# SIFG를 이용한 프로그램 복잡도 척도

## (A Program Complexity Measure using the Slice-based Information Flow Graph(SIFG))

최 완 규 † 정 일 용 †† 이 성 주 †††

(WanKyoo Choi) (IlYong Chung) (SungJoo Lee)

**요 약** 본 논문은 프로그램 슬라이스에서의 데이타 토큰들의 정보 흐름에 기초하여 프로그램에서의 정보 흐름을 모델링하기 SIFG(Slice-based Information Flow Graph)를 개발하였다. 다음으로, SIFG에서의 정보 흐름의 복잡도 측정를 통해서 프로그램의 복잡도를 측정하기 위해 SCM(Slice-based Complexity Measure)을 정의하였다. Zuse의 방법에 따라, 본 연구에서는 SIFG에서의 극소 수정(atomic modification)을 통해 SCM이 순서척도가 됨을 보여주었고, 이항 연산 MBSEQ에 대해서 SCM이 가법성을 만족함을 보여주었고, 이항 연산 MBALT에 대해서는 Zuse의 가법성을 만족하지 않지만 Weyuker의 9번째 공리를 만족함을 보여주었다. 또한 기존 척도들과의 비교를 통해서, SCM이 프로그램 내에서의 제어와 데이타 흐름뿐만 아니라 프로그램의 물리적 크기를 반영하는 측정이 이루어진다는 것을 보여주었다.

***Abstract*** We developed a SIFG(Slice-based Information Flow Graph) for modeling the information flow on program on the basis of the information flow of data tokens on program slices. Then we defined a SCM(Slice-based Complexity Measure) for measuring the program complexity by measuring the complexity of information flow on SIFG. We showed that, according to Zuse's approach, it assumed ordinal scale based on atomic modifications on SIFG, and that it was additive to binary operation MBSEQ, and that it was not additive to binary operation MBALT but satisfied Weyuker's 9th axiom. Also, based on comparison with the existing measures, we showed that SCM could measure not only the control and data flow in program but also the physical size of program.

## 1. 서 론

전체 시스템 비용에서 소프트웨어 비용의 증가에 따라서, 소프트웨어의 심리적 복잡도의 측정에 대한 관심이 증가해왔다. 소프트웨어 개발 단계에서 소프트웨어 측정은 프로그래머에게 중요한 정보를 제공할 수 있고, 이해하기 쉬운 코드의 작성에 도움이 될 수 있다. 그러나 측정 분야에서 대부분의 기존의 메트릭들은 프로그램의 표면적인 특징들만을 고려하고 있다고 말할 수 있다. 프로그램의 이해에 지대한 영향을 미치는 것은 프로그램의 표면적인 특징들이 아니라 프로그램의 정보 흐름(information flow)의 특징이다.

† 정 회 원 : 광주대학교 컴퓨터전사통신공학부 교수
wkchoi@kwangju.ac.kr
†† 종신회원 : 조선대학교 컴퓨터공학부 교수
iyc@mail.chosun.ac.kr
††† 비 회 원 : 조선대학교 컴퓨터공학부 교수
sjlee@mail.chosun.ac.kr
논문접수 : 2000년 9월 22일
심사완료 : 2001년 9월 14일

프로그램 이해에 대한 대부분의 어려움은 프로그래머가 프로그램의 이해를 위해서 사용하는 문(statement)들의 묶음(grouping)과 소스 프로그램 리스트와 프로그래밍 환경이 제시하는 문들의 묶음과의 차이 때문이다[1]. Weiser[2]에 의하면 프로그래머들이 프로그램을 이해할 때 순차적 관계 이외의 방법으로 문들을 묶는 경향이 있다고 하였다. 일반적으로 묶음을 위한 기준은 정보의 흐름인 데이타와 제어의 흐름에 관계가 있다. 이런 정보가 슬라이스(slice)[3]에 분명하게 표현되므로, 정보 흐름의 복잡도는 슬라이스를 이용하여 가장 쉽고 분명하게 측정될 수 있다.

슬라이스는 프로그램의 특정 위치에서 한 변수의 값에 영향을 미치는 문들의 묶음에 대한 추상화이다. 프로그래머가 프로그램을 이해하고 디버깅할 때 슬라이스를 사용한다면[2], 프로그램의 이해는 슬라이스에서의 정보 흐름을 탐색하는 과정이라고 할 수 있다. 따라서 슬라이

스 복잡도의 측정은 의미가 있다.

그러므로 본 연구에서는 슬라이스에서의 정보 흐름의 복잡도 측정을 통하여 프로그램에서의 정보 흐름의 복잡도를 측정하는 슬라이스 기반 복잡도 척도(Slice-based Complexity Measure: SCM)을 제안한다.

슬라이스 기반 복잡도 척도가 의미 있는 측정을 제공하기 위해서는 척도가 엄밀하게 정의되어야하고, 측정을 위한 속성을 정확히 반영해야하고, 이런 속성들을 포착하는 모델에 근거해야한다[4]. 따라서 본 연구에서는 정보 흐름에 따른 미세 변화(elementary change)들을 잘 표현할 수 있도록 슬라이스 상의 데이타 토큰들에서의 정보 흐름을 통해 프로그램의 정보 흐름을 모델링하기 위해 슬라이스 기반 정보 흐름 그래프(Slice based information flow graph: SIFG)를 개발한다. SIFG는 데이타 슬라이스에서 정보의 흐름 과정을 분명하게 보여주므로, 관심 있는 변수 값의 변화 과정을 쉽게 탐색할 수 있고, 프로그램의 이해를 증진시킬 수 있다.

SCM은 측정 모델에서 정량적인 값으로의 사상을 명시한다. SCM 측정값은 복잡도 순서와 일치해야한다. 따라서 본 연구에서는 SCM이 슬라이스 기반 복잡도 모델 순서와 일치함을 보이고, SCM의 척도특성(scale property)들을 결정함으로써 SCM을 검증한다. 척도의 척도유형(scale type)은 매우 중요하고 측정 값 사이에서 의미 있게 수행될 수 있는 산술 연산을 결정한다[5, 6].

본 논문은 다음과 같이 구성된다. 2장에서는 프로그램 슬라이스에 대해여 고찰하고, 3장에서는 슬라이스 상의 데이타 토큰들에서의 정보 흐름을 통해 프로그램의 정보 흐름을 모델링하기 위해 슬라이스 기반 정보 흐름 그래프와 슬라이스 기반 복잡도 척도를 정의한다. 4장에서는 제안된 척도의 척도특성을 평가한다. 5장에서는 기존 척도들과의 상관관계 분석을 통해서 제안된 척도를 고찰한다. 6장에서 결론을 제시한다.

## 2. 프로그램 슬라이스

슬라이싱은 Weiser에 의해 소개된 프로그램 축소 방법으로 디버깅 도구와 프로그램 이해를 위한 보조 수단으로 제안되었고, 종래의 흐름 분석의 "사용"관계를 포착한다. 슬라이스는 다음과 같이 정의된다[7].

