객체 지향 시스템에서의 클래스 간 의존성 강도 측정을 위한 커플링 척도

(A Coupling Metric for Measuring Strength of Dependency between Classes in Object-Oriented Systems)

화 지 민 † 이 숙 희 †
(Ji Min Hwa) (Suk Hee Lee)

권 용 래 ††
(Yongr Rae Kwon)

요 약 객체지향 패러다임에서 커플링은 유지보수활동에 가장 큰 영향을 주는 속성 중 하나로 많은 연구들이 진행되어 왔다. 그러나 기존의 커플링 척도는 클래스들간의 의존성 여부만 측정할 뿐 의존성 강도를 표현하지 못하기 때문에 리팩토링이나 시스템 분해와 같이 의존성 강도 정보가 필요한 유지보수활동에서 의사 결정을 지원하는데 한계가 있다. 이 논문에서는 이를 해결하기 위해 클래스 간의 의존성 여부뿐만 아니라 유지 보수성-재검사 비용 측면에서 클래스 간의 의존성 강도까지 표현할 수 있는 커플링 척도를 제안한다. 그리고 시스템 분해 문제에 적용한 예제를 통하여 시스템 유지 보수를 위한 여러 분야에 적용될 수 있음을 보인다.

키워드 : 커플링, 의존성 강도, 척도, 재검사 비용

Abstract The coupling measurements of object-oriented programs can be valuable information for various maintenance tasks and there exists a variety of metrics proposed by numerous researchers. Those metrics, however, cannot measure the strength of dependencies between classes, but only imply the existence of dependencies. Hence they are of limited value for assisting maintenance tasks such as refactoring and system decomposition, which requires information about the strength of dependency. In this paper, a coupling metric which can measure the strength of dependency as well as detect the existence of dependencies is proposed. Our coupling metric is evaluated based on the cost required for performing the maintenance tasks. We have applied the proposed coupling metric to an example of system decomposition in an effort to assess the potential benefits of our approach for maintenance tasks.

Key words : Coupling, the strength of dependencies between classes, metrics, retest cost

1. 서 론

객체지향 패러다임에서 커플링은 클래스 간의 의존성을 나타내는 구조적 속성으로 시스템의 유지보수활동에 가장 큰 영향을 주는 속성 중 하나로 다양한 유지보수 활동의 중요한 지표로 사용되고 있다[1-3]

하지만 대부분의 기존 연구들은 클래스 또는 메소드 간의 의존성(dependency)을 정의하고 그 정의에 따라 대상 클래스가 대상 시스템에서 몇 개의 클래스 또는 메소드와 의존 관계가 존재하는지에 대해서만 보고한다. 즉, 기존의 커플링 척도들은 클래스 간의 의존성 유무만 보여줄 수 있을 뿐 의존성 강도를 측정하지 못한다.

그러므로 기존의 커플링 척도들은 리팩토링(refactoring)[4]이나 시스템 분해(system decomposition)[5]와 같이 클래스 간의 의존성 여부 뿐만 아니라 의존성 강도 정보가 필요한 유지보수 활동에서 의사결정을 지원하는 데 한계가 있다.

이 논문에서는 이러한 문제점을 해결하기 위하여 클래스 간의 의존성 강도를 측정할 수 있는 커플링 척도를 제안한다. 의존성 강도는 척도를 적용할 분야에 따라 기능의 유사성, 유지보수성과 같은 다양한 외부 품질 요소 측면에서 고려될 수 있다. 이 논문에서 제안된 커플링 척도는 유지 보수 활동에서 의사결정을 지원하기 위해 제안되었으며, 그에 따라 의존성 강도를 다양한 외부 품질 요소 중 유지 보수성 측면에서 정의한다.

