component numbers, component complexity, etc, are adopted as the metrics for performance improvement[3].

However, most of these methods all do not consider practical reuse data as an information source for refactoring and one of their common means for refactoring is to analyze the semantics and structure information of components to find deficiencies in current designs and eliminate them. But in fact it is a paradox, i.e., if these deficiencies really exist, then they could be detected in design phase and eliminated; and it is not necessary to refactor until components have been reused. Therefore, these methods are more considered as design methods rather than refactoring methods.

Our view is that practical reuse data have to be considered as a pivotal information source of refactoring. This is because that those data used for component design are all hypothesized by designers according to requirements and along with reuse process, the original suppositions will usually derivate from actual situations, which thus indicates that the design and practical reuse requirements are inconsistency and component reuse performance does not achieves the optimal status as imagined. Therefore, it is necessary to revise the original design with the aid of actual data, to make it better adaptable to reuse.

In this study, we propose a component refactoring method using Bayesian analysis. This method takes a set of component stability parameters as optimization goals, takes a mass of data produced during reuse period as the basis of refactoring to calculate posterior distribution of various parameters, then, gets the variation tendency of each parameter by comparison between each parameter's posterior and prior value and adopts specific refactoring rules according to this tendency.

SOME BACKGROUNDS

Feature-oriented component model and reuse mechanism: Currently there exist various component semantics models, e.g., 3C[10], Wright[11], JBCOM[12], etc., each of which has their emphasis on describing business semantics contained in components. In order for succinct and clear explanations, we simplify and unify these models to get a unified feature-oriented component model[2], i.e., no matter what kind of heterogeneous component models, they can be uniformly expressed as the form of feature space and most of characteristics of component models can be realized by utilizing advantages of feature modeling[13,14], such as:

1. Functions implemented by components can be uniformly expressed as features f and all features contained in a component C form the feather space $\Omega(C)$ of C.
2. Composition relationships between functions. $\Omega(C)$ could be represented as a feature tree, in which "a

function uses another function" can be expressed as "Parent-Child" features, i.e., if f uses g, then g∈child(f), f∈parent(g)

3. Reuse mechanism. A feature f may contain multiple feature items, denoted as dom(f)={τ_1, τ_2,..., τ_n}, each of which is a variable implementation of f. If f has only one feature item, i.e, |dom(f)|=1, f is called a fixed feature, or the commonality of the component; if f has multiple feature items, i.e., |dom(f)|>1, f is called a variable features, or the variability of the component. We have VARIATION_PART(C)={f | |dom(f)|>1} and FIXED_PART(C)= {f | |dom(f)|=1}.
4. Dependencies between component functions could be expressed as the forms of feature dependency (FD), i.e., X→Y means that all features in Y are dependent on features in X.
5. Component interfaces. A component may have two types of interfaces: PROVIDING and REQUIRED interfaces. A component provides its features to other components or environment via its PROVIDING interfaces and accesses features of other components via REQUIRED interfaces.

Therefore, a component can be denoted as the form of C=<f_{root}, F, FD, PS, RS>, just as illustrated in Fig. 2, where f_{root}, F, FD, PS, RS are the foot feature of C, feature set, feature dependency set, PROVIDING interfaces and REQUIRED interfaces, respectively.

A reusable component is actually an abstract software artifact, which cannot be reused directly and only after each variable features in C has been instantiated as a specific feature item, can C be reused. Therefore, the final reused component is in fact an

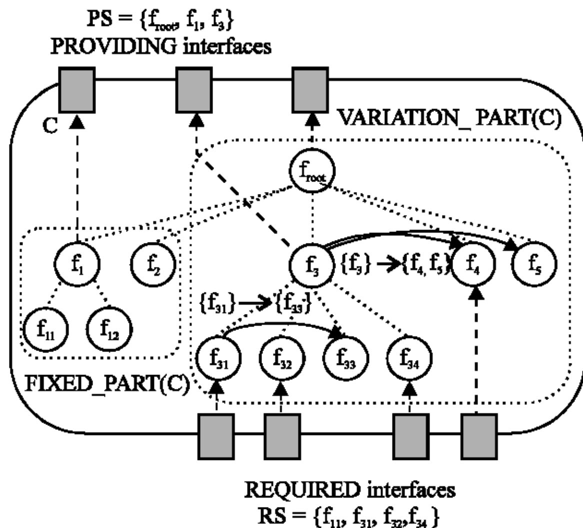

Fig. 2: An example of feature-oriented component

instance of this component. Denote all the instances of C as instance(C)={t_1, t_2,..., t_p}. A component instance could be considered as a set consisted of one feature item of each feature in C.

During software reuse, specific requirements are realized by reusing specific components. The reuse styles can be classified into the following four types, also called four modification levels of component reuse, according to the consistency degree between functions that a component could provide the practical requirements:

- L_1(Directly Reused, DR): directly reuse components to construct new requirements without any instantiation or modification on these components.
- L_2(Reuse after instantiation, IR): reuse an instance of an abstract component after instantiating it, i.e., choosing one specific feature item for each variable feature.
- L_3(Reuse after modification, MR): an existing component cannot fully satisfy the requirements, so some modifications must be done before the component is really reused. The concrete modification means include:

L_{30}(no modify features, NF)☆	
L_{31}(Modify features, MF)	
L_{310}(no modify sub features, NS)☆	L_{315}(modify feature items, MI)
L_{311}(add new child features, AS)	L_{316}(no modify FDs, ND)☆
L_{312}(modify child features, MS)	L_{317}(add new FDs, AD)
L_{313}(no modify feature items, NI)☆	L_{318}(delete FDs, DD)
L_{314}(add new feature items, AI)	L_{319}(modify FDs)

☆ these means are not real modifications and we place them here only for completeness

where, L_{310}-L_{315} belong to refactoring on business logic contained in components and L_{316}-L_{319} belong to refactoring on relationships between functions.

- L_4(No reuse, NR): all existing components cannot completely satisfy requirements, therefore no reuse and new components must be designed and implemented for the requirements.

In conclusion, the process of component reuse can be divided into four phases: component selection, modification, instantiation and reuse, just as shown in Fig. 3.

Stability-oriented component refactoring principle: Suppose that when we use component set {C_1, C_2, ..., C_n} to construct specific systems, the percent of four types of reuse (L_1~L_4) are respectively p_1, p_2, p_3 and p_4 (p_1+p_2+p_3+p_4=1), with the unit reuse cost respectively c_1,

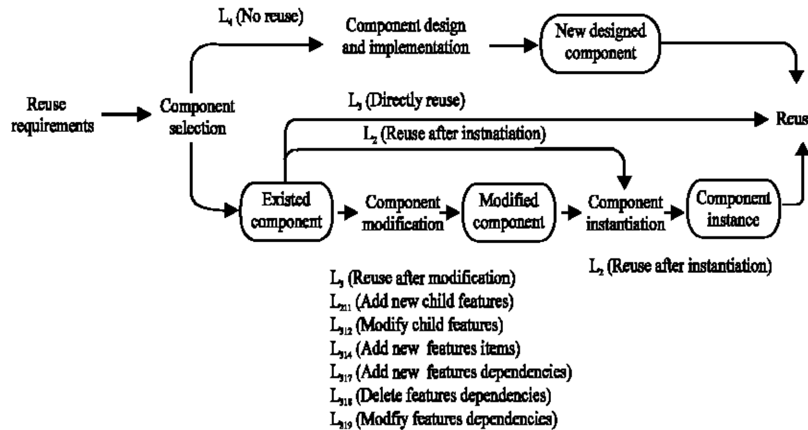

Fig. 3: Primary process for component reuse

c_2, c_3 and c_4. It is easy to understand that $c_1 \leq c_2 \leq c_3 \leq c_4$. Then to decrease reuse cost, we can refactor these n components by the following two goals:

Goal 1: Optimize percent of the four types of components, i.e., p_1, p_2, p_3 and p_4;
Goal 2: Optimize unit reuse cost c_1, c_2, c_3 and c_4.

