객체지향 프로그램의 클래스 상속 깊이에 관한 연구

문양선
서해대학 사무자동화과

Study on Class Inheritance Depth of Object-Oriented Program

Yangsun Moon
Office Automation, Sohae Junior College

요 약

객체 지향 패러다임의 상속성은 코드 재사용 및 확장을 용이하게 하여 소프트웨어 생산성을 높여준다. 그러나 상속을 남용하여 상속 계층구조(class inheritance hierarchy)의 상속 깊이가 깊게 되면 프로그램을 이해하는데 어려움을 준다. 이러한 관점에서 볼 때 상속성의 이용에 제한을 둘 필요가 있다. 본 논문은 상속 계층구조의 상속 깊이에 대한 기준을 세우기 위하여 인지실험을 행하고 그 결과를 보고한다. 본 논문의 목적은 객체지향의 특성 중 상속성의 장점을 잘 이용하면서 이해하기 쉬운 객체지향 소프트웨어 개발을 돕는 것이다.

1. 서론

소프트웨어가 점점 복잡해지고 대형화됨에 따라 소프트웨어 위기를 맞게 되었다. 객체지향 패러다임의 등장은 상속성, 캡슐화, 다형성이라는 특성을 제공하여 위기를 극복하는데 많은 도움을 주고 있다. 특히 상속성(inheritance)은 코드(code) 재사용 및 확장을 용이하게 하여 소프트웨어 생산성을 높여주기 때문에 많이 이용되고 있는 특성이다. 그러나 여러 연구에서 상속 계층구조의 상속 깊이가 깊은 프로그램은 이해하는데 많은 노력이 따른 다는 것을 지적하고 있다[1, 2, 3]. 유지보수 단계는 소프트웨어 개발의 전체 비용 중에서 60~80%를 소요하며, 유지보수 활동 중 소프트웨어 이해 작업은 유지보수의 전체 비용 중 50%이상 차지한다[4]는 것을 볼 때 상속성을 이용하는데 있어서 기준이 필요하다. 이러한 필요에 따라 본 논문에서는 인간의 인지 능력에 관한 실험을 통하여 어느 정도 깊이까지 상속성을 이용하는 것이 바람직 한 것인가를 밝힌다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 살펴보고, 3장에서는 인지심리학에서 정의된 청크 개념으로부터 객체지향 프로그램의 평면 청크와 내포 청크들을 정의한다. 4장에서는 상속 관계를 갖는 클래스들(내포 청크)의 상속 깊이에 기준을 세우기 위해 인지 실험을 하여 그 결과를 논한다. 5장에서는 결론 및 향후 연구에 대해서 기술한다.

2. 관련 연구

[그림1]은 5개의 클래스로 구성된 다양한 상속 계층구조를 보여준다. (a)의 경우는 상속계층구조의 너비는 4이면서 깊이는 1인 구조를 갖고있다. 즉 한 클래스로 부터 4개의 클래스가 상속을 받은 생성된 것이다. (b)는 상속계층구조의 너비와 깊이가 모두 2인 구

조를 갖고있다. (c)는 상속계층구조의 너비는 1이고 깊이가 4인 구조를 갖고있다. 따라서 (a)의 경우는 클래스 하나의 코드만 재사용하고 있으며, (b)의 경우는 클래스 2개의 코드를 재사용하고 있고, (c)의 경우는 클래스 4개의 코드를 재사용하고 있다. 그러므로 코드 재사용 측면에서 볼 때는 (c)의 구조가 가장 효율적이지만 프로그램 이해 측면에서는 가장 어려움이 따른다. 왜냐하면 상속을 받은 클래스의 코드를 이해하기 위해서는 상속을 한 코드를 먼저 이해해야 하기 때문이다.

