Journal of The Korea Society of Computer and Information
Vol. 19, No. 2, February 2014
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2014.19.2.183

웹 소프트웨어의 순환복잡도에 대한 정량적 분석

김 지 현*

A Quantitative Analysis of the Cyclomatic Complexity of the Web Software

JeeHyun Kim*

요 약

본 연구는 웹 소프트웨어의 복잡도와 객체지향 프로그래밍 측정자인 클래스 수(NOC), 메서드 수(NOM)와의 상관관계를 통하여 순환복잡도를 정량적으로 분석하고자 한다. 웹 환경에서 소프트웨어의 복잡도, NOC, NOM의 빈도분포를 근거로 복잡도한계값과 NOC한계값, 복잡도 한계값과 NOM한계값의 상관관계를 파악하기 위하여 실제 사용되는 10개의 웹 프로젝트에서 4,000여개의 ASP 파일이 표본으로 사용되었다. 실험 결과 NOC한계값은 21, NOM한계값은 40이었고 복잡도는 68의 높은 값을 보였으며 10개의 프로젝트 중 NOC, NOM이 특히 높은 빈도를 보이는 2개의 프로젝트를 제외한 8개의 프로젝트는 NOC한계값은 12, NOM한계값은 21이었고 복잡도한계값도 52의 상대적으로 낮은 값을 나타내 상관관계가 있는 것으로 판명이 되었다. 또한 복잡도, NOC, NOM의 한계값이 낮은 8개의 프로젝트는 내부 관리 소프트웨어이었고 나머지 2개의 프로젝트는 외부 매매 서비스 소프트웨어임이 밝혀져 업무 특성에 따라 설계 단계에서 클래스 수, 메서드 수가 결정됨에 따라 복잡도도 미리 추정할 수 있어 품질 향상에 기여할 수 있을 것이다.

▸ Keywords : 순환복잡도, NOC, NOM, 한계값, 웹 소프트웨어

Abstract

In this study Cyclomatic Complexity of Web Software has been analyzed quantitatively by correlation between complexity and Number of Classes(NOC), and Number of Methods(NOM) which are object oriented programming measures. Based on the frequency distribution of complexity, NOC and NOM of software at the Web environment, correlation between complexity threshold and NOC threshold, and NOM threshold has been measured and more than 4,000 ASP

•제1저자 : 김지현 •교신저자 : 김지현
•투고일 : 2014. 2. 3, 심사일 : 2014. 2. 11, 게재확정일 : 2014. 2. 18.
* 서일대학교 컴퓨터소프트웨어과(Dept. of Computer Software, Seoil University)
※ 본 논문은 2012년도 서일대학교 학술연구비에 의해 연구되었음.

files of 10 Web real projects have been used as the sample. The empirical result shows that NOC threshold is 21, NOM threshold is 40 and complexity threshold is 68 as high value, and 8 projects among of 10 except 2 projects with especially high frequency distribution shows that NOC threshold is 12, NOM threshold is 21 and complexity threshold is 52 with relatively low value, so correlation has been found out as exists. And also 8 projects with low threshold of complexity, NOC and NOM were internal management software, and the other 2 projects were external sales service software, Number of Classes and Number of Methods can be defined at the design stage according to business properties , and also the complexity can be estimated in order to improve the software quality.

▸ Keywords : Cyclomatic Complexity, NOC, NOM, threshold, Web software

I. 서 론

소프트웨어 공학에서 측정은 힘든 작업으로 남아 있는데 그 이유는 측정 라이프사이클 전 단계인 설계에서 결과 도출까지 여러 종류의 어려움이 단계별로 나타나기 때문이다. 이전의 작업은 측정 방법이 무엇을 측정하고 범위는 맞는지 확실치 않은 설계 단계와 연관된 어려움이 강조되었다. 또한 모든 어려움이 해결됐을 지라도 측정 결과의 도출은 측정자가 관련 값을 분석하고 결정하도록 유도하는데 이러한 작업이 아직 연구에서 드물게 나타나는 매우 어려운 문제로 남아있다. 예를 들어 자바 소프트웨어가 100개의 클래스를 가질때 '매우 복잡'으로 정의할 수 있는가?이다[1].