Aiming at Goal 1, because $c_1 \leq c_2 \leq c_3 \leq c_4$, from the point of modification cost, it is necessary to decrease the modification level of each component as far as possible, i.e., realize the following transformations as far as possible:

Goal 1.1: (L_4) No reuse→(L_3) Reuse after modification;
Goal 1.2: (L_3) Reuse after modification→(L_2) Reuse after instantiation/(L_1)Directly reuse;
Goal 1.3: (L_2)Reuse after instantiation→(L_1)Directly reuse.

Aiming at Goal 1.1, we could add those new designed and implemented components into component library after every times' application construction and by gradual accumulations, the percent of L_4 components would be reduced in subsequent reuses. This is not a task of refactoring. Aiming at Goal 1.2 and 1.3, we have to refactor these components by re-configuring each component's feature spaces.

Aiming at Goal 2, we firstly analysis a specific component's reuse cost. According to the reuse process presented in section 2.1, a component's reuse cost CRC(C) can be considered as the sum of three parts: instantiation cost CI(C), composition cost CC(C) and modification cost CM(C), CRC(C) = CI(C)+CC(C)+CM(C). For L_1 components, CI(C) = 0, CM(C) = 0; for L_2 components, because its fixed part to not be instantiated, CI(C) is determined by the instantiation degree of its variable part and CM(C)=0; for L_3 components, CI(C) is also determined by the instantiation degree of its variable part and CM(C) is determined by the modification degree of C. No matter what types of components, CC(C) is determined by the complexity of interaction relationships with other components. Therefore, the larger the fixed part of a component has, the lower the modification degree of a component holds, the fewer associations with other components, the less reuse cost the component will have. According to the analysis above, we draw a conclusion that component reuse cost could be optimized by

Goal 2.1: Increasing fixed part of a component;
Goal 2.2: Decreasing modification degree of a component;
Goal 2.3: Decreasing associations with other components.

Here we present the concept of stability. If a component requires frequently modifications in reuse, it shows that this component is unstable, along with higher modification cost than other stable ones. Nevertheless, a component C is unstable does not mean that all features/FDs contained in it are all unstable and if we separate those relative stable elements from relative unstable ones to form one new stable component C_s and one new unstable component C_u, this separation will make the modification scope limited to C_u without C_s. In turn, if two components are reused frequently together and they are both relative stable, then they should be merged into one component, therefore when they are reused, the composition cost between them will be saved.

Therefore, we propose the following refactoring principle:

1. Clear separation of modification. For the stable and closely related functions (features), accumulate coarse-grained components; for those unstable functions (features), separate them with stable ones to form unstable and fine-grained components;

   Another view of stability is starting from the reuse frequency of component instances, i.e., if a component instance will be reused in more chances than other ones, then we think it more stable than others. This is called Locality Principle in component reuse, i.e., a frequently reused instance is sure to be reused in the recent future with a higher possibility than those seldom reused instance. If we distill this instance out to form a (semi-) instantiated component, then when the instance is reused, this instantiated component can be directly reused and it is not necessary to instantiate the original component repetitiously, therefore the instantiation cost will be decreased remarkably. So we get another refactoring principle:
2. Clear separation of reuse frequency. For those frequently reused component instances, distill them out to form instantiated components.

   The information required by the two principles are both reuse data produced during reuse and related stability parameters, e.g., reuse frequency of instances, modification degree of components, etc, could be obtained by analyzing these reuse data.

Component reuse data: Table 1 shows an example of practical reuse data for an organization in some periods:

According to five layers "Component-Component Instance-Feature-Feature Item-Feature Dependency", these data are analyzed and get the following statistical data, as shown in Table 2-4.

All the parameters (i.e., reuse style frequencies for component/feature item/feature/feature dependency and reuse frequencies for component instance and feature item) contained in above tables constitute the parameter set for component stability.

Component stability evaluation: Using these stability parameters, here we propose concrete stability evaluation methods for components.

1. Stability of a feature item:

$$S(\tau)=1-\exp\left(-\frac{p(NM,\tau,f,C)}{p(MI,\tau,f,C)}\right)$$

2. Stability of a feature dependency:

$$S(d)=1-\exp\left(-\frac{p(NM,d,f,C)}{p(DD,d,f,C)+2\cdot p(MD,d,f,C)}\right)$$

The reason that we add a coefficient 2 before p(MD,d,f,C) is that the influence of MD is larger than DD.

3. Stability of a feature: $S(f) = S_I(f)\cdot S_M(f)$, where, $S_I(f)$ is f's instantiation stability, expresses the degree of frequently switch between f's different feature items in multiple reuse and is related to the distribution on reuse frequency of each feature item in dom(f). We use stability entropy P(f) to evaluate it, denoted: as

$$S_I(f)=1-\exp\left(-\frac{1}{|dom(f)|\times P(f)}\right),$$
$$P(f)=-\sum_{\tau\in dom(f)} p(\tau,f,C)\log p(\tau,f,C)$$

A larger P(f) indicates that the distribution on reuse frequency of different feature items is more balanced and f is less stable because f has more chances to be frequently switched between its items. A smaller P(f)

Table 1: A segment of practical reuse data of an organization

People	Date	Reused component	Reused instance	Reuse style	Modification contents
User 1	12-03-04	C_4	t_{42}	IR	-
User 2	15-03-04	C_2	-	DR	-
User 2	15-03-04	C_5	t_{52}	MR	Add one feature and its feature items, add two feature dependencies
User 3	16-03-04	C_1	t_{14}	IR	-
User 1	18-03-04	C_4	t_{47}	IR	-
User 2	18-03-04	C_3	-	DR	-
User 4	20-03-04	C_6	t_{62}	IR	-
…					
User 1	05-06-04	C_4	t_{42}	IR	-
User 2	07-06-04	C_5	t_{52}	MR	Modify two feature items of a feature
User 4	07-06-04	C_5	t_{52}	MR	Delete one feature dependency
User 5	10-06-04	C_5	t_{52}	MR	Add one feature, delete one feature dependency
User 6	12-06-04	C_1	t_{13}	IR	-
User 6	12-06-04	C_2		DR	-

Table 2: Statistical reuse data for each component

Component	Reuse frequency	Reuse style	Reuse style frequency
C	p(C)	DR	p(DR,C)
		IR	p(IR,C)
		MR	p(MR,C)

Table 3: Statistical reuse data for each component instance

Component	Component instance	Reuse frequency
C	t_1	$p(t_1,C)$
	t_2	$p(t_2,C)$

Table 4: Statistical reuse data for each feature item

Feature	Feature item	Reuse frequency	Reuse style	Reuse style frequency
f	τ	p(τ, f, C)	NM	p(NM, τ, f, C)
			MI	p(MI, τ, f, C)

Table 5: Statistical reuse data for each feature dependency

FD	Reuse style	Reuse style frequency
d	NM	p(NM, d, f,C)
	DD	p(DD, d, f,C)
	MD	p(MD, d, f,C)

Table 6: Statistical reuse data for each feature

Feature	Reuse frequency	Reuse style	Reuse style frequency
f	p(f,C)	NF	p(NF, f, C)
		NS	p(NS, f, C)
		AS	p(AS, f, C)
		MS	p(MS, f, C)
		NI	p(NI, f, C)
		AI	p(AI, f, C)
		MI	p(MI, f, C)
		ND	p(ND, f, C)
		AD	p(AD, f, C)
		DD	p(DD, f, C)
		MD	p(MD, f, C)

indicates that the distribution is more lopsided and f is more stable.