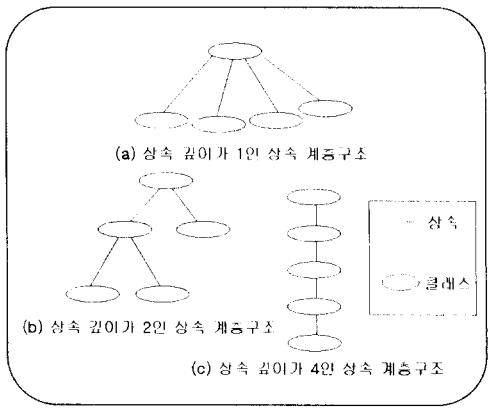

그림1. 5개의 클래스로 구성된 다양한 상속 계층구조

B. Rumbaugh[5]는 객체지향 모델링 및 설계의 관점에서 "서브 클래스(subclass)들을 너무 깊게 내포(nest)시키지 말라. 깊게 내포된 서브 클래스들은 이해하기 어려우며, 이것은 절차적 언어에서 깊게 내포된 블록과 같다. 상속 계층구조의 깊이는 2나 3이 좋으며, 5나 6은 적당할 수도 적당하지 않을 수 있고, 10정도의 깊이는 너무 지나친 것이다" 라고 언급하였다. 따라서 이러한 깊이 관점의 내포 구조와 관련된 직관적·경험적 판단이나 주장을 비교적 타당성이 높은 지침으로 체계화·정량화 하는 작업이 필요하다.

M. Lorenz와 J. Kidd[6]는 Smalltalk와 C++프로젝트들을 중심으로 프로젝트 관점의 척도(Project Metrics)들과 설계 관점의 척도(Design Metrics)들을 여러 가지 측면에서 제시하였다. 그 중에서 상속 깊이에 대해 언급된 척도로서는 '클래스 상속 계층구조의 중첩 수준(class hierarchy nesting level)은 6이하여야 한다(이는 Smalltalk 시스템 자체 프레임워크의 수준까지를 포함한 수임)'이다. 즉, 이 연구에서는 상속 깊이의 제한을 6으로 두고 있는 것이다. 이 연구에서 제시한 척도들은 그들이 몇 년 동안 행한 객체 지향 프로젝트의 결과를 분석해서 얻어진 것이다. 그러나 소프트웨어 개발 작업에 포함되는 모든 인간의 행위(예, 소프트웨어 요구 분석, 설계, 프로그래밍, 테스팅, 유지보수등)는 인간의 인지 능력을 바탕으로 이루어지므로[7, 8, 9] 인지 실험을 통해 클래스 상속 깊이의 기준을 세우는 것이 더욱 의미가 있을 것이다.

3. 객체지향 프로그램의 청크

3.1 청킹 이론과 프로그램 이해

G. Miller[10]는 단기 기억에서의 기억폭(memory span)은 7±2 개의 청크라고 제안하였다. 여기에서 기억 폭이란 항목들을 한 번 제시한 후에 회상해 낼 수 있는 항목의 수이다. 하나의 청크는 '한 입력 문자열을 하나의 구조화된 그룹으로 재배열한 것' 이라고 정의된다. 예를 들어, 하나의 단어는 음소(phonemes)들의 청크이고, 한 문장은 단어들의 청크이다. 인간의 단기 기억에 저장할 수 있는 항목의 수는 제한되어 있으나 항목의 크기를 크게 함으로써(청킹함으로써) 단기 기억에 저장할 수 있는 정보량을 증가시킬 수 있다. 이 가설은 유명하며 단기 기억의 능력과 특성들에 관한 많은 연구들의 기초가 되었다[8, 11].

G. Greeno[7]와 R. Mayer[12]등이 주장한 바에 의하면 프로그램 의미의 인지 과정은 한마디로 덩이화(chunking, 청킹) 과정이라 할 수 있다. 숙련된 프로그래머와 미숙련자를 비교한 경험적 연구에 의하면 덩이

(chunk, 청크)는 프로그램을 이해하는 과정에 있어서 매우 중요한 역할을 한다. 프로그래머는 프로그램을 한 줄씩 읽어 이해하는 것이 아니라 하나의 통일된 기능을 위한 프로그램의 의미 있는 덩이를 발견하고 그 의미를 나름대로 파악한 후, 덩이를 이루는 모든 실행문을 잊어버리고 기능만을 나타내는 간단한 어휘로 기억하게 된다는 것이다. 작은 프로그램의 덩이들이 계속 모아지면 또 다른 차원의 통일된 의미의 덩이를 이루게 된다. 이렇게 덩이화 작업이 계속되면서 프로그래머는 전체 프로그램을 이해할 수 있게 되는 것이다. 프로그램 이해의 과정이 이렇게 진행되는 이유는 G. Miller[10]가 제안한 바와 같이 인간의 단기 기억(short-term memory)에는 한계가 있어서 생소한 프로그램을 접하였을 때 기억할 수 있는 양은 매우 적기 때문이다. 그러나 부분적으로 관심을 집중하여 과거의 경험과 학습된 프로그램 패턴을 적용한다면 덩이들을 찾을 수 있고 그 덩이는 하나의 단어로 표현되어 기억되기 때문에 복잡하고 양이 많은 프로그램을 이해하게 되는 것이다. 결국 프로그램의 이해 과정은 프로그램의 실행문들을 덩이화 작업을 통해서 프로그래머가 이해하는 언어로 바꾸는 작업이며, 프로그램을 개발할 때 쓰이는 상세화(refinement) 개념과 반대되는 추상화(abstraction; 의미 추출) 개념이 적용된다[13].

