2000년 한국정보처리학회 추계 학술발표논문집 제 7 권 제 2 호

UML 기반의 수정 영향 분석을 위한 제어 의존성 관계 그래프와 알고리즘

최미숙*, 김경희**, 박재년*, 윤용익*
*숙명여자대학교 컴퓨터과학과
**천안외국어대학 산업전산과
e-mail : cms@cs.sookmyung.ac.kr

The Algorithm and Control Dependency Graph for Change Impact Analysis based on UML

Mi-Sook Choi*, Kyung-Hee Kim**, Jae-Nyun Park*,
*Dept. of Computer Science, Sook-Myung Women's University
**Dept. of Industry & Computer Science, Chonan Foreign Studies College

요 약

지금까지의 수정영향 분석에 대한 연구는 주로 원시코드 기반으로 진행되어 왔다. 그러나 원시코드를 기반으로 한 소프트웨어의 수정영향 분석은 세부적이고 복잡하여, 소프트웨어의 수정영향 범위를 효과적으로 분석하고 클래스간의 제어의존성을 파악하기에 부적합하다. 따라서, 본 논문에서는 UML 의 설계단계 산출물인 순차도(sequence diagram)를 기반으로 하여, 클래스간의 수정영향범위 분석을 위한 제어의존성 관계그래프와 알고리즘을 제안한다. 본 논문에서 제안한 제어의존성 관계그래프와 알고리즘은, 소프트웨어에 수정이 일어났을 경우에 수정의 영향범위를 결정하고 제어구조에 따라서 테스트의 순서를 명확히 정의함으로 유지보수 단계에서 시스템을 효율적으로 관리하고 유지하는데 드는 비용과 노력을 절약한다.

1. 서론

소프트웨어의 생명주기(Software Life Cycle)는 일반적으로 다섯 단계를 거치고, 그것은 요구분석 단계, 설계 단계, 구현 단계, 테스트 단계, 유지보수 단계이다[1][2]. 근래에는 컴퓨터 시스템이 대형화되고 업무가 복잡해짐에 따라 소프트웨어 시스템을 유지보수(maintenance)하는 비용이 점차적으로 증가하고 있다. 한 조사에 따르면, 소프트웨어를 개발하는 모든 비용의 70%이상이 유지 보수하는데 소요된다고 한다[2].

소프트웨어의 유지 보수 단계에서는 소프트웨어 변경사항으로 인한 수정(Change)이 빈번하게 발생한다. 이 때 발생한 수정이 프로그램에서 올바른 영향을 미치는가 하는 것과 그 프로그램의 수정으로 인해 발생되는 문제가 다른 부분들에게 미치는 영향을 찾아야 한다[3].

소프트웨어의 유지 보수 단계에서 중요한 역할을 감당하는 회귀 테스팅(regression testing)은, 시스템에서 수정이 발생한 이후에 소프트웨어를 재 테스팅(retesting)하는 것을 포함한다. 수정은 명세에서 발생하기도 하고 원시 코드에서 발생하기도 한다[4][5].

지금까지의 수정영향 분석에 대한 연구는 주로 원시코드 기반으로 진행되어 왔다. 그러나 원시코드를 기반으로 한 소프트웨어의 수정영향 분석은 세부적이고 복잡하여, 소프트웨어의 수정영향 범위를 효과적으로 분석하기에 부적합하다.

따라서, 본 논문에서는 UML 의 설계단계 산출물인 순차도(sequence diagram)를 기반으로 하여, 클래스간의 수정영향범위 분석을 위한 제어의존성 관계그래프와 알고리즘을 제안한다. 본 논문에서 제안한 제어의존성 관계그래프와 알고리즘은, 유지보수 단계에서 시스템을 효율적으로 관리하고 유지하는데 드는 비용과 노력을 절약한다.

2. 관련연구

소프트웨어는 모든 소프트웨어 개발 주기를 통하여 수정된다. 이러한 수정들은 크거나 작고, 단순하거나 복잡하고, 중요하거나 중요하지 않은 것들이다.