슬라이스 기준은 순서쌍 $C=<i, V>$로 표현된다. 여기서, $i$는 모듈 M에서 특정 문장의 번호이고, $V$는 M에서의 변수들의 집합이다. 슬라이스 기준 $C$에서 모듈 M의 모듈내 슬라이스(intramodule slice) S는 다음 특성을 갖는 임의의 실행 가능한 모듈이다: (1) S는 M에서 임의의 문장들을 제거하여 획득될 수 있다. (2) 임의의 주어진 입력에 대해서, 슬라이스 S에서 모든 점들에서 실행은 문장 $i$에서 $V$의 변수들에 관하여 모듈 M에서 관찰된 것과 동일하다.

예를 들어서, 그림 1의 *Sum1* 프로시저에서 슬라이싱 기준 $C=<T_{14}, \{sumX\}>$와 $C=<T_{14}, \{sumSqrX\}>$로 획득되는 슬라이스는 그림 2와 그림 3과 같다.

```
T1:  void Sum1(int N1, int *X1, int *sumX1, int *sumSqrX1)
T2:  {
T3:        int  I1;
T4:        *sumX2 = 01;
           for(T5: I2 = 11;
               T6: I3 < N2;
               T8: I5++)
T7:            *sumX3 = sumX3 + X2[I4];
T9:        *sumSqrX2 = 02;
           for(T10: I6 = 12;
               T11: I7 < N3;
               T13: I10++)
T12:           *sumSqrX3 = *sumSqrX3 + X3[I8] * X4[I9];
T14: }
```

그림 1 Sum1 프로시저

```
T1:  void Sum1(int N1, int *X1, int *sumX1)
T2:  {
T3:        int  I1;
T4:        *sumX2 = 01;
           for(T5: I2 = 11;
               T6: I3 < N2;
               T8: I5++)
T7:            *sumX3 = sumX3 + X2[I4];
```

그림 2 C=<$T_{14}$, {sumX}>으로 획득된 슬라이스

```
T1:  void Sum1(int N1, int *X1, int *sumSqrX1)
T2:  {
T3:        int  I1;
T9:        *sumSqrX2 = 02;
           for(T10: I6 = 12;
               T11: I7 < N3;
               T13: I10++)
T12:           *sumSqrX3 = *sumSqrX3 + X3[I8] * X4[I9];
T14: }
```

그림 3 C=<$T_{13}$, {sumSqrX}>으로 획득된 슬라이스

[8]에서, 변수의 "사용"과 "사용됨" 관계를 설명하는 메트릭 슬라이스라 불리는 새로운 슬라이싱 형태가 정의되었다. 메트릭 슬라이스는 VRES(varable-referent executable statement)에 기초하여 모듈의 출력 변수들에 대해서 계산된다.

Bieman[8, 9]은 데이타 토큰을 사용하기 위해서 메트

릭 슬라이스를 수정한 데이타 슬라이스(data slices)에 근거한 응집도 척도를 제안했다. 데이타 슬라이스에서 데이타 토큰은 문장에서 정의된 변수와 상수정의를 의미한다. 데이타 슬라이스 역시 각 출력에 대해서 계산된다.

그림 1의 프로시저 Sum1에서 출력 sumX와 sum SqrX에 대한 데이타 슬라이스는 그림 2와 3에 나타나는 일련의 데이타 토큰들의 집합이다. 즉,

*DataTokens*(Sum1) = {$N_1$, $X_1$, sum$X_1$, sumSqr$X_1$, $I_1$, sum$X_2$, $0_1$, $I_2$, $1_1$, $I_3$, $N_2$, $I_4$, sum$X_3$, sum$X_4$, $X_2$, $I_5$, sumSqr$X_2$, $0_2$, $I_6$, $1_2$, $I_7$, $N_3$, $I_8$, sumSqr$X_3$, sumSqr$X_4$, $X_3$, $I_9$, $X_4$, $I_{10}$}.

*DataSlice*(sumX) = {$N_1$, $X_1$, sum$X_1$, $I_1$, sum$X_2$, $0_1$, $I_2$, $1_1$, $I_3$, $N_2$, $I_4$, sum$X_3$, sum$X_4$, $X_2$, $I_5$}.

*DataSlice*(sumSqrX) = {$N_1$, $X_1$, sumSqr$X_1$, $I_1$, sumSqr$X_2$, $0_2$, $I_6$, $1_2$, $I_7$, $N_3$, $I_8$, sumSqr$X_3$, sumSqr$X_4$, $X_3$, $I_9$, $X_4$, $I_{10}$}.

그러나 위와 같은 메트릭 슬라이스나 데이타 슬라이스는 정보 흐름을 반영하지 못하므로, 슬라이스에서 데이타 토큰의 정보 흐름을 표현할 수 있는 슬라이스 표현이 필요하다. 따라서, 본 연구에서는 슬라이스 상의 데이타 토큰들에서의 정보 흐름을 모델링하기 위한 슬라이스 기반 정보 흐름 그래프(SIFG)를 개발하였다.

## 3. 슬라이스 기반 복잡도 척도

### 3.1 슬라이스 기반 정보 흐름 그래프(SIFG)

슬라이스 근거하여 프로그램 복잡도를 측정하기 위하여, 본 연구에서는 슬라이스 기반 정보 흐름 그래프(Slice-based Information Flow Graph: SIFG)를 정의한다. SIFG는 프로시저의 출력 변수에 대한 슬라이스 내의 데이타 토큰들의 정보 흐름을 모델링한다.

SIFG(S)로 표현되는 슬라이스 S에 슬라이스 기반 정보 흐름 그래프는 방향성 그래프, SIFG(S)=<N, E>이다. 여기서, N은 슬라이스 S에 포함된 데이타 토큰들의 집합이고, E는 데이타 토큰들 간의 정보 흐름을 나타내는 간선들의 집합이다

단일 슬라이스에서의 SIFG(S)는 다음과 같은 단계를 거쳐서 생성된다.

**Step 1**: 문장 내에서 사용되는 데이타 토큰들 간의 "정보 흐름" 관계에 따라서 슬라이스 내의 모든 문장들을 정보흐름 간선들의 집합으로 표현한다.

$d_i$, $d_j \in N$이 하나의 문장에서 나타나는 데이타 토큰이라 하자. 문장 내에서 $d_i$를 사용하는 연산의 결과가 $d_j$에 배정되면 ***$d_j$는 $d_i$를 직접 사용한다***고 하고, $d_i \rightarrow d_j$로 표현한다. 그렇지 않으면, 즉 비교연산 등이 사용되는 경우는 ***$d_j$는 $d_i$를 간접 사용한다***고 하고, $d_i \dashrightarrow d_j$로 표현한다. $d_j$가 $d_i$를 직접 또는 간접 사용하면 ***$d_i$에서 $d_j$로 직접 또는 간접적으로 정보가 흐른다***라고 한다.

$N(T_k)$를 슬라이스 내의 문장 $T_k$에 나타나는 모든 데이타 토큰들의 집합이라 하자. 이때, 임의의 문장 $T_k$ 내의 데이타 토큰들간의 정보흐름 간선들의 집합 $E(T_k)$는 다음과 같다.

$$E(T_k) = \{d_i \rightarrow d_j \text{ or } d_i \dashrightarrow d_j \mid d_i, d_j \in N(T_k)\}$$

**Step 2**: 동일 변수에 속하는 데이타 토큰들간의 "정보 흐름" 관계에 따라서 슬라이스 내의 임의의 변수들에 관한 정보 흐름 간선들의 집합을 구한다.

$N(v)(N(v) \in N)$를 슬라이스 내에서 사용되는 임의의 변수 $v$에 대한 데이타 토큰들의 집합이라 하자. $d_i$, $d_j \in N(v)$일 때, $d_i \in N(T_p)$이고 $d_j \in N(T_q)$이고, $p<q$이면 ***$d_j$는 $d_i$를 직접 사용한다***(즉, ***$d_i$에서 $d_j$로 직접적으로 정보가 흐른다***)라고 한다.