다음 절에서는 인지 심리학에서의 청크 개념으로 부터 객체지향 프로그램에서의 평면 청크(flat chunk)와 내포 청크(nested chunk)들을 유도해 낸다.

3.2 평면 청크

[표 1]은 인지심리학에서의 청크 개념을 컴퓨터 프로그램의 구성 성분들에 대응시킨 것이다.

한편, 객체지향 프로그램은 절차 지향 프로그램의 구성 성분 외에 메소드(method)나 클래스(class)와 같은 객체지향 개념의 청크들이 추가로 존재한다.

표1. 인지심리학에서의 청크와 프로그램에서의 청크

인지심리학에서의 청크	프로그램에서의 청크
단어	연산자나 피연산자
단어들로 구성된 문장	연산자와 피연산자들로 구성된 프로그램 문장
문장들로 이루어진 단락	코드 라인들로 이루어진 블록(block)
단락들로 이루어진 절(section)	블록들로 이루어진 모듈(module)
절들로 이루어진 장(chapter)	모듈들로 이루어진 파일(file)
장들로 이루진 한 권의 책	파일들로 이루어진 프로그램

따라서, 객체지향 프로그램에서 나타날 수 있는 청크에 대해서 다음과 같이 정의한다.

[정의 1] 하나의 인스턴스 변수(an instance variable)는 기존의 피연산자에 해당하므로 하나의 청크이다.

[정의 2] 하나의 메소드는 일련의 문장들로 구성된 기존의 함수에 해당되므로 하나의 청크이다.

[정의 3] 메소드와 인스턴스 변수들로 이루어진 하나의 클래스는 하나의 청크이다.

[정의 4] 클래스들로 이루어진 하나의 클래스 상속 계층구조(a class inheritance hierarchy)는 하나의 청크이다.

[정의 5] 직접적으로 관련있는 클래스들의 모임 또한 하나의 청크이다.

여기에서 직접적으로 관련있는 클래스들의 모임이란 한 클래스와 그것의 직접적인 서브클래스들의 모임이나 상속과는 무관하면서 직접적인 참조 관계를 갖고 있는 클래스들의 모임을 말한다. 즉, 전자는 'generalization' 또는 'aggregation' 관계를 갖는 상속구조와 관련된 클래스들을 말하며, 후자는 상속구조에는 포함되지 않으나 메시지 송·수신과 관계가 있는 클래스들의 모임을 의미한다.

[정의 5]는 "관련 있는 단어들로 이루어진 숙어 또한 하나의 청크"가 될 수 있다는 개념에서 유도되었다.

3.3 내포 청크(Nested chunk)

[정의 1]~[정의 5]에서 정의한 청크들은 프로그램의 단순 평면구조(flat structure)를 갖는 구성 성분들에 관한 것들이다. 그러나 내포 블록(nested block)처럼 하나의 청크를 이해하는데 다른 청크의 이해가 선행되어야 하는 성분들, 즉 프로그램의 이해 측면에서 내포구조의 성질을 갖는 성분들에 대한 청크의 정의가 별도로 필요하다. 이후부터 전자의 청크를 평면 청크라하고, 후자의 청크를 내포 청크라 칭한다. [그림 2]는 프로그램 이해 관점에서 내포 청크를 보여준다. 이 그림의 의미는 외부 블록, 호출 함수, 자녀 클래스를 이해하기 위해서는 내부 블록, 피호출 함수, 부모 클래스의 이해가 선행되어야 함을 뜻한다.

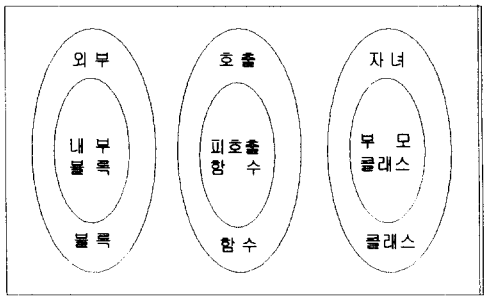

그림2. 프로그램 이해 관점에서의 내포 청크

따라서 객체지향 프로그램에서 내포 청크를 다음과 같이 정의한다.