$S_M(f)$ is f's modification stability denoting the degree of modification in reuse. f can be considered as the composition of three parts: a set of child features child(f), a set of feature dependencies D(f) related to f and a set of feature items dom(f), therefore $S_M(f)$ can be measured by $S_M(f) = S_{SF}(f) \cdot S_{FI}(f) \cdot S_{FD}(f)$ and

$$S_{SF}(f) = \left(1 - \exp\left(-\frac{p(NS,f,C)}{p(AS,f,C) + 2 \cdot p(MS,f,C)}\right)\right) \cdot \prod_{g \in child(f)} S(g)$$

$$S_{FI}(f) = \left(1 - \exp\left(-\frac{p(NI,f,C)}{p(AI,f,C) + 2 \cdot p(MI,f,C)}\right)\right) \cdot \left(1 - \prod_{\tau \in dom(f)} (1 - S(\tau))\right)$$

$$S_{FD}(f) = \left(1 - \exp\left(-\frac{p(ND,f,C)}{p(AD,f,C) + p(DD,f,C) + 2 \cdot p(MD,f,C)}\right)\right) \cdot \prod_{d \in D(f)} S(d)$$

4. Stability of component: $S(C) = S_I(C) \cdot S_M(C)$, where $S_I(C)$ is C's instantiation stability, expresses the distribution on reuse frequency of each component instance (C) and we can also use stability entropy P(C) to evaluate it, denoted as:

$$S_I(C) = 1 - \exp\left(-\frac{1}{|instance(C)| \times P(C)}\right),$$
$$P(C) = -\sum_{t \in instance(C)} p(t,C) \log p(t,C)$$

$S_M(C)$ is C's modification stability and can be denoted as the stability of C's root feature:

$$S_M(C) = S(f_{root})$$

Basic process for component refactoring: In this section, we show the basic process for our stability-oriented component refactoring method using Bayesian analysis, in which the presupposed data is used as the prior distribution of those stability parameters, practical reuse data is used as the sample data and Bayesian analysis method[15,16] is adopted to calculate the posterior distribution of each parameter, according to which,

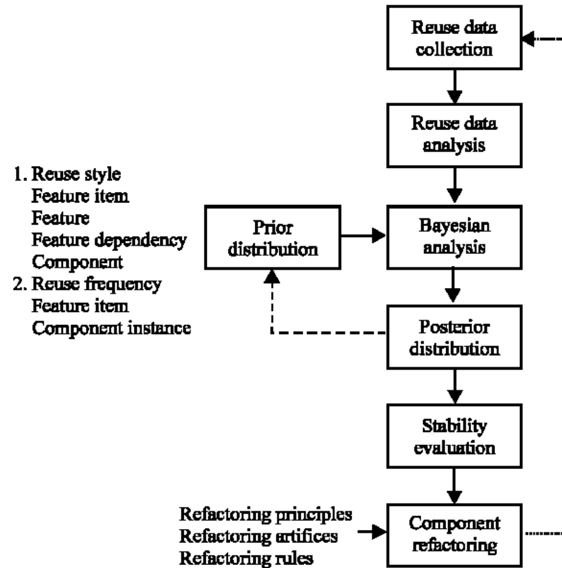

Fig. 4: Process of component refactoring based on statistical reuse data and Bayesian analysis

components are continuously refactored to realize performance optimization. The process is shown in Fig. 4.

From the Fig. 4 we can see that refactoring is an iterative process and the posterior distribution of last refactoring is the prior distribution of next refactoring, with the newest reuse data to adjust component designs and make it much closer to reality.

In this process, the key problems are as followings: (1) How to determine the prior distribution of each stability parameter? (2) How to calculate posterior distribution? (3) How to obtain the final value of each stability parameter according to posterior distribution? (4) How to evaluate each stability parameter according to the prior and posterior value? (5) How to refactor components according to the evaluation results? We will answer these questions in next section.

COMPONENT REFACTORING USING BAYESIAN ANALYSIS

Posterior stability calculation for unknown stability parameters

Prior distribution of unknown parameters: Obviously, a critical feature of Bayesian analysis is the choice of a prior. The key here is that when the data have sufficient signal, even a bad prior will still not greatly influence the posterior.

For a feature item, the possible situations of its changes may have two: (L_{320}) no modify and (L_{322}) modify itself, therefore, each reuse of τ is considered as a Bernoulli test, described as follows:

Suppose $X_1, X_2, ..., X_n$ are independent random draws from the same Bernoulli distribution[17] with parameter θ, thus θ for X_i~Bernoulli (θ) for $i \in \{1,2,....,n\}$. The times of τ is not modified in n reuse is denoted as $Y=\sum_{i=1}^{n} X_i$, which satisfies Binomial distribution, i.e., Y ~ Bernoulli(θ, n) or Y ~ B(n, θ) and the probability density of Y is denoted as

$$P(Y=k)=\binom{n}{k}\theta^k(1-\theta)^{n-k}, 0 \le k \le n, \binom{n}{k}=\frac{n\,!}{k\,!(n-k)\,!}.$$

The parameter $\theta \in [0,1]$ is not a determined value, it has its own probability distribution and before the feature item is reused, we don't know its concrete distribution, so θ is an unknown parameter. Our goal is to obtain the concrete distribution form of θ by Bayesian analysis along with some reuse data. The likelihood function of θ is denoted as $\ell(\theta|Y=k)=\binom{n}{k}\theta^k(1-\theta)^{n-k}$.

According to related literatures of Bayesian analysis, the conjunct distribution[18] of Binominal distribution is Beta distribution[19], so we suppose the prior of θ satisfy Beta distribution, with the probability density function.

$$f(\theta)=\frac{1}{B(\alpha,\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1} \text{ or } \theta \sim \text{Beta}(\alpha,\beta)$$

where, $B(\alpha,\beta)=\frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ and α, β are unknown parameters, $\Gamma(z+1) = z!$ is Gamma function.

Similar to above analysis for a feature, the possible situations it could meet during its reuse can be classified into 8 types, i.e., L_{30}(no modify features), L_{311} (add new child features), L_{312}(modify child features), L_{314} (add new feature items), L_{315}(modify feature items), L_{317} (add new FDs), L_{318}(delete FDs) and L_{319}(modify FDs). Suppose the times of each situation in n reuse is $T_1, T_2...., T_K$ respectively, where, $K = 8$, $\sum_{i=1}^{K} T_i = n$, then the joint probability distribution of $T_1, T_2, ..., T_K$ satisfies Multinominal distribution[20,21], denoted as

$$P(T_1=t_1,...,T_K=t_K)=\frac{n\,!}{\prod_{i=1}^{K} t_i\,!}\prod_{i=1}^{k}\theta_i^{t_i},$$

where $\sum_{i=1}^{K} t_i = n$, or $(T_1,...., T_K)$, where $\theta_1,..., \theta_K$ are the probability of each situation and $\sum_{i=1}^{K}\theta_i = 1$.