논문의 구성은 다음과 같다. 다음 장에서는 객체 지향적 커플링 척도에 대한 기존의 관련 연구를 살펴본다. 3장에서는 이 연구에서 제안하는 커플링 척도에 대해 설명하고 4장에서는 시스템 분해 문제에 제안된 척도를 적용하는 예제를 통하여 시스템의 디자인을 개선하기

· 이 논문은 2007 한국컴퓨터종합학술대회에서 '객체 지향 시스템에서의 클래스 간 의존성 강도 측정을 위한 커플링 척도'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 한국과학기술원 전자전산학과
jmhwa@salmosa.kaist.ac.kr
shlee@salmosa.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전자전산학과 교수
kwon@salmosa.kaist.ac.kr

논문접수 : 2007년 10월 2일
심사완료 : 2008년 1월 4일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제1호(2008.2)

위한 여러 분야에 적용될 수 있음을 보인다. 그리고 마지막으로 5장에서는 결론을 맺고 향후 연구에 대해서 논의한다.

2. 관련연구

객체 지향 패러다임에서 커플링 척도에 대한 연구는 활발하게 진행되어 왔고 다양한 접근 방법으로 다양한 관점에서의 커플링 척도들이 제안되었다. 이러한 척도들은 측정 방법에 따라 크게 정적 측정 척도[6,7]와 동적 측정 척도[8,9]로 나눌 수 있다.

기존의 정적 척도들은 두 클래스(혹은 메소드) 간의 메소드 호출이나 변수 참조로 의존 관계를 정의하고 한 클래스(혹은 메소드)가 얼마나 많은 외부 클래스나 메소드와 의존관계에 있는가를 표현할 뿐 의존 관계에 있는 클래스들 각각에 대해서 얼마나 강하게 의존되어 있는지에 대해서는 나타내지 못한다.

예를 들어 가장 널리 사용되는 정적 측정 커플링 척도인 CBO(Coupling Between Objects)[7], RFC(Response for class)[7]를 살펴보면, 두 척도 모두 단순히 의존된 클래스나 메소드, 혹은 메소드 호출의 개수로만 정의됨으로써 한 클래스와 메소드 혹은 클래스들 간의 의존성의 유무만 알려 줄 뿐 어떤 의미에서 얼마나 강하게 의존되어 있는지에 대한 정보를 주지 못한다.

동적 측정 척도들[8,9]은 프로그램 실행을 통해 정적 측정 척도와는 다른 측면의 정보를 제공하기 때문에 정적 척도와 상호보완적으로 사용된다. 이 논문에서는 정적 측정 척도를 제안하기 때문에 동적 측정 커플링 척도에 대해서는 더 이상 언급하지 않는다.

3. 클래스 간의 의존성 강도 측정을 위한 커플링 척도 제안

3.1 이론적 배경

시스템 유지 보수 과정에서 한 클래스를 수정하게 되면 리플 효과(ripple effect)에 의해 수정된 클래스에 의존된 클래스들도 그 영향을 받게 된다[10]. 즉, 시스템이 올바르게 수정되었는지를 테스트하기 위해서는 수정된 클래스뿐만 아니라 그 클래스와 의존된 클래스들까지도 재검사 해야 하며 이런 재검사 비용은 유지 보수 비용의 많은 부분을 차지하고 재검사가 유발될 가능성이 클수록 재검사 비용이 커지기 때문에 유지보수 비용이 높아진다. 그러므로 시스템의 유지보수 비용을 줄이고 유지보수성을 높이기 위해서는 추가적인 재검사 비용을 유발시킬 수 있는 클래스 간의 의존성을 살펴보는 것이 필요하다.

따라서 이 논문에서는 클래스 간의 의존성으로 인해 유발되는 재검사 비용 관점에서 의존성 강도를 측정하고자 한다.

유발되는 재검사 비용을 측정하기 위해서는 첫째로 시스템에 변화가 생길 부분과 변화의 빈도를 알아야 하며, 둘째로 이 변화로 인해 영향을 받는 시스템 부분을 알아야 한다.