[정의 6] 호출 메소드(함수)와 그에 대응하는 피호출 메소드(함수) 세트(set)는 하나의 청크이다.

[정의 7] 부모 클래스(parent class)와 그것의 자녀 클래스(child class) 세트는 하나의 청크이다.

[정의 6]은 피호출 메소드가 다시 다른 메소드를 호출한다면 호출된 메소드도 같은 청크에 포함됨을 뜻하며, [정의 7] 또한 서브 클래스가 또 자신의 서브 클래스를 갖는다면 그 서브 클래스까지도 같은 청크에 포함됨을 뜻한다.

G. Miller[10]는 실험을 통해서 인간의 단기 기억에서 평면 청크를 처리할 수 있는 한계가 '7±2 chunks'라고 밝혔다. 이 '7±2 chunks' 이론은 낱자, 단어, 의미 있는 구절(phrase) 등과 같은 일련의 나열된 정보를 인간의 단기 기억에 저장할 수 있는 항목의 수가 7±2개라는 이론이다. 예를 들어 {1 4 9 1 6 2 5 3 6 4 9 6 4 8 2}의 숫자 집합(set)을 기억하는 과제가 주어졌을 때, 인간의 단기 기억에 기억할 수 있는 숫자의 개수는 9개까지가 될 것이라는 것이다. 그러나 이 숫자는 내포 청크에는 해당되지 않는다. 그 이유는 앞에서도 설명했듯이 평면 청크를 인지하는 것보다 내포 청크를 인지하기 위한 노력이 더 필요하기 때문이다.

4. 상속 깊이의 기준을 세우기 위한 실험

이 장에서는 이러한 내포 구조에 대한 인간의 인지적 특성을 알아보기 위하여 실험을 하였다. 즉, 이 실험은 인간의 단기기억에서 내포구조의 청크를 처리하는데 내포의 깊이에 대한 인간의 처리 능력의 한계를 알아보는 것이다. 이 실험에서 사용된 데이터로는 1차원에서 5차원의 단어나 숫자들로 구성된 10가지의 데이터 셀(set)들이며, 각 셀마다 15개의 데이터로 구성되어 있다. 이러한 데이터들을 실험 데이터로 사용한 이유는 프로그램의 단순 평면 구조를 실세계에서의 1차원 정보라고 할 때, 내포 구조는 다차원의 정보(2중 내포구조는 2차원, 3중 내포구조는 3차원)라고 할 수 있기 때문이다.

실험 대상자는 38명의 대학원생으로, 이 실험에서 알아보고자 하는 것은 피 실험자들이 한정된 시간동안에 각 데이터 세트들 내의 데이터들을 얼마나 기억해 낼 수 있는 가를 알아보는 것이다. 따라서 10회의 실험을 하였으며, 각 회마다 주어진 시간은 각 데이터

셀 안의 데이터 수 당 1초씩 계산하여 1차원 데이터 셀의 경우는 1초, 2차원 데이터 셀의 경우는 30초와 같이 주어졌다.

[표 2]는 실험 결과를 보여 준다. [표 2]의 각 셀(cell)안의 숫자들은 피실험자들이 정확하게 회상해낸 데이터의 평균 개수이다. [표 2]에서 알 수 있듯이 1차원 데이터의 결과는 7±2 청크 이론과 일치하고 있다. 이는 실험에서 피실험자들에게 주어진 시간이 적절하였음을 내포하고 있다. 데이터 유형 1의 결과가 이론적 수치보다 약간 크게 나온 이유는 피실험자들이 숫자를 4~5개씩 구분암기하여 기록한데서 기인한다. 이는 또한 제시된 숫자들이 반복하기 쉬운 한 음절로 된 청크라는 점과 관련이 있다. 3차원 이후의 데이터들에 대해서는 결과가 아주 좋지 않았다. 특히 4차원 데이터부터는 기억 개수가 급격히 떨어지는 현상이 나타났다.

표 2. 실험 결과

데이터 그룹 / 차원	문자 데이터 유형	숫자 데이터 유형
1차원	6.1	10.4
2차원	2.3	2.6
3차원	0.9	0.9
4차원	0.2	0.4
5차원	0	0.1

위와 같은 실험 데이터 분석 결과로 보아 우리는 다차원 자극에 대해서 단기 기억에서 정보처리 능력의 한계는 3차원이 한계라는 실험적 결과를 얻을 수 있다. 따라서 본 논문에서는 이러한 실험 결과를 그 동안의 직관적·경험적 관점을 뒷받침할 수 있는 대단히 의미 있는 결과로 보고 다음과 같은 정리를 도출하였다.