Parameters $\theta_1, ...,\theta_K$ are also unknown parameters, with the likelihood functions are

$$\ell(\theta_1,...,\theta_K|T_1=t_1,...,T_K=t_K)=\frac{n\,!}{\prod_{i=1}^{K} t_i\,!}\prod_{i=1}^{k}\theta_i^{t_i}$$

Suppose the joint probability distribution of $\theta_1, ...,\theta_K$ satisfies the conjunct distribution of Multinominal distribution, Dirchlet distribution[21], with the density function as

$$f(\theta_1,...,\theta_K)=\frac{1}{B(\theta)}\prod_{i=1}^{K}\theta_i^{\alpha_i-1}=\frac{\Gamma\left(\sum_{i=1}^{K}\alpha_i\right)}{\prod_{i=1}^{K}\Gamma(\alpha_i)}\prod_{i=1}^{K}\theta_i^{\alpha_i-1}$$

or

$$(\theta_1,...,\theta_K) \sim \text{Dirichlet}(\alpha_1,...,\alpha_K)$$

where $\alpha_1,...,\alpha_K$ are unknown parameters, $\alpha_i>0$. This can be considered as the prior distribution of $\theta_1, ...,\theta_K$.

Follow the similar analysis process above we know that (1) the probability of reuse style of FD, (2) the reuse frequency of all feature items of a feature, (3) the reuse frequency of all instances of a component, all satisfies

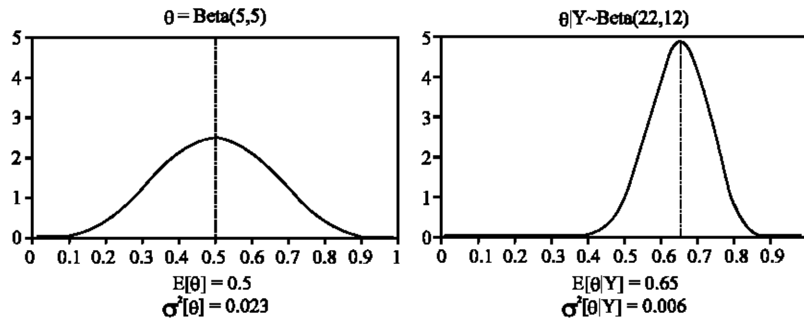

Fig. 5: Probability density function (PDF) for prior and posterior distribution of function item's reuse style

Multinominal distribution, whose parameters satisfy Dirchlet distribution.

Posterior distribution of unknown parameters: From analysis we know that all the stability parameters can be concluded as Binominal and Multinominal distributions, whose parameters satisfy Beta and Dirchlet distributions respectively. In this section, we use feature item and feature's reuse as examples to present Bayesian analysis process.

Suppose in n reuse of one feature item there are k times it is not modified, then for parameter θ in Binominal distribution, its posterior distribution can be obtained using the following theorem:

Theorem 1: The posterior distribution of θ satisfies $\theta|k\sim\text{Beta}(k+\alpha, n+\beta-k)$.

For Multinominal distribution, suppose in n reuse of one features, there are $t_1,\ldots, t_k$ times it is in each of the K situations, then for parameter vector $(\theta_1, \theta_2, \theta_3)$ the posterior distribution can be obtained using the following theorem.

Theorem 2: The joint posterior distribution of $(\theta_1,\ldots,\theta_k)$ satisfies

$$(\theta_1,\ldots,\theta_K|t_1,\ldots,t_K)\sim\text{Dirichlet}(t_1+\alpha_1,\ldots,t_K+\alpha_K)$$

From the joint distribution we could obtain the marginal distribution of each parameter in $(\theta_1,\ldots,\theta_k)$ using Theorem 3.

Theorem 3: The marginal posterior distribution of parameter θ_1 satisfies

$$(\theta_i|T)\sim\text{Beta}\left(t_i+\alpha_i-1, \sum_{j=1}^{K}(t_j+\alpha_j)-(t_i+\alpha_i)-1\right)$$

We ignore the proofs of these theorems, please refer to[15-17].

Here we present an example. Support a feature item τ has a probability θ not to be modified during process and satisfies $\theta\sim\text{Beta}(5,5)$ and the statistical data of τ in a period shown that in 24 times reuse, there are 17 times that it is reused directly. According to Theorem 1 we know that the posterior distribution of θ is $\theta|(k = 17, n = 24)\sim\text{Beta}(17+5, 24-17+5) = \text{Beta}(22,12)$. Figure 5 shows that the posterior distribution is more concentrated than the prior, which shows that the practical reuse data have taken effective adjustment on the prior distribution.

In Table 7 concluded all the information of stability parameters, including their prior and posterior distribution representation forms.

Posterior value estimation: After obtaining the posterior distribution of each stability parameter using Bayesian analysis, here we present the way of how to estimate the value of each parameter. We adopt the maximum likelihood estimation (MLE) method in point estimation to do this.

Theorem 4: For a parameter θ which satisfies Beta distribution (e.g., each reuse style probability of feature September 29, 2005 items), its maximum likelihood estimation is

$$\hat{\theta}=\frac{k+\alpha-1}{n+\alpha+\beta-2}$$

Theorem 5: For parameter vector $(\theta_1,\ldots,\theta_k)$ which satisfies Dirchlet distribution (e.g., reuse style of features, reuse frequency of feature items, etc), the MLE of arbitrary parameter in $(\theta_1,\ldots,\theta_k)$ is

$$\hat{\theta}_i=\frac{t_i+\alpha_i-1}{n+\alpha_0-2}, i\in\{1,2,\ldots,K\}, n=\sum_{i=1}^{K}t_i, \alpha_0=\sum_{j=1}^{K}\alpha_j$$

The proofs of theorem 4 and 5 can be find by many researchers[15,16,22].

Table 7: Prior and posterior probability distribution for parameters in general stability