소프트웨어의 유지보수 단계에서 효율적으로 작업을 수행하기 위해서 수정 범위를 정확히 분석하고, 이러한 수정이 다른 소프트웨어에 올바른 영향을 미치는지를 검증을 할 필요가 있다.

수정영향 분석이란 소프트웨어의 수정을 올바로 수행하기 위해서, 수정의 발생과 필요성을 분석하거나 수정의 잠재적인 요소를 식별하는 것이다[6].

수정영향 분석은 정적 수정영향 분석(static impact analysis)과 동적 수정영향 분석(dynamic impact analysis)으로 구성되며, 정적 수정 영향 분석은 프로그램(code) 수행 없이 정적 코드 구조(static code structure)를 분석하는 것이고, 동적 수정 영향 분석은 수정의 영향을 결정하기 위하여 소프트웨어를 수행(executes)하는 방법이다.

수정 영향의 범위를 확인하기 위해서, 시험자는 소프트웨어 요소들 사이의 관계의존(dependency relationship)과 추적성(traceability)을 이용하여 수정영향의 범위를 분석하고 테스트 순서를 결정한다.

관계의존은 자료 의존(data dependency)과 제어의존(control dependency) 으로 구분된다. 제어 의존은 프로그램이 실행될 때, 소프트웨어 요소들 간의 의존관계를 뜻하고, 자료 의존은 자료를 사용하거나 정의하는 소프트웨어 요소들 사이의 의존을 뜻한다. 관계의존은 프로그램 분할(program slicing), 추론(inferencing)에 의하여 이루어지고 있고, 개체(entity)들 사이에서 코드의 의존성을 상세하게 분석하는 것을 포함한다.

추적성(traceability) 분석은 요구(requirement), 설계(design), 구현(code), 테스트(test) 단계 산출물인 개체들 사이의 관계를 추적하고 정의하는 것을 포함한다.

본 논문에서는 정적 수정 영향 분석을 이용해서 수정의 영향 범위를 분석한다. 또한, 수행되어진 수정영향 분석을 평가하기 위하여, 추적성 분석을 통하여 관계의존 분석을 수행하고, 관계의존 중에서 제어흐름에 기준을 두고 수정의 범위를 분석하고 테스트의 순서를 결정한다.

3. UML

UML 은 Booch, Rumbaugh, and Jacobson 의 3 인이 연합하여 개발한 것으로, 객체지향 개발 방법론을 위한 산업 표준(industry standard)으로 OMG 에 의해서 받아들여 졌다[7][8].

본 논문에서는 산업의 표준으로 자리잡은 UML 의 각 단계 산출물중 설계단계의 순차도(sequence diagram)를 이용하여 클래스간의 제어흐름을 분석하고 평가한다. 즉, 부여된 순차도로부터 수정의 영향 범위를 결정하고 결정된 범위 내에서 테스팅의 순서를 결정한다[9].

순차도는 UML 의 요구 분석 단계의 산출물인 사용사례(UseCase)와 사용사례(Usecase)의 시나리오에 의하여 클래스와 클래스간의 메시지를 순차적으로 시간의 흐름에 따라서 기술한다. 각 순차도는 하나의 시나리와 연관된다.

순차도는 하나의 시나리오에 참여하는 클래스들과, 순서에 따라서 일어나는 클래스들간의 메시지, 각 메시지를 따라서 전달 되어지는 데이터와 리턴값을 포함한다. 순차도에서 추출된 클래스들간의 메시지는 클래스의 공용 메소드(public method)를 정의하는데 사용되어 진다.

4. UML 기반의 클래스간의 제어의존 관계그래프

객체지향 소프트웨어(object oriented software)의 기본적 구성 블록(basic building block)인 클래스는 데이터(data, attributes)와 프로시져(procedure, member function)를 은닉(encapsulation)하고 객체지향 테스트에서 테스팅의 기본 단위가 된다.