첫째로 시스템에 발생하는 변화를 예측하기 위해서는 시스템의 결함을 모두 예측할 수 있어야 할 뿐만 아니라 요구 사항과 시스템 환경의 변화도 예측할 수 있어야 하기 때문에 이는 거의 불가능하다. 그러므로 이 논문에서는 다음과 같이 가정한다.

가정 1. 랜덤유지보수활동

기존의 연구들에서는 시스템 변화 부분과 빈도 예측 문제를 해결하기 위해 "시스템에 대한 유지보수 활동은 어떤 부분에서든 똑같은 확률로 발생한다"라는 "랜덤유지보수활동"을 가정한다[11]. 이 연구에서도 재검사 유발성을 측정하기 위해 랜덤 유지보수 활동을 가정한다.

둘째로 시스템의 변화로 인해 영향을 받는 시스템 부분을 예측하기 위해서는 시스템 부분들 간의 호출관계, 데이타 사용관계 등 다양한 구조적 특성이 고려될 수 있다. 이 논문에서는 멤버 메소드들 간의 호출관계에 의한 클래스 간의 의존성을 살펴보기 때문에 시스템 변화로 인해 영향을 받는 시스템 부분을 다음과 같이 가정한다.

가정 2. 피호출 메소드에서 호출 메소드로의 리플 효과

본 연구에서는 시스템 변화로 인한 영향 예측을 행하는 기존의 연구들에서와 같이 시스템 변화로 인해 영향을 받는 시스템 부분은 변화된 메소드를 호출하는 메소드로 가정한다.

가정 1에 따르면 하나의 메소드에서 유지보수 활동이 일어날 가능성은 메소드의 크기, 즉 SLOC에 비례한다. 그리고 가정 2에 따르면 피호출 메소드를 호출하는 다른 메소드들은 피호출 메소드의 변화에 의해 재검사 되어야 한다. 따라서 호출 메소드의 재검사가 피호출 메소드의 변화에 의해 유발될 가능성은 피호출 메소드의 SLOC로 나타낼 수 있다.

3.2 용어정의

정의 1. 의존 클래스와 피의존 클래스

클래스 c1의 메소드들이 클래스 *c2*의 메소드들을 호출할 때, c1을 의존 클래스, *c2*를 피의존 클래스라 정의한다.

정의 2. 메소드에 대한 관계와 집합

$R(c1, c2)$는 의존 클래스 *c1*이 호출하는 피의존 클래스 *c2*의 메소드들의 집합이다. $AR(c)$은 클래스 *c*에서 호출하는 다른 모든 클래스에 속한 메소드들의 집합으로 정의된다. 자세한 정의는 표 1에 나타나 있다.

표 1 메소드에 대한 관계와 집합 정의

용어	정의
m1→m2	메소드 *m1*과 *m2*에 대해 *m1*이 *m2*를 호출
C(S)	시스템(또는 서브시스템) *S*내의 모든 클래스들의 집합
NOC(S)	시스템(또는 서브시스템) *S*에 속한 클래스의 개수
M(c)	{*m*|*m*은 *c*∈*C(S)*의 메소드}
R(c1, c2)	{ m2 | c1∈C(S) ∧ c2∈C(S) ∧ m1∈M(c1) ∧ m2∈M(c2) ∧ (m1→m2) ∧ (c1≠c2) }
AR(c)	$\bigcup_{x \in C(S)} R(c,x)$

정의 3. MLOC

MS를 메소드의 집합이라 할 때,

$$MLOC(MS) = \sum_{m \in MS} m의\ SLOC$$

3.3 클래스 간의 상대적인 커플링 척도(RCBC)

정의 4. *RCBC*(Relative Coupling Between Classes)