	Reuse style of feature items	Reuse style of FDs
Parameters	θ_{NI}	θ_{ND}, θ_{DD}, θ_{MD}
Meanings of parameters	Probability that a feature item is reused without modification	Probability that a feature dependency is reused without modification, after deletion and after modification
Prior dist.	$\theta_{NI} \sim \text{Beta}(\alpha, \beta)$	$(\theta_{ND}, \theta_{DD}, \theta_{MD}) \sim \text{Dirichlet}(\alpha_{ND}, \alpha_{DD}, \alpha_{MD})$
Sample data	There are k times that the feature item is directly reused without modification in n times of reuse	There are t_{ND}, t_{DD}, t_{MD} times that the FD is reused with modification, after deletion and after modification respectively in n times of reuse
Posterior dist.	$\theta_{NI} \mid k \sim \text{Beta}(k+\alpha, n+\beta-k)$	$(\theta_{ND}, \theta_{DD}, \theta_{MD} \mid t_{ND}, t_{DD}, t_{MD}) \sim \text{Dirichlet}(t_{ND}+\alpha_{ND}, t_{DD}+\alpha_{DD}, t_{MD}+\alpha_{MD})$
Stability dist.	$S(\tau) = 1 - \exp\left(-\frac{\theta_{NI}}{1-\theta_{NI}}\right)$	$S(d) = 1 - \exp\left(-\frac{\theta_{ND}}{\theta_{DD} + 2\cdot\theta_{MD}}\right)$
	Reuse frequency of feature items	Reuse frequency of component instances
Parameters	$\left(\theta_{IF}^{(1)}, \ldots, \theta_{IF}^{(m)}\right), m = \lvert \text{dom}(f) \rvert$	$\left(\theta_{IF}^{(1)}, \ldots, \theta_{IF}^{(m)}\right), p = \lvert \text{instance}(f) \rvert$
Meanings of parameters	The reuse frequency of m feature items of f in practical reuse	The reuse frequency of p instances of C in practical reuse
Prior dist.	$\left(\theta_{IF}^{(1)}, \ldots, \theta_{IF}^{(m)}\right) \sim \text{Dirichlet}\left(\alpha_{IF}^{(1)}, \ldots, \alpha_{IF}^{(m)}\right)$	$\left(\theta_{CF}^{(1)}, \ldots, \theta_{CF}^{(p)}\right) \sim \text{Dirichlet}\left(\alpha_{CF}^{(1)}, \ldots, \alpha_{CF}^{(p)}\right)$
Sample data	$t_{IF}^{(1)}, \ldots, t_{IF}^{(m)}$	$t_{CF}^{(1)}, \ldots, t_{CF}^{(p)}$
Posterior dist.	$\left(\theta_{IF}^{(1)}, \ldots, \theta_{IF}^{(m)} \mid t_{IF}^{(1)}, \ldots, t_{IF}^{(m)}\right) \sim \text{Dirichlet}\left(t_{IF}^{(1)} + \alpha_{IF}^{(1)}, \ldots, t_{IF}^{(m)} + \alpha_{IF}^{(m)}\right)$	$\left(\theta_{CF}^{(1)}, \ldots, \theta_{CF}^{(p)} \mid t_{CF}^{(1)}, \ldots, t_{CF}^{(p)}\right) \sim \text{Dirichlet}\left(t_{CF}^{(1)} + \alpha_{CF}^{(1)}, \ldots, t_{CF}^{(p)} + \alpha_{CF}^{(p)}\right)$
Stability dist.	$S_I(f) = 1 - \exp\left(\frac{1}{\lvert \text{dom}(f) \rvert \times \left(\sum_{i=1}^{m} \theta_{IF}^{(i)} \log \theta_{IF}^{(i)}\right)}\right)$	$S_I(C) = 1 - \exp\left(\frac{1}{\lvert \text{instance}(C) \rvert \times \sum_{i=1}^{p} \theta_{CF}^{(i)} \log \theta_{CF}^{(i)}}\right)$

	Reuse styles of features
Parameters	$\theta_{NM}, \theta_{AS}, \theta_{MS}, \theta_{AI}, \theta_{MI}, \theta_{AD}, \theta_{DD}, \theta_{MD}$
Meanings of parameters	The corresponding probability of eight reuse styles of a feature in practical reuse
Prior dist.	$(\theta_{NM}, \ldots, \theta_{MD}) \sim \text{Dirichlet}(\alpha_{NM}, \ldots, \alpha_{MD})$
Sample data	In n times of reuse, eight reuse styles appear with the number t_{NM}, ..., t_{MD} respectively
Posterior dist.	$(\theta_{NM}, \ldots, \theta_{MD} \mid t_{NM}, \ldots, t_{MD}) \sim \text{Dirichlet}(t_{NM} + \alpha_{ND}, \ldots, t_{MD} + \alpha_{MD})$

Table 7: Continue

	Reuse styles of features
Stability dist.	$S(f)=S_I(f)\cdot$ $\left(1-\exp\left(-\frac{\theta_{NS}}{\theta_{AS}+2\cdot\theta_{MS}}\right)\right)\cdot\prod_{g\in child(f)}S(g)\cdot \quad \rightarrow S_{SF}(f)$ $\left(1-\exp\left(-\frac{\theta_{NI}}{\theta_{AI}+2\cdot\theta_{MI}}\right)\right)\cdot\left(1-\prod_{\tau\in dom(f)}(1-S(\tau))\right)\cdot \quad \rightarrow S_{FI}(f)$ $\left(1-\exp\left(-\frac{\theta_{ND}}{\theta_{AD}+\theta_{DD}+2\cdot\theta_{MD}}\right)\right)\cdot\prod_{d\in D(f)}S(d) \quad \rightarrow S_{FD}(f)$
	Component
Parameters	—
Meanings of parameters	—
Prior dist.	—
Sample data	—
Posterior dist.	—
Stability dist.	$S(C)=S_I(C)\cdot S(f_{root})$

After getting the MLE of each stability parameter, we can obtain the final stability using these parameters. We use $S_{pior}(°)$ and $S_{post}(°)$ as the prior and posterior MLE of stability, where ° may be τ, d, f, or C.

Besides $S_{pior}(°)$, we also use "stability change ratio" to denote the variation tendency or change degree, of stability: $\Delta S(°)=\frac{S_{post}(°)-S_{prior}(°)}{S_{prior}(°)}$. The larger $\Delta S(°)$ is, the larger degree that the original design is inconsistency and not matching with practical reuse and the more urgent it should be refactored. In next section we will present some basic factoring rules based on $S_{pior}(°)$ and $\Delta S(°)$.

Basic refactoring artifices and rules: According to two basic principles, we here present three basic refactoring operations.

O1(Component granularity decomposition, CGD) Component C's one CGD refers to decomposing C's feature space Ω(C) into two sub space Ω_1 and Ω_2, each of which is mapped to an independent finer-grained component C_1 and C_2, respectively, denoted as $C\xrightarrow{CGD}(C_1,C_2)$.

O2(Component granularity mergence, CGM) One CGM refers to merging feature spaces $\Omega(C_1)$, $\Omega(C_2)$ of two components C_1 and C_2 into one space Ω and forming a coarser-grained component C, denoted as $(C_1,C_2)\xrightarrow{CGM}C$.

O3(Component Instance Decomposition, CID) Component C's one CID refers to decomposing C's instance set instance(C) into two subset I_1 and I_2, each of which is mapped to a (semi-)instantiated component C_1 and C_2, respectively, denoted as $C\xrightarrow{CID}(C_1,C_2)$.

According to the variation tendency of $S_{post}(°)$ and $\Delta S(°)$, using above three basic operations, we present the following eight refactoring rules.

Rule 1: (Feature Item Decomposition Rule, FID) ∀ τ ∈ dom (f), g ∈ Ω (C) if $S_{post}(\tau)$ (where δ is a valve value of stability, usually with the value 0.4~0.5) and $\Delta S(\tau)<0$, then decompose τ out from dom(f) to form two new features f_1 and f_2, where $dom(f_1)$ = {τ}, $dom(f_2)$ = dom (f_2) = dom (f) - {τ}. Now f_1 and f_2 are both called the ectypal features of f.

FID shows that, if a feature item's stability decreases and is less than the valve value, this feature item should be distilled out separately.

Rule 2: (Feature Item Mergence Rule, FIM) If $\exists f_i, f_j \in \Omega(C)$ are the two ectypal features of f and ∀ τ ∈ dom (f_i) ∪ dom (f_j), both $S_{pior}(\tau)>\delta$ and $\Delta S(\tau)>0$ must be true, then merge f_i, f_j into one new feature f_{ij}.