이때, 슬라이스 내의 임의의 변수 $v$에 대한 데이타 토큰들간의 정보 흐름 간선들의 집합 $E(v)$은 다음과 같다.

$$E(v) = \{d_i \rightarrow d_j \mid d_i, d_j \in N(v),\ i \neq j\}$$

**Step 3**: $E(T_1), \cdots, E(T_n)$와 $E(v_1), \cdots, E(v_m)$를 결합하여 슬라이스 S에 대한 SIFG(S)를 생성한다.

슬라이스 S에 대한 SIFG(S)에서 간선들의 집합은 $E=E(T_1) \cup \cdots \cup E(T_n) \cup E(v_1) \cup \cdots \cup E(v_m)$이고, $d_i$, $d_j$, $d_k \in E$일 때, 다음 조건들을 만족하면 SIFG(S)는 $d_i$에서 $d_j$로의 정보흐름 간선(즉, $d_i \rightarrow d_j$ 또는 $d_i \dashrightarrow d_j$)을 포함한다.

1) $d_i$에서 $d_j$로 정보가 흐른다.
2) $d_i$에서 $d_k$로 그리고 $d_k$에서 $d_j$로 정보가 흐르는 노드 $d_k$가 존재하지 않는다

**Step 4**: 슬라이스 내에 반복문이 있는 경우, 반복문 내에서 사용되는 동일 변수에 관한 데이타 토큰들간의 정보흐름을 추가한다.

$SIFG_{loop}(v)$를 반복문 구조 내에서 위치한 문장들에서 사용되는 변수 $v$에 관한 데이타 토큰들로부터 생성되는 SIFG(S)의 부분그래프(sub-graph)라 할 때, $SIFG_{loop}(v)$ 내에서 $v$의 값이 연산의 결과로 변하는 경우 종단노드 $d_i$에서 시작노드 $d_j$로의 직접사용 간선(즉, $d_i \rightarrow d_j$)을 추가한다.

예를 들어서, 그림 2의 슬라이스에 대해서, 각 문장들에 대한 노드들의 집합은 다음과 같다.

$N(T_1)$ = {$N_1$, $X_1$, sum$X_1$}
$N(T_3)$ = {$I_1$}
$N(T_4)$ = {sum$X_2$, $0_1$}
$N(T_5)$ = {$I_2$, $1_1$}
$N(T_6)$ = {$I_3$, $N_2$}

$N(T_7)$ = {$sumX_3$, $sumX_4$, $X_2$, $I_4$}
$N(T_8)$ = {$I_5$}

또한, 각 문장들을 위한 간선들의 집합은 다음과 같다.

$E(T_4)$ = {$0_1 \rightarrow sumX_2$}
$E(T_5)$ = {$1_1 \rightarrow I_2$}
$E(T_6)$ = {$N_2 \cdots \rightarrow I_3$}
$E(T_7)$ = {$sumX_4 \rightarrow sumX_3$, $X_2 \rightarrow sumX_3$, $I_4 \rightarrow X_2$}
$E(T_8)$ = {$I_5 \rightarrow I_5$}

사용되는 변수들에 대한 간선들의 집합은 다음과 같다.

$E(N)$ = {$N_1 \rightarrow N_2$}
$E(X)$ = {$X_1 \rightarrow X_2$}
$E(I)$ = {$I_1 \rightarrow I_2$, $I_2 \rightarrow I_3$, $I_3 \rightarrow I_4$, $I_4 \rightarrow I_5$}
$E(sumX)$ = {$sumX_1 \rightarrow sumX_2$, $sumX_2 \rightarrow sumX_3$, $sumX_2 \rightarrow sumX_4$}

위의 문장들에 대한 간선들의 집합과 변수들에 대한 간선들의 집합에서 생성된 슬라이스 기반 정보 흐름 그래프에 반복분 $T_6 \sim T_8$에서 사용되는 변수들 중에서 연산의 결과로 값이 변하는 변수들은 I와 SumX이므로 I와 SumX에 관한 데이타 토큰들간의 정보 흐름간선(즉, $I_5 \rightarrow I_3$, $sumX_3 \rightarrow sumX_4$)을 추가한 그림 2에 대한 SIFG(sumX)는 그림 4와 같다. 또한 그림 3에 대한 SIFG(sumSqrX)는 그림 5와 같다.

프로시저 P에서 생성되는 모든 슬라이스에 대한 SIFG의 결합을 통해서 프로시저 P에 대한 SIFG(P)는 다음과 같이 정의된다.

**정의 1**: $N(S_i)$와 $E(S_i)$가 $SIFG(S_i)$의 노드와 간선들의 집합일 때, 프로시저 P에 대한 SIFG(P)는 다음과 같다.

$SIFG(P)=<N_{S1} \cup N_{S2} \cup \cdots \cup N_{Sn}, E_{S1} \cup E_{S2} \cup \cdots \cup E_{Sn}>$

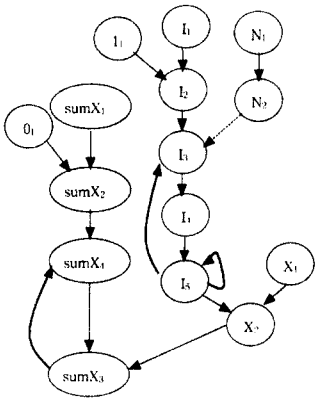

그림 4 SIFG(*sumX*)

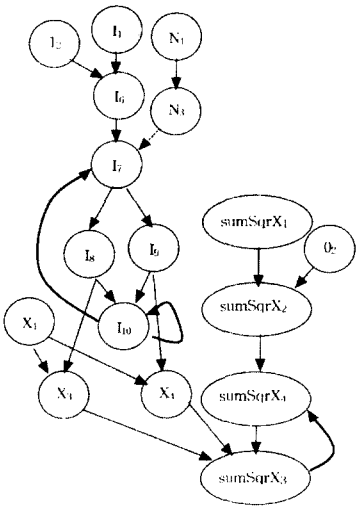

그림 5 SIFG(*sumSqrX*)

그림 4와 5를 결합한 *Sum1* 프로시저에 대한 SIFG는 그림 6과 같다.