두 클래스 *c1*, *c2* ∈ *C(S)* (단, *c1* ≠ *c2*)에 대하여,

$$\mathbf{0 \le RCBC(c1,c2) = \frac{MLOC(R(c1,c2))}{MLOC(AR(c1))} \le 1}$$

*RCBC(c1, c2)*는 클래스 *c1*이 시스템의 변화에 의해 재검사 유발될 가능성 중 클래스 *c2*에 의해 재검사가 유발될 가능성을 나타내며 0부터 1사이의 값을 가진다. *RCBC(c1, c2)*가 0을 값으로 가지면 클래스 *c1*은 클래스 *c2*의 변화에 의해 재검사가 유발될 가능성이 없음을 나타낸다. 즉, *c1*과 *c2* 사이에 의존성이 존재하지 않는 경우이다. 그리고 *RCBC(c1, c2)*가 1의 값을 가진다는 것은 클래스 *c1*에 재검사를 유발하는 피의존 클래스가 클래스 *c2*밖에 없음을, 즉 *c1*이 *c2*에 대해서만 의존성을 가짐을 나타낸다.

*RCBC(c1, c2)*는 클래스 *c1*과 의존관계에 있는 모든 클래스들 중 클래스 *c2*와의 의존성 강도를 측정하고 있기 때문에 시스템 분해, 리팩토링 등과 같이 클래스 간의 의존성 강도의 측정이 필요한 작업에 유용한 정보를 제공할 수 있다.

3.4 서브시스템 간의 상대적인 커플링 척도(RCBS)

리팩토링, 시스템 분해 등과 같은 유지보수 활동에서는 클래스 간의 의존성 강도를 측정해야 할 뿐만 아니라 여러 개의 클래스를 포함하는 서브시스템들 간의 의존성 강도 역시 측정해야 한다. 따라서 *RCBC*를 확장하여 서브시스템 간의 의존성 강도를 측정하는 커플링 척도를 다음과 같이 정의하였다.

정의 5. *RCBS*(Relative Coupling Between Subsystems)

두 서브시스템 *S1*, *S2*에 대해 *C(S1)*, *C(S2)* ⊂ *C(S)* (단, *S1* ∩ *S2* = *φ*)일 때,

$$\mathbf{0 \le RCBS(S1,S2) = \frac{\sum_{c1 \in S1} \sum_{c2 \in S2} RCBC(c1,c2)}{NOC(S1)} \le 1}$$

*RCBS(S1, S2)*는 서브시스템 *S1*의 각 클래스가 시스템의 변화에 의해 재검사가 유발될 가능성 중 서브시스템 *S2*의 변화에 의해 서브시스템 *S1*의 각 클래스에 재검사가 유발될 가능성의 평균값 나타내며 0부터 1사이의 값을 가진다. *RCBS(S1, S2)*가 0의 값을 가진다는 것은 *S1*에 속한 클래스들의 *S2*의 클래스들에 대한 의존성이 존재하지 않는 경우이다. 그리고 *RCBS(S1, S2)*가 1의 값을 가진다는 것은 서브시스템 *S1* 내의 모든 클래스들이 서브시스템 *S2* 내의 클래스들에 대해서만 의존성을 가짐을 나타낸다.

4. RCBC의 예제

4.1 RCBC 계산과 기존 커플링 척도와의 비교

그림 1은 예제로 사용한 클래스 다이어그램이다. 급여계산시스템 클래스는 의존 클래스이고 사원정보관리시스템 클래스와 성과평가시스템 클래스는 급여계산시스템 클래스에 대한 피의존 클래스이며, 급여계산시스템 클래스가 호출하는 메소드들만 다이어그램에 표현하고 있다. 그림 2에는 급여계산시스템 클래스의 메소드들과