FIM is the inverse rule of FID and it shows that if some feature items' stability increases, they should be merged. Figure 6 gives an example of FIM and FID.

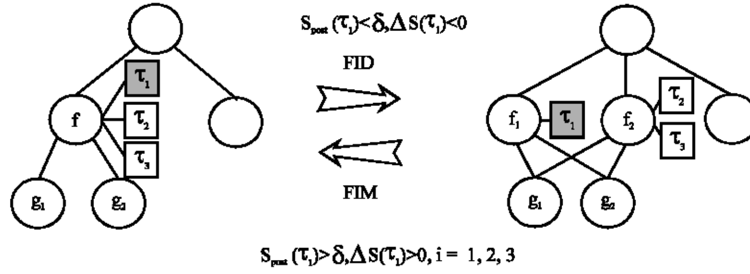

Fig. 6: Decomposition and merge rules for a feature item

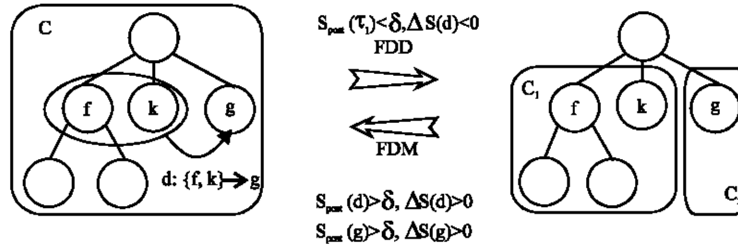

Fig. 7: Decomposition and merge rules for a feature dependency

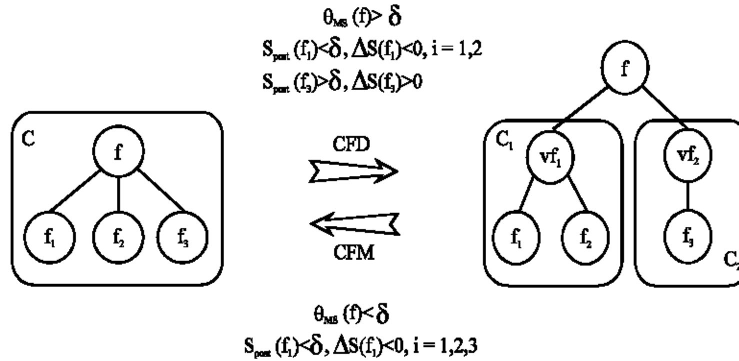

Fig. 8: Decomposition and merge rules for child features of a feature

Rule 3: (Feature Dependency Decomposition Rule, FDD) $\forall s \in D(f)$, $f \in \Omega(C)$ and d: $X \rightarrow Y$, if $\exists g \in Y$, $g \in \Omega$ (C) and S_{post}, (d)<δ, then decompose f out from C to form a new component.

Rule 4: (Feature Dependency Mergence Rule, FDM) $\forall s \in D(f)$, $f \in \Omega(C)$ and d: $X \rightarrow Y$, if S_{post}, (d)<δ, ΔS(d)>0 and and there $\exists g \in T$, $g \notin \Omega$ (C) and S_{post} (g)<δ, S_{post} (g)>δ, then add g into Ω(C).

FDD and FDM are also two inverse rules, which shows that if a feature dependency is usually modified or deleted, then the cohesion between features connected by the feature dependency is fallen and these features are less changed together, so they should be decomposed if they belong to the same component before; Contrarily, they should be merged into one component. The process is illustrated in Fig. 7.

Rule 5: (Child Feature Decomposition Rule, CFD) $\forall f$, if chid(f) $\neq \varnothing$, f and child(f) are contained in the same component C and θ_{MS}(f)>δ, then decompose child(f) into two subsets CF_1 and CF_2, which makes $\forall g \in CF_2$, there are S_{post} (g)<δ, S_{post} (g)<δ and S_{post} (g) < δ, S_{post} (g)<δ, there are S_{post} (g)<δ, S_{post} (g)<δ,. Construct virtual features vf_1 and vf_2 for CF_1 and CF_2, respectively and satisfy child(vf_1) = CF_1, child(vf_2) = CF_2, child(vf_1) = {Vf_1, Vf_2}. Decompose vf_2 out to form a new component.

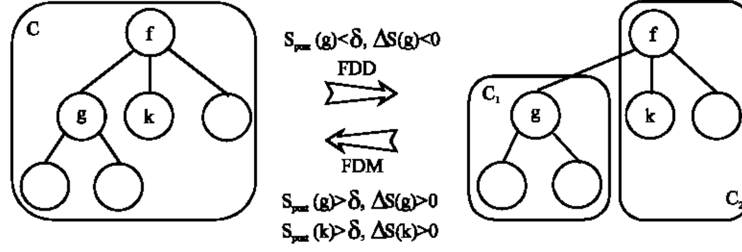

Fig. 9: Decomposition and merge rules for a feature

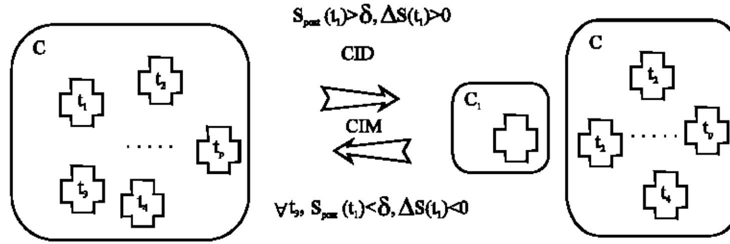

Fig. 10: Decomposition and merge rules for component instances

Rule 6: (Child Feature Mergence Rule, CFM) It is the reversible rule of CFD and we will not show its details.

CFD and CFM show that if a set of child features of f are usually modified, then they should be decomposed with other child features; on the contrary, they should be merged. The process is illustrated in Fig. 8.

Rule 7: (Feature Decomposition Rule, FSD) $\forall g \in \Omega(C)$, if $S_{post}(g)<\delta$ and $\Delta S(f)<0$, then decompose f from $\Omega(C)$ and form a new component.

Rule 8: (Feature Mergence Rule, FSM) $\forall g = f_{root}(C_1)$ and $S_{post}(g)>\delta$, $\Delta S(g)>0$, if there $\exists k \in \Omega(C_2)$, g, k child(f) and $f \in \Omega(C_2)$, $S_{post}(k)>\delta$, $\Delta S(K)>0$, then merge C_1 and C_2 into one component.

FSD and FSM show that if the stability of a feature decreases, then it should be decomposed out; contrarily, it should be merged with other stable sibling features, as shown in Fig. 9.

Rule 9: (Component Instance Decomposition Rule, CID) $\forall t \in$ instance(C), $S_{post}(t)>\delta$ if $\Delta S(t)>0$, then decompose t out to form an instantiated component C_1, where instance $(C_1) = \{t\}$, instance (C) - {t}.

Rule 10: (Component Instance Mergence Rule, CIM) If $\exists C_1$, C_2 are both instantiated components, where instance(C_1) = {t}, instance(C_2) $\{t_2\}$ and S_{post} $(t_1)<0$, $S_{post}(t_2)<\delta$, $\Delta S(t_2)<0$, then merge t_1 and t_2 into a semi-instantiated component C, where instance (C) = $\{t_1, t_2\}$.

CID and CIM show that if an instance of a component is more frequently reused than other instances and the reuse frequency is more than the valve value, then it should be decomposed to form a new instantiated component; contrarily, two seldom reused instantiated components should be merged over again. The process is illustrated in Fig. 10.