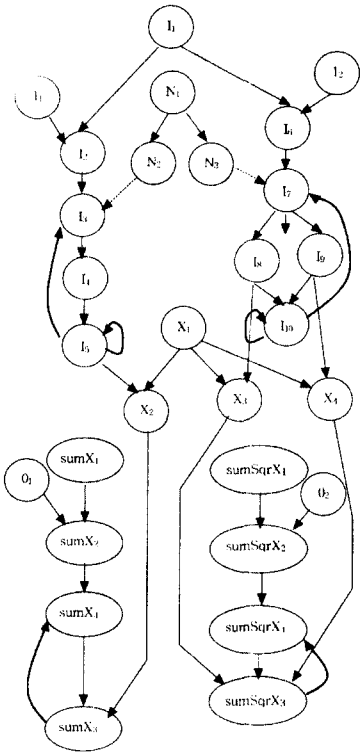

그림 6 SIFG(*Sum1*)

SIFG의 노드로서 데이타 토큰들을 사용함으로써, 정보 흐름에 따른 미세 변화들이 SIFG에 반영되고 프로그램 복잡도에 영향을 미치게된다. 이러한 변화들에는 데이타 토큰의 추가와 삭제, 주어진 정황에서 데이타 값 흐름의 변화 등이 있다.

### 3.2 슬라이스 기반 복잡도 척도(SCM)

프로그램 이해와 디버깅에 슬라이스를 사용한다면[2], 프로그램 이해는 슬라이스에서의 정보 흐름을 탐색하는 과정이라고 할 수 있다. 즉, 슬라이스의 모든 점들에서 실행의 과정을 예측하는 것으로 볼 수 있다. 따라서 특정 시점에서 실행의 예측을 위한 많은 선행 조건들이 요구된다면 그 시점에서의 프로그램의 이해는 어렵고 복잡하게된다.

따라서 본 연구에서는 SIFG의 임의의 노드의 내차수와 간접사용 간선의 개수에 의해 프로그램의 복잡도를 측정하는 슬라이스 기반 복잡도 척도(Slice-based Complexity Measure: SCM)를 다음과 같이 정의한다.

**정의 2**: 슬라이스 S에 대한 SIFG(S)에서, N과 E는 SIFG(S)의 노드와 간선들의 집합이고, $E_{dirt}$($E_{dirt}\subset$E)는 직접 사용 간선들의 집합이고, $E_{ind}$($E_{ind}\subset$E)는 간접사용 간선들의 집합이고, E=$E_{dirt}\cup E_{dirt}$이고, $d_i$($d_i\in$N)에 대한 $ind(d_i)$는 $d_i$로 향하는 직접사용 간선과 간접사용 간선들 개수, 즉 내차수(indegree)라 할 때, 슬라이스 기반 복잡도는 다음과 같다.

$$SCM=|E_{ind}|+\sum_{i=1}^{|N|}(ind(d_i),\ \text{if}\ ind(d_i)\geq 2)$$

그림 4, 5와 6에 대한 $SCM$ 측정값은 다음과 같다.

$SCM(sumX)$ = 15+ 1 = 16

$SCM(sumSqrX)$ = 19 + 1 = 20

$SCM(Sum1)$ = 34 + 2 = 36

그림 4에서, 간접 사용 간선은 $N_2\rightarrow I_3$이므로 $|E_{ind}|$=1이다. 또한, $I_2$, $I_3$, $I_5$, $sumX_2$, $sumX_3$, $sumX_4$, $X_2$가 내차수가 2이상인 노드이므로, $ind(I_2)$=2, $ind(I_3)$=3, $ind(I_5)$=2, $ind(sumX_2)$=2, $ind(sumX_3)$=2, $ind(sumX_4)$=2, $ind(X_2)$=2이다. 따라서 $SCM(sumX)$ =1+2+3+2+2+2+2=16이다.

## 4. Validation

연구자들은 소프트웨어 척도의 원하는 다양한 특성(property)들을 제안하였다. 그러한 프레임워크는 새로운 척도의 연구를 위한 지침이 된다. 또한 기존의 척도들은 이 특성들에 근거하여 정당화된다[10].

Zuse[5]는 소프트웨어 척도들이 그들의 척도 특성(scale property)에 의해 검증될 수 있다고 하였고, 측정이론 방법론을 복잡도 척도에 적용하여, 소프트웨어 척도들이 어떤 유형의 스케일을 가정하는 가를 보여주었다. Zuse는 제어 흐름 그래프(Control Flow Graph: CFG)에 근거한 척도들이 가정하고 있는 스케일 속성들을 분석하였다.

따라서 본 연구에서는 슬라이스 복잡도 척도를 검증하기 위해서 Zuse의 방법에 근거하여, $SCM$이 직관과 일치하는 순서척도임과 $SCM$이 비율척도의 요구사항을 가정하고 있음을 보여준다.

### 4.1 순서척도(ordinal scale)

Zuse에 의하면, SCM이 순서척도가 되기 위해서는 경험 관계(empirical relation) 또는 뷰포트(viewport)라 불리는 SCM에 대한 직관은 세 가지 공리를 만족해야한다: 반사성, 이행성, 완전성[5]. 이것들은 약순서(weak order)의 필요충분 조건들이다. Zuse의 정의에 따라서, 슬라이스 기반 정보 흐름그래프로 표현되는 프로시저($\boldsymbol{P}_1$, $\boldsymbol{P}_2\in\boldsymbol{P}$)들에서의 이진 관계로 슬라이스 기반 복잡도 뷰포트를 다음과 같이 정의한다.

$\boldsymbol{P}_1 \bullet > \boldsymbol{P}_2$ $\boldsymbol{P}_1$이 $\boldsymbol{P}_2$보다 더욱 복잡하다.

$\boldsymbol{P}_1 \bullet = \boldsymbol{P}_2$ $\boldsymbol{P}_1$과 $\boldsymbol{P}_2$의 복잡도는 같다.

$\boldsymbol{P}_1 \bullet \geq \boldsymbol{P}_2$ $\boldsymbol{P}_1 \bullet > \boldsymbol{P}_2$ 혹은 $\boldsymbol{P}_1 \bullet = \boldsymbol{P}_2$이다.

슬라이스 기반 복잡도 뷰포트에 대한 일반적인 정의를 내리는 것은 가능하지 않기 때문에 원소 뷰포인트(*elementary viewpoint*)라 불리는 위의 관계의 부분집합을 사용할 수 있다[6]. 원소 뷰포인트는 $\boldsymbol{P}$에서의 극소수정(atomic modification)[5]들의 유한 집합에 의하여 정의된다.

SCM이 순서척도임을 보여주기 위해서, 그것이 극소수정과 일치함을 보여줄 필요가 있다[6]. 따라서 본 연구에서는 P에서의 극소 수정들의 직관적으로 분명한 효과에 의하여 SCM의 복잡도 순서를 평가한다.

극소 수정은 $\boldsymbol{P}$에서 간선들이나 노드들의 첨가, 삭제, 이동과 같은 임의의 수정에 의해 $\boldsymbol{P}$의 $\boldsymbol{P}'$로의 변환이므로, 극소 수정은 복잡도를 고려하지 않고 P에 적용될 수 있다.

모든 척도들은 극소 수정들에 대해서 특별한 반응을 보이는데, 이것을 척도의 부분특성(partial properties)이라 한다. SCM 또한 다음 원자 수정에 대해서 특별한 반응을 보인다.