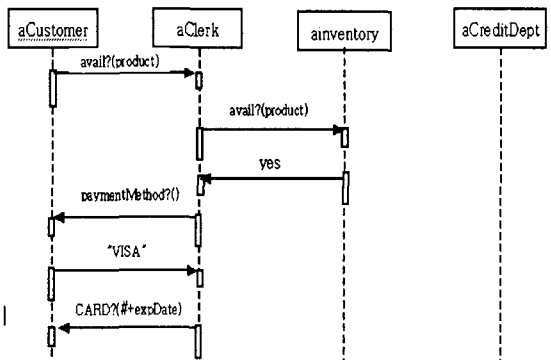
본 논문에서는 테스팅의 기본 단위인 클래스를 단위로 하고 클래스간의 제어 의존 분석을 위하여 UML의 산출물인 순차도를 사용한다.

다음은 신용카드를 사용해서 상품을 주문하기 위한 시나리오의 예이다. 이시나리오에 포함되는 액터와 클래스는 aCustomer, aClerk, ainvetory, aCreditDept 이다.

시나리오는 다음과 같다.

1) Customer asks for product
2) Clerk checks Inventory
3) Product is available
4) Clerk asks Customer for method of payment
5) Customer answers "credit card"
6) Clerk asks for card number and expiration data
7) Customer provides card number and expiration date
8) Clerk checks the Customer's credit through the Credit Department
9) Clerk Department authorizes credit card purchase
10) Clerk confirms the order with Customer and reduces Inventory.

다음의(그림 1)은 앞의 시나리오를 순차도로 보인 것이다.



(그림 1) ATM 순차도의 예

(그림 1)에서 클래스 aCustomer 는 avail?(product)메시

지를 클래스 aClerk 에게 보낸다. 이는 클래스 aClerk 에 정의된 메소드인 avail?(product)를 클래스 aCustomer 에서 호출(call)하는 형태이다. 따라서, 순차도는 클래스들 사이에서 메소드의 호출여부 즉, 제어의 흐름을 확인하고, 클래스들 간의 관계성을 분석하도록 돕는다.

본 논문에서는 순차도를 트리 구조로 변환하여 효율적으로 수정 영향 분석을 수행한다. 다음의 <알고리즘 1>은 순차도를 트리(tree)구조로 변경하기 위한 것이다.

<알고리즘 1>

(1) 순차도를 ASF(Automic System Function)[8] 단위로 나눈다.

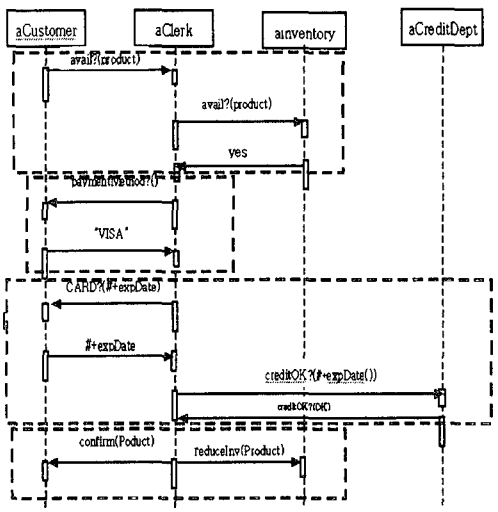
(2) ASF 단위로 순차구조, 조건구조, 병행구조를 포함한 서브 트리를 생성한다.

(3) (그림 3)에서 보인 바와 같이, ASF 단위로 구분하여 클래스와 메소드에 순차적인 번호를 부여한다.

(4) 한 순차도에서 생성된 각 서브 트리를 모아 하나의 완전한 트리로 구성한다.

일반적으로, ASF 는 입력에서 시작하여 출력까지의 단위를 뜻한다. 본 논문에서는 ASF 를 연속적으로 수행되는 메시지들의 최대의 집합으로 정의한다.