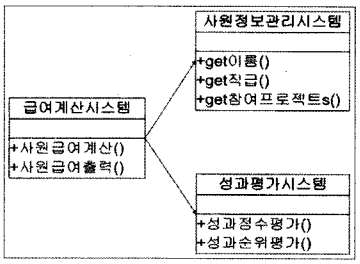

그림 1 예제 클래스 다이어그램

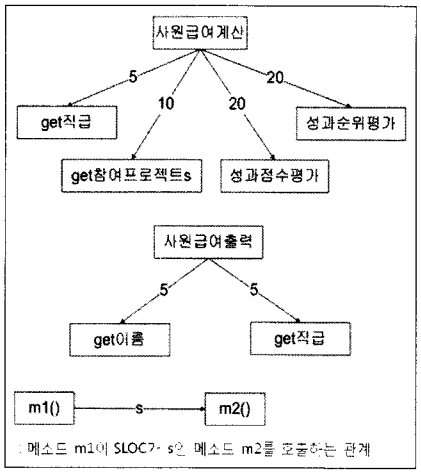

그림 2 메소드들 간의 호출 관계

다른 클래스의 메소드들 간의 호출 관계가 나타나 있다.

이를 바탕으로 각 클래스 간의 *RCBC* 계산을 위해 분자 값을 구하면 다음과 같다.

MLOC(R(급여계산시스템, 사원정보관리시스템))
= MLOC({get이름(), get직급(), get참여프로젝트s()}))
= 2 + 2 + 10 = 14
MLOC(R(급여계산시스템, 성과평가시스템))
= MLOC({성과점수평가(), 성과순위평가()})
= 20 + 20 = 40

그리고 분모 값을 구하면 다음과 같다.

MLOC(AR(급여계산시스템))
= MLOC({R(급여계산시스템, 사원정보관리시스템)
∪R(급여계산시스템, 성과평가시스템)})
= 14 + 40 = 54

이를 이용하여 *RCBC* 값을 구하면 아래와 같다(단, 소수점 셋째 자리에서 반올림 한다).

$$RCBC(\text{급여계산시스템, 사원정보관리시스템}) = \frac{MLOC\left(R(\text{급여계산시스템, 사원정보관리시스템})\right)}{MLOC\left(AR(\text{급여계산시스템})\right)} = \frac{14}{54} = 0.26$$

$$RCBC(\text{급여계산시스템, 성과평가시스템}) = \frac{MLOC\left(R(\text{급여계산시스템, 성과평가시스템})\right)}{MLOC\left(AR(\text{급여계산시스템})\right)} = \frac{40}{54} = 0.74$$

위에서 구한 *RCBC*와 함께 그림 1의 예제에서 급여계산시스템 클래스와 다른 두 클래스 간의 의존성 강도를 CBO, RFC의 관점에서 구해보면 아래의 표 2와 같다.

표 2 클래스 간의 CBO, RFC, RCBC 값

	CBO	RFC	RCBC
급여계산시스템 - 사원정보관리시스템	1	5	0.26
급여계산시스템 - 성과평가시스템	1	4	0.74

그림 1의 예제에서 급여계산시스템 클래스의 성과평가시스템 클래스에 대한 의존도가 상대적으로 더 높다는 것을 직관적으로 알 수 있다. 그러나 표 2의 결과에 따르면 그림 1의 예제와 같은 경우에 CBO와 RFC의 관점에서는 클래스 간에 존재하는 의존성 강도를 제대로 나타내지 못함을 알 수 있다. 반면 이 논문에서 제안한 *RCBC*는 피의존 클래스에 의한 의존 클래스의 재검사 유발 가능성 관점에서 클래스들 간에 존재하는 의존성 강도 측정함으로써 직관에 맞는 결과를 나타내고 있다.

4.2 *RCBC*의 시스템 분해 적용

*RCBC*가 의존성 강도가 필요한 분야에서 의사결정에 유용한 정보를 제공해 줄 수 있음을 그림 3의 "도서 예약 시스템"의 클래스 다이어그램(예약접수 클래스의 호출 함수와 피호출 함수만 표기)