**AM**$_1$: $\boldsymbol{P}$의 임의의 위치에 하나의 노드와 하나의 직접 또는 간접 사용 간선을 추가한다. 결과로서 $\boldsymbol{P}$에서의 내차수의 합 또는 간접 사용 간선의 개수가 증가하므로 프로시저의 복잡도는 증가하게 된다.

$$SCM(P')\geq SCM(P)$$

**AM**$_2$: P의 한 위치로부터 다른 위치로 직접 또는 간

접 사용 간선을 이동시키다. 결과로서 P에서의 내차수의 합 또는 간접 사용 간선의 개수는 변하지 않으므로 프로시저의 복잡도는 변하지 않는다.

$$SCM(\boldsymbol{P}') = SCM(\boldsymbol{P})$$

**AM$_3$**: P내의 임의의 위치에 직접사용 또는 간접 사용 간선을 추가한다. 결과로서 P에서의 임의의 노드의 내차수가 증가하므로 프로시저의 복잡도는 증가하게 된다.

$$SCM(\boldsymbol{P}') > SCM(\boldsymbol{P})$$

따라서, P에서의 노드와 간선의 추가가 프로시저의 복잡도를 증가시킬 수 있다는 것에 동의하면, SCM을 순서척도로 사용할 수 있다. 즉, SCM에 의해 부과되는 순서가 원소 뷰포인트에 관한 사용자의 직관에 일치하는 정도로 SCM은 순서척도 상에 있다.

### 4.2 비율척도(ratio scale)

측정값에 대한 곱과 나누기를 수행하기 위해서는 척도가 비율척도이어야 한다[6]. 척도가 비율척도 상에 있다는 것을 보여주는 하나의 방법은 순서척도 평가에서 사용된 관계형 시스템에 합성 연산자 "$\circ$"를 추가하는 것이다.

복잡도 뷰포트에 "$\circ$"의 추가는 슬라이스 기반 정보흐름 그래프에 의해 표현되는 프로시저 $\boldsymbol{P}$에 대한 확장된 관계형 시스템 ($\boldsymbol{P}$, $\bullet\geq$, $\circ$)을 생성한다. 확장 관계형 시스템이 다음의 공리를 만족하면 확장구조(extensive structure)이다[5]. 4.1절에서 $\boldsymbol{P}$에 노드와 간선을 추가하는 것이 $\boldsymbol{P}$의 복잡도를 증가시킨다는 것을 보았다. 따라서 확장 관계형 시스템에서 관계 $\bullet\geq$ 대신에 SCM에 의한 실수 값을 비교하기 위해서 $\geq$를 이용한다[6].

A1(weak order):

(SCM($\boldsymbol{P}$), $\geq$)

A2(weak associativity):

SCM($\boldsymbol{P}_1 \circ (\boldsymbol{P}_2 \circ \boldsymbol{P}_3)$) = SCM(($\boldsymbol{P}_1 \circ \boldsymbol{P}_2) \circ \boldsymbol{P}_3$)

A3(weak community):

SCM($\boldsymbol{P}_1 \circ \boldsymbol{P}_2$) = SCM($\boldsymbol{P}_2 \circ \boldsymbol{P}_1$)

A4(weak monotonicity):

SCM($\boldsymbol{P}_1$)$\geq$SCM($\boldsymbol{P}_2$)$\Rightarrow$SCM($\boldsymbol{P}_1 \circ \boldsymbol{P}_3$)$\geq$SCM($\boldsymbol{P}_2 \circ \boldsymbol{P}_3$)

A5(archimedian axiom):

SCM($\boldsymbol{P}_3$)>SCM($\boldsymbol{P}_4$)이면 SCM($\boldsymbol{P}_1 \circ n\boldsymbol{P}_3$)>SCM($\boldsymbol{P}_2 \circ n\boldsymbol{P}_4$)인 $n$이 존재한다.

위의 공리들은 척도의 특성들을 위해하기 위해서 중요하다[11]. 확장 구조 ($\boldsymbol{P}$, $\bullet\geq$, $\circ$)에서, 척도가 실수 값 함수 m일 때, 다음 공리를 만족하면 척도는 비율척도가 된다[5].

$$P_1 \bullet > P_2 \Leftrightarrow m(P_1) \geq m(P_2)$$

$$m(P_1 \oplus P_2) \Leftrightarrow m(P_1) + m(P_2)$$

첫 번째 공리는 $m$이 측정된 속성에 의해 부과된 프로시저의 직관적 순서와 일치할 것을 요구한다. 즉 m이 순서척도이다. 두 번째 공리는 m이 가법적(additive)임을 요구한다.

Zuse는 제어 흐름 그래프(CFG)에 근거한 척도들을 분석하기 위해서 순차적 결합 연산자인 BSEQ와 "if-then-else" 이항연산자인 BALT를 제안하였다. 임의의 그래프 $P_1$과 $P_2$가 주어질 때, BSEQ=$P_1 \circ P_2$는 $P_1$의 종단노드와 $P_2$의 시작 노드를 연결하며, BALT=$P_1 \bullet P_2$는 하나의 결정노드와 하나의 결합(join) 노드로 $P_1$과 $P_2$를 연결한다[5].

본 연구에서는 Zuse의 BSEQ와 BALT를 수정하여, SIFG로 표현되는 프로시저 $\boldsymbol{P}$=<N, E> 상에서 MBSEQ (modified BSEQ)와 MBALT(modified BALT)라 불리는 두 개의 합성 연산자를 정의한다.

**정의 3**: 두 프로시저 $\boldsymbol{P}_1$과 $\boldsymbol{P}_2$가 주어진다고 하자. MBSEQ=$\boldsymbol{P}_1 \circ \boldsymbol{P}_2$는 $\boldsymbol{P}_1$의 종단노드로부터 $\boldsymbol{P}_2$의 시작노드로의 직접 사용 간선을 추가한다.

**정의 4**: 두 프로시저 $\boldsymbol{P}_1$과 $\boldsymbol{P}_2$가 주어진다고 하자. MBALT=$\boldsymbol{P}_1 \bullet \boldsymbol{P}_2$는 하나의 결정노드로부터 $\boldsymbol{P}_1$와 $\boldsymbol{P}_2$의 시작노드들로 그리고 $\boldsymbol{P}_1$와 $\boldsymbol{P}_2$의 종단노드들에서 하나의 결합(join) 노드로의 직접 사용 간선들을 추가한다.

그림 7은 이항 연산 MBSEQ와 MBALT의 예를 보여준다.

SCM이 이항연산 MBSEQ와 MBALT에 대해서 공리 A1, A2, A3, A4, A5를 항상 만족시키므로 SCM은 이항연산 MBSEQ와 MBALT에 대해서 확장구조이다라고 말 할 수 있다.

이항연산 MBSEQ가 $\boldsymbol{P}_1 \circ \boldsymbol{P}_2$에서 임의의 노드의 내차수를 증가시키지 않으므로, SCM은 이항연산 MBSEQ에 대해서 가법성을 만족한다. 즉,

$$SCM(\boldsymbol{P}_1 \circ \boldsymbol{P}_2) = SCM(\boldsymbol{P}_1) + SCM(\boldsymbol{P}_2).$$

따라서, 사용자가 순서척도의 부분특성들과 이항 연산 MBSEQ에 동의하면 SCM은 비율척도로 사용될 수 있다.