Stability improvement evaluation: Here we present the method to evaluate the performance improvement after using the ten refactoring rules. Suppose there exists component set $\mathbb{C} = \{C_1, C_2, ..., C_n\}$ and after carrying refactoring rules for multiple times, we obtain the final component set $\mathbb{C}' = \{C_1', C_2', ..., C_m'\}$, then the following metrics may be adopted to evaluate the performance improvement.

1. Δ_{I-D}, denoting the improvement on number of components that can be directly reused without instantiation in $\mathbb{C}'$ relative to $\mathbb{C}$, i.e.,

$$\Delta_{I\rightarrow D} = \frac{\lambda_D - \lambda_D'}{\lambda_D},$$

where

$$\lambda_D' = \frac{1}{n} \cdot \left| \left\{ C_i' \middle| C_i' \in \mathbb{C}', \text{VARIABIE_PART}(C_i') = \varnothing \right\} \right|,$$

$$\lambda_D = \frac{1}{m} \cdot \left| \left\{ C_i \middle| C_i \in \mathbb{C}, \text{VARIABIE_PART}(C_i) = \varnothing \right\} \right|;$$

2. Δ_{IRC}, denoting the improvement on total instantiation cost for all components in $\mathbb{C}'$ relative to $\mathbb{C}$, i.e.,

$$\Delta_{IRC} = \frac{IRC(\mathbb{C}) - IRC(\mathbb{C}')}{IRC(\mathbb{C})} = 1 - \frac{\sum_{C_i' \in \mathbb{C}'} IRC(C_i')}{\sum_{C_i \in \mathbb{C}} IRC(C_i)},$$

or, we can approximately use the improvement on the total number of fixed features, i.e.,

$$\Delta_{IRC} = \frac{\sum_{C_i' \in \mathbb{C}'} \left|FIXED_PART(C_i')\right|}{\sum_{C_i \in \mathbb{C}} \left|FIXED_PART(C_i)\right|} - 1;$$

3. $\Delta_{M\text{-}I}$, denoting the improvement on the number of components that can be reused directly or after instantiation without modification in $\mathbb{C}'$ relative to $\mathbb{C}$,

i.e. $\Delta_{M \to I} = \frac{\hbar_I - \hbar_I'}{\hbar_I}$, where

$$\hbar_I' = \frac{1}{n} \cdot \left|\left\{C_i' \middle| C_i' \in \mathbb{C}', \forall f' \in \Omega(C_i'), \theta_{NM}(f') = 1\right\}\right|,$$

$$\hbar_I = \frac{1}{m} \cdot \left|\left\{C_i \middle| C_i \in \mathbb{C}, \forall f \in \Omega(C_i), \theta_{NM}(f) = 1\right\}\right|.$$

4. Δ_{CRC}, denoting the improvement on the total modification cost for all components in $\mathbb{C}'$ relative to $\mathbb{C}$, i.e.,

$$\Delta_{CRC} = \frac{CRC(\mathbb{C}) - CRC(\mathbb{C}')}{CRC(\mathbb{C})} = 1 - \frac{\sum_{C_i' \in \mathbb{C}'} CRC(C_i')}{\sum_{C_i \in \mathbb{C}} CRC(C_i)};$$

or we can approximately use the improvement on the sum of feature stability, i.e.,

$$\Delta_{CRC} = \frac{\sum_{C_i' \in \mathbb{C}'} \sum_{f' \in \Omega(C_i')} \theta_{NM}(f')}{\sum_{C_i \in \mathbb{C}} \sum_{f \in \Omega(C_i)} \theta_{NM}(f)} - 1.$$

5. Δ_S, denoting the improvement on the average stability of components in $\mathbb{C}'$ relative to $\mathbb{C}$,

i.e. $$\Delta_S = \frac{\sum_{C_i' \in \mathbb{C}'} S(C_i')}{\sum_{C_i \in \mathbb{C}} S(C_i')} - 1.$$

A CASE

In a practical Enterprise Resource Planning (ERP) system, there are 25 reusable components in Purchasing domain (because of limited space, we will not list feature spaces of each components). We collected some reuse data produced during these components were reused for constructing and implementing purchasing software systems in 4 manufacturing enterprises as the sample data for our refactoring method. Before Bayesian analysis, the values of α-series parameters in those unknown stability parameters (θ-series) are shown in Table 8.

According to the statistical data and 10 refactoring rules, we did refactoring on these 25 components. Table 9 only lists several refactoring operations as examples due to limited space.

After refactoring, we obtain 45 components total, in which there are 12 ones unchanged, 18 ones are the results of decomposing and mergence of original

Table 8: Parameter values for calculating posterior stability

Scope	Parameters	Value
Reuse styles of feature items	α, β	$\alpha = 2$, $\beta = 5$
Reuse styles of feature dependencies	α_{ND}, α_{DD}, α_{MD}	$\alpha_{ND} = 5$, $\alpha_{DD} = 1$, $\alpha_{MD} = 2$
Reuse styles of features	$\alpha_{NM}, \alpha_{AS}, \alpha_{MS}, \alpha_{AI}$, $\alpha_{MI}, \alpha_{AD}, \alpha_{DD}, \alpha_{MD}$	$\alpha_{NM} = 60$, $\alpha_{AS} = 10, \alpha_{MS} = 30$, $\alpha_{AI} = 20, \alpha_{MI} = 10$, $\alpha_{AD} = 3, \alpha_{DD} = 2, \alpha_{MD} = 5$
Reuse frequencies of feature items	$\alpha_{IF}^{(1)}, \ldots, \alpha_{IF}^{(m)}$	$\alpha_{IF}^{(i)} = 2$
Reuse frequencies of component instances	$\alpha_{CF}^{(1)}, \ldots, \alpha_{CF}^{(p)}$	$\alpha_{CF}^{(i)} = 2$

Table 9: Refactoring operations on component set

Original components	Refactoring rules	Refactoring operations	Objective components
C_1	FID	Separate feature item τ_{1113} with other items	C_1
C_2	FSD	Separate f_{113} out to form new component	$C_{2\text{-}1}$,$C_{2\text{-}2}$
C_3,C_4	FSM	Merge f_{12} and f_{13} together	C_{3+4}
C_9	CFD	Decompose child feature set of f_{31} into two subsets {f_{311}, f_{312}} and {f_{313}}, the latter subset forms a new component	C_9,$C_{9\text{-}1}$
$C_{2\text{-}1}$	CID	Distill component instance t_1 and t_2 out to form two new instantiated components respectively	$C_{2\text{-}1}$,$C_{2\text{-}1\wedge 1}$,$C_{2\text{-}1\wedge 2}$

Table 10: Performance improvements before and after refactoring

Metrics	Meaning	Before Refactoring	After Refactoring	Improvement
$\Delta_{I\text{-}D}$	Number of components that can be directly reused without instantiation	5	20	300%
	Percent of components that can be directly reused without instantiation	0.2	0.444	122%
Δ_{IRC}	Sum of instantiation cost for all components (using sum of number of fixed features to approximate measure)	23	48	109%
$\Delta_{M\text{-}I}$	Number of components that can be reused after instantiation without modification	5	11	120%
	Percent of components that can be reused directly or after instantiation without modification	0.2	0.229	14.6%
Δ_{CRC}	Sum of modification cost for all components (using sum of all features' stability to approximate measure)	41.627	52.036	25.1%
Δ_S	Average stability of all components	0.239	0.273	14.2%