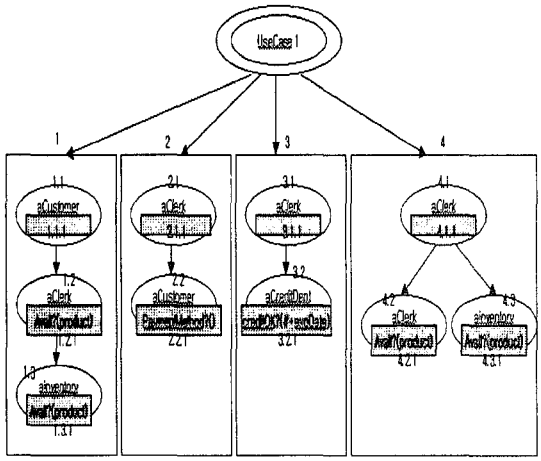
다음 (그림 2)는 <알고리즘 1>에서 정의한 바대로 (그림 1)의 순차도를 ASF 단위로 변화한 것을 보인다.



(그림 2) ASF 단위로 분리한 순차도의 예

(그림 2)는 4 개의 ASF 단위로 나누어지고 각각을 ASF1, ASF2, ASF3, ASF4 라고 정의한다.

(그림 2)는 알고리즘 1 에서 제시한 제어구조 중에 순차구조와 병행 구조만을 포함하고 있으며 ASF1, ASF2, ASF3 는 제시한 제어구조 중에 순차구조를 나타내고 있으며, ASF4 는 병행구조를 나타내고 있다.



(그림 3) 수정영향분석을 위한 트리구조

(그림 3)은 (그림 2)에서 발견된 ASF 를 트리구조로 변환한 것이다. (그림 3)의 트리의 노드는 클래스와 클래스내의 메소드(method)를 의미하고, edge 는 클래스들간의 메소드의 호출 관계를 표현한다.

다음의 <알고리즘 2>는 ASF 단위로 분리한 트리를 이용하여 수정영향의 범위와 테스트의 순서를 결정하는 알고리즘이다.

<알고리즘 2>

(1) 수정이 일어난 노드를 시작점으로 해서 단말노드까지, 노드들의 순번과 제어구조의 형태에 따라 순차적으로 검색한다.

(2) 검색된 결과에 의하여 테스트 할 노드의 순서가 순차번호로 출력된다.

본 논문에서는 제어 의존의 경우를 순차구조, 조건구조, 병행구조로 분리하고 제어 구조의 유형에 따라 수정 영향 범위를 결정 한다.

본 논문의 예에서와 같이 순차구조는 트리의 순서대로 테스트 순서를 정의하고, 병행구조는 트리의 부모 노드에 연결 되어 있는 자식 노드가 왼쪽 노드와 오른쪽 노드가 다 있을 경우이고 테스트의 순서는 동시에 2 가지의 테스트의 순서를 정의할 수 있다. 또한 조건구조는 참(true)일 경우와 거짓(false)일 경우로 나누어 테스트의 순서를 정의할 수 있다.

또한 본 논문에서는 수정의 발생을 세가지로 분류한다. 그것은 ASF 단위로 수정이 발생할 경우와 클래스에서 수정이 발생할 경우, 그리고 메소드에서 수정이 발생할 경우이다.

ASF 단위로 수정이 발생했을 경우의 예이다. (그림 3)의 2 번 ASF 에서 수정이 발생하였을 경우, 테스트할 노드의 순서는 각 노드의 순서 번호와 관련되어, 2.1 클래스의 2.1.1 메소드를 테스트하고, 다음으로 2.2 클래스의 2.2.1 메소드를 테스트한다. 또한, 4 번 ASF 에서 수정이 발생하였을 경우, 4.1 클래스의 4.1.1 메소

드를 테스트하고, 다음으로 4.2 클래스의 4.2.1 메소드와 4.3 클래스의 4.3.1 메소드를 동시에 테스트한다.

클래스에서 수정이 발생하였을 경우의 예이다. 클래스 3.1 에서 수정이 발생하였을 경우, 클래스 3.1 의 3.1.1 메소드를 테스트하고, 3.2 클래스의 3.2.1 메소드를 테스트한다. 또한 1.2 클래스에서 수정이 발생하였을 경우, 1.2 클래스의 1.2.1 메소드를 테스트하고, 1.3 클래스의 1.3.1 메소드를 테스트한다.