그러나 SCM은 이항 연산 MBALT에 대해서 가법성을 만족시키지 않는다. 즉,

$$SCM(\boldsymbol{P}_1 \bullet \boldsymbol{P}_2) = SCM(\boldsymbol{P}_1) + SCM(\boldsymbol{P}_2) + 2.$$

따라서 SCM은 MBALT에서 비율척도로 사용될 수 없다. 그러나, 다음과 같이 정의되는 SCM2는 MBALT에서 가법적이다.

$$SCM2 = SCM + 2.$$

SCM2는 확장구조이고 가법성을 만족하므로, 비율척

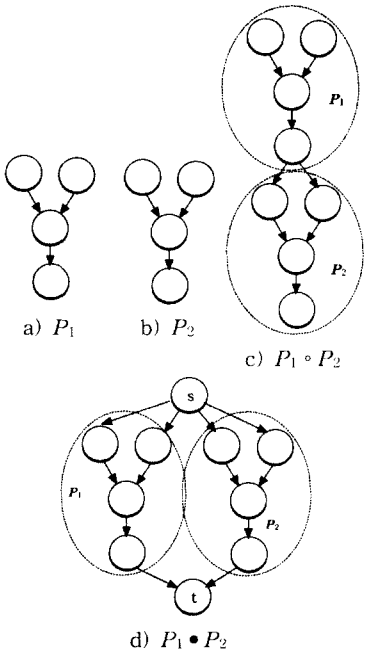


그림 7 이항 연산의 예

도로 사용될 수 있다.

SCM은 이항 연산 MBALT와 MBSEQ에서 Zuse가 요구하는 가법성을 항상 만족하지는 않는다. 그러나 Weyuker[12]의 9번째 공리(즉, $\mu(P)+\mu(Q)\leq\mu(P;Q)$)를 만족한다.

Weyuker의 9번째 공리는 비율척도에 대해서 의미가 있다. 즉, 슬라이스 복잡도 척도가 비율척도를 나타내면 이 공리는 의미를 갖는다[5]. 그러므로, 이진 연산 MBSEQ와 MBALT에 대한 Weyuker의 9번째 공리에 동의하고, 순서척도의 부분특성을 수용하면, 슬라이스 복잡도 척도를 이진 연산 MBSEQ와 MBALT에 대해 비율척도로 사용할 수 있다.

## 5. 실험 및 결과

프로그램의 복잡도를 측정하는 대표적인 척도들에는 Lines Of Code(LOC), Maccabe의 순환수(Cyclomatic number), Halstead의 소프트웨어 과학(Software Science) 등을 들 수 있다[5].

LOC는 일관성이 없다는 단점이 있지만 간단하여 사용자가 직관적으로 알 수 있다는 장점이 있다[13].

Halstead의 Length, Volume, Difficulty, Effort 등은 프로그램에 내재된 논리의 규모를 측정하는 척도들이다. 이 척도들은 적용과 검증 결과 정확성이 뛰어나고, 실험적으로도 매우 광범위하게 그 타당성을 인정받고 있으며, 프로그램을 작성하는데 요구되는 시간과 정신적인 노력을 예측하는데 관련이 있다[15]. IBM의 연구자들은 LOC(lines of code), Length, Volume들 사이의 강한 관련성을 발견했다[14].

McCabe가 제안한 순환수(cyclomatic number)는 프로그램의 논리적인 복잡도를 표현하는 제어 구조에 기반을 둔 대표적인 척도이다. 이 척도는 많은 유용성에도 불구하고 제어구조에만 의존하여 복잡도를 측정함으로써 순차적으로 구성된 프로그램들이 크기에 관계없이 동일한 측정값을 갖는다는 문제는 있지만, 경험적 연구를 통하여 그 효율성이 증명되었다[16].

따라서, 본 연구에서는 SCM과 기존 척도들과의 관련성을 통해서 SCM을 분석한다. 먼저 SCM과 기존 척도와의 관계를 통해서 SCM을 이용한 프로그램 분류기준을 찾고, SCM과 기존 척도들과의 상관관계의 분석을 통해 SCM의 특성을 보여준다.

본 연구에서는 C-언어 런타임 라이브러리 소스와 [6], [8], [9], [17], [18] 등에서 추출한 270개의 프로시저 집합을 대상으로 실험하였다.

표 1은 270개의 프로시저에 대한 기술통계량(descriptive statistics)을 보여준다.

표 1 기술통계량

| | Min | Max | Mean | Std. Deviation | Variance |
|---|---|---|---|---|---|
| LOC | 4.00 | 83.00 | 20.4630 | 13.8378 | 191.484 |
| CYC | 1.00 | 20.00 | 3.3481 | 2.3018 | 5.298 |
| VOL | 33.00 | 3380.97 | 386.407 | 411.5702 | 169390.012 |
| DIF | 2.81 | 123.18 | 20.6431 | 15.4483 | 238.649 |
| EFF | 133.19 | 214827.3 | 12242.8 | 22875.836 | 523303907.2 |
| SCM | 1.00 | 176.00 | 20.4148 | 23.1900 | 537.775 |

※CYC: CYClomatic number, VOL:VOLume,
DIF:DIFficulty EFF: EFFort

### 5.1 SCM을 이용한 프로그램 분류

산업 현장에서의 실험적인 연구들은 LOC는 30이하, CYC는 10이하, DIF는 10이하, EFF는 10,000이하가 적합하고, VOL은 평균적으로 1,000이하가 적당하다는 것을 보여주었다[14, 16].

따라서, 위의 기준을 근거로 하여, SCM과 강한 상관관계를 보여주는 LOC, VOL, DIF, EFF에 대한 중회귀분석을 수행하였다.

표 2 회귀모델의 요약

| R | $R^2$ | Adjusted $R^2$ | Std. Error of the Estimate | Durbin-Watson |
|---|---|---|---|---|
| 0.873 | 0.762 | 0.759 | 11.396 | 1.408 |

Predictors: (Constant), EFF, LOC, VOL, DIF
Dependent Variable: SCM

표 2는 중회귀분석의 결과로 발견된 회귀 모델의 요약을 보여준다. 표 2에서 결정계수($R^2$)가 0.758로 전체 분산 중에서 약 75%를 설명할 수 있다는 것을 보여주며, 자기상관(autocorrelation) 현상을 측정할 수 있는 Durbin-Watson 통계량이 1.408로 자기상관이 없다고 할 수 있다.

분산 분석의 결과 F=184.060>F(4, 149;0.01)=6.8081이고 F-확률은 0.000으로 유의수준 1%에서 매우 유의하다고 할 수 있으며, 잔차의 평균 제곱(residual mean square)이 7.7662이므로 관찰값들이 회귀방정식 주위에 대체적으로 가깝게 밀집되어 있다고 할 수 있다. 또한 오차가 모집단에서 정규분포를 따른다고 가정하여도 잔차의 분포가 근사적으로 정규분포를 따름을 알 수 있었고, 표준화된 잔차들이 정규분포 근처에 있으므로 예측 정도가 높다고 할 수 있으며, 모든 변수들의 공차한계(tolerance limit: TOL)가 0.1이상이고 분산 팽창 계수(variance inflation factor: VIF)가 10 이하이므로 실험 데이타의 다중공선성은 없다고 말할 수 있으며, 잔차들이 0을 중심으로 수평대(horizontal band)를 형성하고 있으므로 등분산성의 가정에 아무런 이상이 없었다[19].