[정 리] 프로그램의 내포구조 처리 능력에 있어서 인간의 단기기억의 한계 수는 3이다. 이를 "**내포구조의 한계 수 3**(Limit Number 3 of Nested Structure)"이라고 한다.

이상과 같은 실험을 통해 상속계층구조의 상속 깊이는 3이하가 바람직하다는 결론을 얻을 수 있다. 따라서 그림 1의 (b)의 구조가 상속이 주는 코드의 재사용성을 이용하면서 프로그램 이해 측면에서도 많은 부담을 주지 않기 때문에 (a), (c)보다도 바람직한 구조라 할 수 있다.

5. 결론 및 향후 연구

본 논문은 객체지향 패러다임에서의 상속성의 이점을 잘 살리기 위해 상속 계층구조의 상속 깊이에 대한 기준을 세웠다. 그러기 위해서 먼저 인지 심리학의 청크 개념을 객체지향 프로그램에 적용하여 평면 청크와 내포 청크로 분류하였다. 그 이유는 프로그램을 이해 할 때 청크 단위로 이해하기 때문이다. 내포 청크에 해당되는 객체지향 프로그램의 청크로는 상속 관계를 갖는 클래스들이 해당되는데, 상속 깊이에 대한 기준을 세우기 위해 인지 실험을 하였다. 그 결과 상속 깊이는 3이하가 바람직 한 것으로 나타났다.

본 논문은 객체지향 소프트웨어 개발 생산성에 많은 이점을 주는 상속의 남용으로 인해 오히려 이해하기 어려운 소프트웨어의 개발을 지향하면서, 품질 좋은 소프트웨어의 개발을 할 수 있도록 도울 수 있다.

앞으로는 더욱 많은 실험과 연구를 통하여 인지 심리측면에서 객체지향 설계 지침을 세우는 것이다.

참고문헌

[1] M. Lejter, S. Meyers and S. P. Reiss, "Support for Maintaining Object-Oriented Programs," *IEEE Transactions on Software Engineering*, Vol. 18, No. 12, pp. 1045-1052, Dec. 1992.

[2] N. Wilde and R. Huitt, "Maintenance Support for Object-Oriented Programs," *IEEE Transactions on Software Engineering*, Vol. 18, No. 12, pp. 1038-1044, Dec. 1992.

[3] 문양선, 김재웅, 전형수, 유철중, 장옥배, 김용성, "C++ 프로그램의 이해도 증진을 위한 역공학 시각화 도구", 정보과학회논문지(C), 제1권, 제2호, pp. 160-171, 1995.

[4] P. G. Zvegintzov, *Techniques of Program and System Maintenance,* Second Edition, QED Information Sciences, Inc., 1988.

[5] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design,* Prentice-Hall International, Inc., 1991.

[6] M. Lorenz, J. Kidd, *Object-Oriented Software Metrics*, Prentice-Hall, Inc., 1995.

[7] James G. Greeno, "The Structure of Memory and the Process of Solving Problems," in *Contemporary Issues in Cognitive Psychology,* pp. 103-133. 1973.

[8] B. Shneiderman, "Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results," *International Journal of Computer and Information Sciences,* Vol. 8, No. 3, pp. 219-238, 1979.

[9] V. Arunachalam and W. Sasso, "Cognitive Process in Program Comprehension: An Empirical Analysis in the Context of Software Reengineering," *The Journal of Systems andSoftware*, Vol. 34, No. 3, pp. 177-189, 1996.

[10] G. A. Miller, "The Magical Number Seven Plus or Minus Two : Some Limits on Our Capacity to Process Information," *Psychological Reviews*, Vol. 63, pp. 81-87, 1956.

[11] N. S. Coulter, "Software Science and Cognitive Psychology," *IEEE Transactions on Software Engineering*, Vol. 9, No. 2, pp. 161-171, Mar. 1983.

[12] R. Mayer and B. Shneiderman, "Syntactic/Semantic Interactions in Programmer Behavior: Model and Experimental Results," *International Journal of Computer and Information Sciences*, Vol. 8, pp. 213-238, 1979.

[13] 최은만, 이금석, 홍영식, "프로그램의 이해를 지원하는 소프트웨어 유지보수 도구 세트 개발," 정보과학회논문지, 제21권 제 5호, pp. 900-908, 1994.