Table 11: Comparisons between component refactoring methods

	Methods in literatures	Our methods
Refactoring goals	e.g., high-cohesion and low-coupling, change scope, etc	Reuse cost
Refactoring information source	By analysis on component semantics and structure and parameters produced during component design phase, without considering the deviation between the expectation of design and practical reuse requirements	Starting from component stability parameter values in design phase (prior data), consider the reuse data produced during practical reuse (sample data), to get the refactoring causations (posterior data) and forms a "feedback cycle"
Iteration	Because the information source is determinate, refactoring can only proceed once to eliminate all bad designs, therefore component performance cannot be optimized continuously and repeatedly using these methods	Can be applied multiple times and the results of the former refactoring is the starting point of the next refactoring (prior data), using those reuse data produced after the former refactoring as sample data
Refactoring level	Usually aiming at structure refactoring in abstract level	Besides structure refactoring in abstract level, also include refactoring in instance level
Dynamic characteristics	Consider components and component-based systems as static entities, without fully considering the dynamic characteristics produced by the continuous evolution for adapting environment changes[23]	Besides the static structural attributes of components, also consider all types of dynamic modification styles on every part of components and their frequencies
Easy to use	Stress on the refactoring in abstract semantics space using formal methods, which are difficult to understand and apply widely in practice	Focus mainly on the statistics and analysis of data produced in practical reuse, easy to understand and apply
Refactoring operations	e.g., moving attributes or methods between components, merging or decomposing components	Primarily granularity refactoring (including mergence, decomposition) for stability and instantiation refactoring based on locality principle
Typical methods	e.g., 9-interaction matrix[1] BusCod Model[3] Component Refactoring Framework[4] Extract Component Refactoring[6]	Stability-oriented refactoring using Bayesian analysis method
Key techniques	e.g., Component interaction matrix Business semantics meta-model Generalization and inheritance in OO	Bayesian analysis Stability evaluation Locality principle

components and the rest 15 ones are instantiated components of original ones.

In Table 10 presents the performance improvements before and after refactoring.

CONCLUSIONS

In this paper we firstly elaborated the meanings and the motivations of component refactoring, considering refactoring is a way to eliminate "bad" designs of components and clarified that refactoring should be done according to practical reuse data to correct the inconsistencies between component designs and practical reuse requirements. We emphatically discussed stability-oriented refactoring methods to decrease reuse cost, with the corresponding refactoring principles, operations and rules, in which the decomposition/mergence/instantiation on component feature spaces were adopted as the basic ways for refactoring. In this method, Bayesian analysis method was imported to calculate the posterior estimation

of stability parameters recurring to practical reuse data to remove those designs that are inconsistency with practical reuse requirements and make component more suitable for reuse.

Compared with other refactoring methods in literatures, our methods have the following advantages listed in Table 11.

ACKNOWLEDGMENTS

Research works in this study are partial supported by the Specialized Research Fund for the Doctoral Program of Higher Education (SRFDP) in China (Grant No. 20030213027), the National High-Tech. R&D Program (863) for CIMS in China (No. 2003AA4Z3210) and the National Natural Science Foundation of China (No. 60573086).

REFERENCES

1. Stojanović, Z., 2005. A Method for Component-Based and Service-Oriented Software Systems Engineering. Ph.D Thesis, Delft University of Technology, The Netherlands.
2. Wang, Z.J., X.F. Xu and D.C. Zhan, 2005. A component optimization design method based on variation point decomposition. In: Proceedings of the 3rd ACIS International Conference on Software Engineering, Research, Management and Applications (SERA'05), IEEE Computer Society, pp: 399-406.
3. P. Vitharana, H. Jain and F. Zahedi, 2004. Strategy-based design of reusable business components. IEEE Transactions on Systems, Man and Cybernetics- Part C: Applications and Reviews, 34: 460-474.
4. Caballero, R. and S. Demurjian, 2002. Towards the formalization of a reusability framework for refactoring. In: Software Reuse: Methods, Techniques and Tools (Proceedings Seventh International Conference on Software Reuse), Springer-Verlag LNCS 2319, pp: 293-308.
5. France, S. Ghosh, E. Song and D.K. Kim, 2003. A metamodeling approach to pattern-based model refactoring. IEEE Software, 20: 52-58.
6. Washizaki, H. and Y. Fukazawa, 2003. Automated extract component refactoring. In: Extreme Programming and Agile Processes in Software Engineering (Proceedings of the 4th International Conference XP 2003), Springer-Verlag, LNCS 2675, pp: 328-330.
7. Quan, L., J.F. He and Z.M. Liu, 2005. Refactoring and pattern-directed refactoring: A formal perspective. Technical Report, International Institute for Software Technology, the United Nations University, Tokyo, Japan.
8. Cinneide, M., 2000. Automated Application of Design Patterns: A Refactoring Approach. Ph.D. Thesis Trinity College, University of Dublin, Dublin, Ireland.
9. Mili, H., A. Mili, S. Yacoub and E. Addy, 2002. Reuse-Based Software Engineering: Techniques, Organization and Controls. John Wiley and Sons Ltd.
10. Tracz, W., 1994. Implementation Working Group Summary. In: Proceedings of Reuse in Practice Workshop, IDA Document D-754, Pittsburgh, PA, pp: 10-19.
11. Allen, R. and D. Garlan, 1997. A formal basis for architectural connection. ACM Transactions on Software Engineering and Methodology, 6: 213-249.
12. Wu, Q., J. Chang, H. Mei and F.Q. Yang, 1997. JBCDL: An object-oriented component description language. In: Proceedings of 24th International Conference on Technology of Object-Oriented Languages ASIA. IEEE Computer Society Press, pp: 198-205
13. Kang, K.C., S.G. Cohen, J.A. Hess, W.E. Novak and A.S. Peterson, 1990. Feature-Oriented domain analysis (FODA) feasibility study. Technical Report, CMU/SEI-90-TR-21, Carnegie Mellon University, Software Engineering Institute.
14. Kang, K.C., S. Kim, J. Lee, K. Kim, E. Shin and M. Huh, 1998. FORM: A feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering, 5:143-168.
15. Leonard, T. and J.S.J. Hsu, 2001. Bayesian Methods: An Analysis for Statisticians and Interdisciplinary Researchers. Cambridge University Press.
16. Berger, J.O., 1993. Statistical Decision Theory and Bayesian Analysis, Second Edition. Springer.
17. Wikipedia, 2005. Binomial distribution. http://en.wikipedia.org/wiki/Binomial_distribution
18. Wikipedia, 2005. Conjugate prior. http://en.wikipedia.org/wiki/Conjugate_prior
19. Wikipedia, 2005. Beta distribution. http://en.wikipedia.org/wiki/Beta_distribution
20. Wikipedia, 2005. Multinomial distribution. http://en.wikipedia.org/wiki/Multinomial_ distribution
21. Wikipedia, 2005. Dirichlet distribution. http://en.wikipedia.org/wiki/Dirichlet_distribution
22. Congdon, P., 2001. Bayesian Statistical Modeling. John Willey and Sons.
23. Wang, Y.H., S.K. Zhang, Y. Liu and L.F. Wang, 2004. Ripple-Effect analysis of software architecture evolution based on reachability matrix.. J. Software, 15: 1107-1115.