따라서 위의 회귀분석의 결과는 유의하며, 회귀분석의 결과를 통해 SCM에 의해 프로시저들을 <표 3>의 분류기준과 같이 분류할 수 있다.

표 3 SCM에 의한 프로시저의 복잡도 분류

| SCM | 설 명 |
|---|---|
| low complexity (SCM<= 45) | 구조가 단순하고 규모가 작은 프로그램 |
| high complexity (45>SCM) | 매우 구조가 필요 이상으로 복잡하거나 규모가 큰 프로그램 |

5.1.1 SCM의 구분력

F가 소프트웨어의 규모와 구조의 복잡도를 나타내는 척도인 SCM이라 하고, 기존 척도들의 집합을 M={LOC, Cyclomatic Number, Volume, Difficulty, Effort}이라 할 때, 척도 M의 분류 기준 값(Mc)들이 다음 식과 같이 벡터 F의 원소들을 구분할 수 있으면, F의 분류기준 값은 적합하다고 할 수 있다.

$$\begin{pmatrix} M_i > M_c \Leftrightarrow F_i > F_c & \text{and} \\ M_i < M_c \Leftrightarrow F_i < F_c & \end{pmatrix}$$

적절한 통계적 방법인 카이스퀘어($\chi^2$) 검정을 이용하여 위의 식이 성립하는 정도를 보여줄 수 있다[20]. 즉

$\chi^2 > \beta_\alpha$ , $\beta_\alpha$: $\alpha$에 의해 명시된 임계치

270개의 프로시저들에 대한 SCM 측정값들에 대해서 각 측정 인자들에 대한 분할표(contingency table)을 구성하고, $\chi^2$ 검정값을 산출하였다.

표 4 기존척도들의 chi-square

| 척 도 | $\chi^2$ |
|---|---|
| Lines of code | 76.781 |
| Cyclomatic number | 13.500 |
| Volume | 2061.102 |
| Difficulty | 6.463 |
| Effort | 72.013 |

표 4에서 보여주듯이, 대부분의 기존 척도들에 대한 $\chi^2$ 의 값은 95% 수준에서의 임계치($\beta_{0.05}$= 3.841) 매우 높게 나타났고, 샘플에서 유의수준은 매우 낮았다.

따라서, 표 3의 SCM에 의한 소프트웨어 분류 기준은 구분력(discriminative power)에 있어서 유효하다고 할 수 있고, 프로그램을 분류하기 위해 사용할 수 있다.

5.1.2 SCM의 일치성

SCM을 산업현장에서 사용할 수 있으려면, SCM 측정값이 기존의 검증된 메트릭과 측정의 일치성(consistency)을 보여야 한다.

F가 소프트웨어의 규모와 구조의 복잡도를 나타내는 척도인 SCM이라 하고, 기존 척도들의 집합을 M={LOC, Cyclomatic Number, Volume, Difficulty, Effort}이라 할 때, 척도 F가 일치성(consistency)를 갖기 위해서는 집합 M과 F의 순위 상관관계 계수(rank correlation coefficient) r이 명시된 임계치(specified threshold)을 초과해야 한다[20]. 즉,

$r > \beta_c$, $\beta_c$: $c$에 의해 명시된 임계치

위의 식은 M의 간접적인 척도로서 F를 사용할 수 있을 만큼 M의 순위(rank)와 F의 순위가 충분히 일치하는 가를 보여준다. 즉,

$$\text{순위}[M_1 < M_2 \ldots < M_n] \Leftrightarrow$$
$$\text{순위}[F_1 < F_2 \ldots < F_n]$$

순위 관계가 성립하는가를 검증하기 위해서 270개의 프로시저들에 대한 순위 상관계수(즉, Spearman의 $\rho$ 계수)를 구했다.

표 5 순위상관관계 계수

| | LOC | CYC | VOL | DIF | EFF |
|---|---|---|---|---|---|
| SCM | .728 | .589 | .787 | .773 | .822 |

<표 5>에서, SCM과 기존척도들과의 유의수준 0.01에서 유의하다. 즉, 척도 SCM는 기존의 척도들과 일치된 순위 관계를 보여준다. 따라서 우리는 SCM을 프로시저들이 상대적 복잡도에 있어서 얼마나 다른가를 평가하기 위해서 기존의 척도들을 대신하여 사용할 수 있다.

### 5.2 SCM과 기존 척도와의 비교 평가

표 6은 기존 척도들(즉, LOC, cyclomatic number, software science)과 SCM 간의 상관관계를 보여준다.

표 6 상관관계 계수

| | LOC | CYC | VOL | DIF | EFF | SCM |
|---|---|---|---|---|---|---|
| LOC | 1.000 | .689 | .827 | .794 | .793 | .707 |
| CYC | .689 | 1.000 | .427 | .586 | .441 | .439 |
| VOL | .827 | .427 | 1.000 | .673 | .825 | .860 |
| DIF | .794 | .586 | .673 | 1.000 | .879 | .651 |
| EFF | .793 | .441 | .825 | .879 | 1.000 | .727 |
| SCM | .707 | .439 | .860 | .651 | .727 | 1.000 |

표 2에서 LOC와 software science간의 상관 관계는 높게 나타났고, LOC와 CYC, CYC와 software science 간에는 상관관계가 낮게 나타났다. 이것은 IBM 연구자들의 실험[14]과 일치하는 결과임을 보여준다.

표 2에서 CYC와 SCM간에는 상관 관계가 낮게 나타났고, SCM과 software science간에는 강한 상관관계를 보여주었다. SCM이 software science와 LOC와 강한 상관관계를 보여준다는 것은 SCM이 프로그램의 물리적 크기를 반영한 측정을 한다는 것을 의미한다. 따라서, SCM이 변수들간의 정보의 흐름에 근거하여 흐름 복잡도를 측정하지만 프로그램의 흐름관계뿐만 아니라 물리적 크기를 반영하여 프로그램의 복잡도를 측정한다는 것을 의미한다.

또한, SCM은 프로그램에 나타나는 연산자와 피연산자의 개수보다는 연산자와 피연산자의 총 발생 빈도수와 더 높은 상관 관계를 보여주었으며, 제어흐름그래프에서 간선과 정점의 개수와도 높은 상관관계를 보여주었다. 따라서 이들의 값이 상대적으로 높은 프로시저들은 다른 프로시저에 비해 높은 SCM 측정값을 갖게된다.

CYC는 프로그램에서 제어 정보의 흐름만을 측정하므로 순차적인 프로그램들이 크기에 관계없이 측정값이 동일하고, software science와 LOC는 프로그램의 물리적 크기만을 반영하는 측정이 이루어지지만, SCM은 프로그램 내에서의 제어와 데이타 흐름뿐만 아니라 프로그램의 물리적 크기를 동시에 반영하는 측정이 이루어짐을 알 수 있다. 따라서 순차적인 프로그램들도 크기에 따라서 상이한 SCM 측정값을 갖게되고, 프로그램의 구분력에 있어서 기존의 척도들보다 향상된 측정을 제시할 수 있다.

실험 결과, 프로그램 상에서의 변수들의 미세 변화와 정보의 흐름을 분명하게 표현해주는 SIFG에 근거한 SCM은 프로그램 내에서의 정보흐름의 복잡도를 측정할 수 있엇다. 또한, 기존의 척도들이 소프트웨어의 구조와 규모의 속성 중 하나의 속성만을 측정하는 데 비하여, SCM은 규모와 구조를 하나의 척도로 측정할 수 있다.