메소드에서 수정이 발생하였을 경우는 클래스에서 수정이 발생하였을 경우와 같다. 따라서, 본 논문에서 제시한 <알고리즘 1>에 의해 작성된 (그림 3)의 트리 구조는 <알고리즘 2>를 통하여 수정영향 범위와 테스트의 순서가 정의된다.

4. 결론

본 논문에서는 하나의 사용사례(UseCase)에 의한 순차도를 ASF 단위로 나누어 서브트리로 구성하고, 이를 통합하여 완전한 트리를 생성한다. 이 트리는 수정 영향의 범위를 찾도록 돕고, 제어 의존의 경우에 따라서 테스트의 순서를 보인다. 본 논문에서 제시한 <알고리즘 1>은 UML 의 순차도를 트리구조로 변환하기 위한 알고리즘이고, <알고리즘 2>는 변환된 트리구조에서 수정영향을 분석하고 테스트의 순서를 결정하는 알고리즘이다. 본 논문에서 제시한 두개의 알고리즘을 통하여 UML 기반의 산출물의 특성을 고려하여 효과적인 수정영향 분석 방법을 제시하였고, 수정을 ASF 단위와 클래스 단위 그리고 메소드 단위로 규정하고, 제어의존의 경우 순차구조, 병렬구조, 조건구조에 따라서 테스트의 순서를 정의하여 수정의 발생과 테스트의 순서를 명료화 하였으며 수정 영향을 범위를 분리하고 차단하는 효과가 발생함에 따라서 유지보수 단계에서 시스템을 효율적으로 관리하고 유지하는데 드는 비용과 노력을 절약한다.

5. 향후 연구 과제

본 논문은 클래스들 사이의 제어 의존성 분석에 기반하여 수정 영향을 분석 하였다.

향후 연구과제로는 수정 영향의 종류를 세분화하고, UML 의 여러 산출물의 특성을 고려한 수정 영향 분석을 하는 연구가 요구된다. 또한, 수정 영향의 종류에 근거한 수정 영향 분석을 자동화하는 자동화 도구에 관한 연구가 요구된다.

참고문헌

[1] 이주헌, "소프트웨어 공학론", 법영사, 1993.
[2] 김에녹, "소프트웨어공학", 이한출판사, 2000.
[3] 김경희, 윤용익, 박재년, "분산 객체지향 소프트웨어를 위한 수정 영향 분석", 한국정보처리학회 논문지, 제 6 권 제 5 호, 1999.
[4] 김경희, 윤용익, 박재년, "분산 객체지향 소프트웨어의 회귀 테스팅 도구 설계", 한국 정보처리학회 춘계 학술 발표 논문집, 제 6 권 제 1 호, 1999.
[5] K.H. Kim, M.S. Choi, Y.I. Yoon, J.N. Park, "Change Impact Analysis for Distributed Object-Oriented Software Using DPDG", Proceedings of the Fifth JCIS, Vol 1. pp. 299-302, 2000.
[6] Shawn A. Bohner and Robert S. Arnold, "An Introduction to Software Change Impact Analysis", IEEE Computer Society Press, pp. 1-26, 1996.
[7] Y.G. Kim, H.S. Hong, S.M. Cho, D.H. Bae, and S.D. Cha, "Test Cases Generation from UML State Diagrams", IEE Procedings – Software, Vol. 146, No. 4, pp. 187-192, Aug. 1999.
[8] 윤회진, 서주영, 최정은, 최병주, "UML 기반 콤포넌트 통합 테스팅", 정보과학회논문지 제 26 권 제 9 호, 1999.
[9] Hans-Erik Eriksson and Magnus Penker, "UML Toolkit", Wiley, 1998.
[10] 김인규, 최은만, "소프트웨어 유지 보수와 수정후의 검증에 관한 연구", 한국정보처리학회 춘계 학술발표논문집 제 6 권 제 1 호, pp. 468-471, 1999.