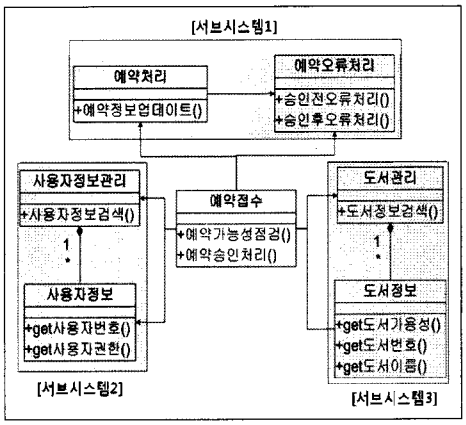

그림 3 "도서 예약 시스템"의 클래스 다이어그램(예약접수 클래스의 호출 함수와 피호출 함수만 표기)

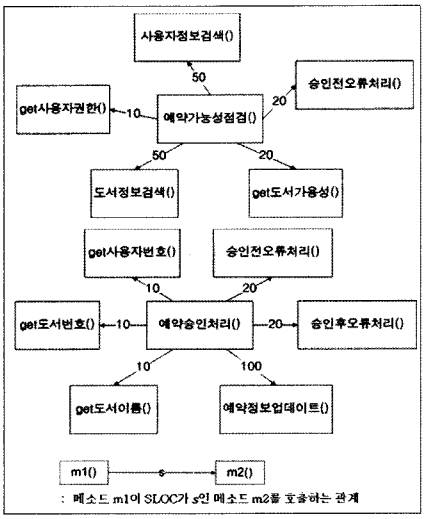

그림 4 메소드들 간의 호출 관계

약 시스템"의 시스템 분해 예제를 통해 알아볼 수 있다. 이 예제에서는 클래스 간의 유사성을 클래스 간의 의존성으로 측정하고, 서브시스템 간의 유사성은 각 서브시스템에 속한 클래스 간의 유사성을 바탕으로 WPGMA [12]를 이용해 측정한다.

그림 3의 클래스 다이어그램에는 총 7개의 클래스와 현재 형성 과정 중인 3개의 서브시스템이 나타나 있으며 서브시스템 간의 상호 관계는 표시되어 있지 않다. 그리고 각 클래스에는 예약접수 클래스에 의해 사용되는 메소드들만 표시되어 있으며 예약접수 클래스의 메소드들과 피호출 메소드들 간의 호출 관계와 피호출 메소드들의 SLOC가 그림 4에 나타나 있다.

이를 바탕으로 예약 접수 클래스와 피의존 클래스 간의 *RCBC* 값을 다음과 같이 구할 수 있다.

RCBC(예약접수, 예약처리) = 0.33
RCBC(예약접수, 예약오류처리) = 0.13
RCBC(예약접수, 사용자정보관리) = 0.17
RCBC(예약접수, 사용자정보) = 0.07
RCBC(예약접수, 도서관리) = 0.17
RCBC(예약접수, 도서정보) = 0.13

그리고 예약접수 클래스와 각 서브시스템 간의 *RCBC*를 기반으로 한 의존성 강도를 구하면 다음과 같다. 서브시스템 예약접수시스템에 대해 C(예약접수시스템) = {예약접수}일 때,

RCBS(예약접수시스템, 서브시스템1) = (0.33+0.13)/1 = 0.46
RCBS(예약접수시스템, 서브시스템2) = (0.17+0.07)/1 = 0.24
RCBS(예약접수시스템, 서브시스템3) = (0.17+0.13)/1 = 0.30

결과적으로 예약접수 클래스와 서브시스템1 간의 의존도가 0.46으로 가장 크게 나왔으며, 이를 예약접수 클래스가 서브시스템1에 포함되어야 한다고 판단하는 근거로 사용할 수 있다.

이 예제를 통해 볼 수 있듯이 *RCBC*는 클래스 간의 의존성 강도를 측정함으로써 서브시스템을 판별하는 데 적용되어 유용한 정보를 제공해 줄 수 있다.