규모와 구조를 동시에 측정하는 대부분의 혼합(hybrid) 척도들은 기존 척도들의 결합을 통하여 측정하지만 SCM은 단일 척도로 이 두 가지의 특성을 측정할 수 있다. 따라서 기존의 혼합 척도들보다는 훨씬 단순한 측정 방법으로 소프트웨어의 규모와 구조를 동시에 측정할 수 있다.

기존 척도들과의 관계에서 발견한 SCM을 이용한 소프트웨어 분류 기준은 소프트웨어 구분력에 있어서 유효하였고, 또한 기존의 척도들과 강한 순위 상관 관계를 보여주었다.

## 6. 결 론

본 연구에서는 측정 이론으로부터의 원칙들을 이용하여 슬라이스에서의 정보흐름에 기초하여 프로그램의 복잡도를 측정하는 방법을 제안하였다. 먼저, 슬라이스에서의 정보 흐름을 모델링하기 위해 슬라이스 기반 정보 흐름 그래프(*SIFG: Slice-based Information Flow Graph*)를 개발하였다. 다음으로, SIFG에서의 정보 흐름의 복잡도를 측정하기 위해 슬라이스 기반 복잡도 척도(*SCM: Slice-based Complexity Measure*)를 정의하였다.

Zuse가 제시한 방법에 따라서, 본 연구에서는 간단한 극소 수정들의 집합에 의해 부과되는 순서가 복잡도에 관한 우리의 직관과 일치하는 정도로 SCM이 순서척도의 요구조건을 만족함을 보여주었다. MBSEQ와 MBALT

의 연산 중에서 MBSEQ에 대해서는 SCM이 Zuse가 요구하는 가법성(additive)을 만족하였다. MBALT에 대해서는 가법성을 만족하지 않지만 Weyuker의 9번째 공리를 만족하였다.

따라서 순위척도의 부분특성들에 동의하고, SCM에 대한 이항연산 MBSEQ에 동의하면 SCM을 비율척도로 사용할 수 있다. 또한 이항연산 MBSEQ와 MBALT에 대한 Weyuker의 9번째 공리에 동의하면 SCM 측정값을 분석할 때 순서척도뿐만 아니라 비율척도 연산도 사용할 수 있다.

또한 기존 척도들과의 비교를 통해서, SCM은 프로그램 내에서의 제어와 데이타 흐름뿐만 아니라 프로그램의 물리적 크기를 반영하는 측정이 이루어진다는 것을 확인할 수 있었다.

## 참 고 문 헌


[1] Karl J. Ottensteion, Linda M. Ottensteion, "The program dependence graph in a software development environment," *Proceeding of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environment, ACM SIGPLAN Notices*, vol.19, no.5, pp.177-184, 1984.

[2] M. Weiser, "Programmers use slices when debuggung," *Communication of the ACM*, vol.25, no.7, pp.446-452, 1982.

[3] M. Weiser, "Program slicing," *In Prodeedings of the 5th International conference on Software Engineering*, pp.439-449, 1981.

[4] B.A. Kitchenham, N. Fenton, S. Lawrence, "Towards a framework for software measurement validation," *IEEE Transaction Software Engineering*, vol.21, no.12, pp.929-944, 1995.

[5] Horst Zuse, *Software Complexity-Measures and Methods*, pp.25-37, Walter de Gruyter, New York, 1991.

[6] J.M.Bieman, L.M.Otto, "Measuring functional cohesion," *IEEE Transaction Software Engineering*, vol.20, no.2, pp.111-124, 1994.

[7] Lnda M. Otto, "Using Slice Profiles and Metrics during Software Maintenance," Proc. 10th Annual Software Reliability Symposium, 1992.

[8] Linda M. Ott, Jeffrey J. Thuss, "Slice based metrics for estimating cohesion," *Proc. IEEE-CS International Software Metrics Symposium*, Baltimore, pp.71-81, 1993.

[9] J.M.Bieman, B.K.Kang, "Measuring design-level cohesion," *IEEE Transaction Software Engineering*, vol.24, no.2, pp.111-124, 1998.

[10] G. Poels, B. Dedene, "Comments on Property-Based Software Engineering Measurement," *IEEE Transaction Software Engineering*, vol.23, no.3, pp.190-195, 1997.

[11] Horst Zuse, "Reply to: Property-Based Software Engineering Measurement," *IEEE Transaction Software Engineering*, vol.23, no.8, pp.533, 1997.

[12] E.Weyuker, "Evaluating software complexity measure," *IEEE Transaction Software Engineering*, vol.14, no.9, pp.1357-1356, 1988.

[13] 이철희 등. *소프트웨어 공학*, 서울:홍릉과학 출판사, 1989.

[14] Arthur, L. J.. *Measuring Programmer Productivity and Software Quality*, pp.138~142, New York:John Wiley & Sons, 1985.

[15] Shen,V. Y. Conte, S. D. Dunsmore, H. E.. "Software Science Revisited : A Critical Analysis of the Theory and its Empirical Support.", *IEEE Transactions On Software Engineering* vol.9, pp.308-320, 1976.

[16] 최완규, "러프논리와 퍼지척도에 기반한 재사용 결정지원 모델", *박사학위 논문, 조선대학교*, 2000.

[17] Linda M. Otto, Jeffrey J. Thuss. "The Relationship between Slices and Module Cohesion," *Proc. 11th ICSE*, 1989.

[18] Kyle Loudon, *Algorithms with C*, O'REILLY, 1999

[19] 김은정, 박양규, *SPSS 통계분석*, 서울:21세기사, 2000.

[20] Schneidewind, N.F., "Methodology For Validating Software Metrics," *IEEE Transactions On Software Engineering*, vol.18, pp.410-422, 1992.



최 완 규

1988년 서울대학교 종교학과(학사). 1992년 ~ 1993년 (주)공성통신 전산실. 1993년 ~ 1995년 한양시스템 전산실. 1997년 조선대학교 전자계산학과(이학석사). 2000년 조선대학교 전자계산학과(이학박사). 2000년 ~ 현재 광주대학교 컴퓨터전자통신공학부 전임강사. 관심분야는 소프트웨어 공학, 프로그래밍 언어, 객체지향 시스템, 퍼지 및 러프집합

정 일 용

1983년 한양대학교 공과대학 졸업(학사). 1987년 미국 City University of New York 전산학과(석사). 1991년 미국 City University of New York 전산학과(박사). 1991년 ~ 1994년 한국전자통신연구소 선임연구원. 1994년 ~ 현재 조선대학교 컴퓨터공학부 부교수. 관심분야는 컴퓨터 네트워크, 분산 및병렬 시스템, 망 보안시스템

이 성 주

1992년 광운대학교 전자계산학과(이학석사). 1998년 대구 카톨릭대학교(이학박사). 1988년 ~ 1990년 조선대학교 전자계산소 부소장. 1995년 ~ 1997년 조선대학교 정보과학대학장. 1981년 ~ 현재 조선대학교 컴퓨터공학부 교수. 관심분야는 소프트웨어 공학, 프로그래밍 언어, 객체지향 시스템, 러프집합