5. 결론 및 향후 과제

클래스 간 의존성 강도를 측정하기 위해 재검사 유발성-유지 보수 비용 측면에서 새로운 커플링 척도 *RCBC*를 제안하였다. *RCBC*는 기존의 커플링 척도들 과는 달리 의존 관계에 있는 클래스들 간의 의존성 강도도 측정할 수 있다. 따라서 시스템 분해, 리팩토링 등과 같이 의존성 강도 측정이 요구되는 분야에서 의사 결정에 유용한 정보를 제공할 수 있으며 클래스의 품질을 평가하는 기준으로도 사용 가능하다.

그리고 예제를 통해 *RCBC* 값을 구하는 과정을 설명함으로써 *RCBC*에 대한 이해를 직관적으로 도왔다. 또, 소프트웨어 품질 개선 분야들 중 시스템 분해 문제에 *RCBC*를 이용하는 예제를 통해서 *RCBC*가 디자인 개선을 위한 의사결정에 도움을 줄 수 있음을 보였다.

현재 가장 우선적인 향후 과제는 실험을 통한 *RCBC*의 검증이라 할 수 있다. 이를 위해 자동으로 *RCBC*를 측정할 수 있는 도구를 구현하고 있으며 이 도구를 이용하여 실제 산업 시스템에 *RCBC*를 적용하여 그 유용성을 검증할 계획이다.

그리고 현재 *RCBC*는 클래스 간의 메소드 호출 관계만 고려하였다. 향후 연구에서는 객체지향 패러다임에서 중요한 특징인 상속, 다형성 등을 반영할 수 있도록 확장할 것이다.

참 고 문 헌

[1] L.C. Briand, J. Wuest, and H. Louinis, "Using Coupling Measurement for Impact Analysis in Object-Oriented Systems," *Proc. of IEEE Int'l Conf. on Software Maintenance*, pp. 475-482, 1999.

[2] P. Yu, T. Systa, and H. Muller, "Predicting Fault-proneness Using OO Metrics. An Industrial Case Study," *The Journal of Syst. and Software*, Vol.52, pp. 99-107, 2002.

[3] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," *IEEE Trans. on Software Eng.*, Vol.31, No.10, pp. 897-910, 2005.

[4] T. Mens, and T. Tourwe, "A Survey of Software Refactoring," *IEEE Trans. on Software Eng.*, Vol. 30, No.2, pp. 126-139, 2004.

[5] C.-H. Lung, M. Zaman, and A. Nandi, "Applications of Clustering Techniques to Software Partitioning, Recovery and Restructuring," *The Journal of Syst. And Software*, Vol.32, No.2, pp. 227-224, 2004.

[6] L.C. Briand, J. Daly, and J. Wüst, "A Unified Framework for Coupling Measurement in Object Oriented Systems," *IEEE Trans. on Software Eng.*, Vol.25, No.1, pp. 91-121, 1999.

[7] S.R. Chidamber, and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Software Eng.*, Vol.20, No.6, pp. 476-493, 1994.

[8] E. Arisholm, L.C. Briand, and A. Føyen, "Dynamic Coupling Measurement for Object-Oriented Software," *IEEE Trans. on Software Eng.*, Vol.30, No.8, pp. 491-506, 2004.

[9] S. Yacoub, H. Ammar, and T. Robinson, "Dynamic Metrics for Object-Oriented Designs," *Proc. 6th Int'l Symp. Software Metrics*, pp. 50-61, 1999.

[10] D.C. Kung, J. Gao, P. Hsia, Y. Toyoshima, and C. Chen, "On Regression Testing of Object-Oriented Software Maintenance," *The Journal of Syst. and Software*, Vol.32, No.1, pp. 21-40, 1996.

[11] R. Leitch and E. Stroulia, "Assessing the Maintainability Benefits of Design Restructuring using Dependency Analysis," *Proc. 9th Int'l Symp. Software Metrics*, pp. 309-322, 2003.

[12] N. Anquetil, and T. C. Lethbridge, "Comparative Study of Clustering Algorithms and Abstract Representations for Software Remodularisation," Proc. Softw., Vol.150, No.3, pp. 185-201, 2003.