이러한 측정과정 중 만나는 주요 문제는 한계값(threshold)의 결정과 관련이 있다. 한계값은 코드의 취약부분을 정의하는 측정과 결합된 숫자 경계를 의미하며 이러한 경계는 측정자가 수용가능 영역 밖의 속성을 정의하게 도움을 준다. 한계값의 범위는 어디까지이고 한계값 사용의 의미를 결정하기 위한 파라메터 정의 방법은 무엇인지에 대한 답은 중대하며 또한 매우 어려운 문제이다. 그것은 한계값에 대한 신뢰 부족과 한계값의 결정을 경험에 기초해 선택하는 것에 원인이 있다. 최대, 최소 한계값은 결과의 개발 도중 의사결정을 돕기 위하여 측정의 응용 전에 정확히 정의되어야 한다. 더구나 최대, 최소 한계값은 그들을 수정할 수 있는 몇 가지 요소에 의존한다는 것은 명백한 것 같다. 그럼에도 불구하고 한계값이 주어진 측정에 제공될 때 그 값들이 사용되는 조건은 잘 정의되지 않으며, 소프트웨어 개발자들은 프로젝트를 측정하지 않고 측정하더라도 기밀로 간주하며 때론 신뢰성을 고려해서 사용하지 않는데 여기에 한계값 적정성 체크의 어려움이 발생하는 것이다[2][3].

선행연구에서 객체지향 소프트웨어에 대한 웹 환경 구조에 따른 복잡도 한계값이 제안되었다. McCabe는 구조적 프로그래밍에서 복잡도 한계값을 10으로 제안하였고, Lopez는 객체지향 프로그래밍에서 복잡도 한계값을 5로 제안하였으며, 선행연구에서는 웹 프로그래밍의 복잡도 한계값이 50으로 제안되었다[3][4][5]. 웹 프로그래밍을 위한 환경 구조는 서버, 클라이언트, HTML의 세 요소가 통합된 것으로 McCabe나 Lopez가 제안한 복잡도 한계값 만으로는 웹 환경 소프트웨어의 높은 복잡도에 대한 설명이 불가능하다.

본 연구에서는 웹 환경에서의 어플리케이션 복잡도에 대한 정량적 분석을 위하여 웹 소프트웨어의 기타 품질 측정자와의 상관관계를 분석하고자 한다. 특히 복잡도와 클래스 수(Number of Classes, 이하 NOC), 복잡도와 메서드 수(Number of Methods, 이하 NOM)의 상관관계 분석으로 상관관계가 있는 것으로 판명되면 웹 소프트웨어의 복잡도를 설계 단계에서 미리 추정이 가능하여 보다 정량적 분석으로 인한 품질 향상에도 기여할 것이다.

논문의 목표는 먼저 웹 어플리케이션에서 순환복잡도, NOC, NOM의 빈도 분포를 계산하고, 이 분포를 기반으로 복잡도한계값과 NOC한계값, 복잡도한계값과 NOM한계값에 대한 실험 작업으로 웹 어플리케이션과 서버와의 복잡도 영역 레벨 차이가 5이상인 복잡도가 높은 어플리케이션의 빈도 분포 분석을 통해 웹 복잡도 한계값, NOC, NOM 한계값과의

상관관계를 파악하는 것이다.

이에 대한 실험을 위해 10개의 웹 사이트 프로젝트에서 ASP어플리케이션의 복잡도, NOC, NOM 빈도 분포에 따른 한계값을 측정한다. 웹 어플리케이션의 구조에 따른 복잡도 산출은 서버 스크립트 요소, 클라이언트 스크립트 요소, HTML 세가지 요소들의 복잡도 합으로 구성되어 있으며 서버 복잡도와 클라이언트 복잡도는 메서드나 모듈의 제어 흐름에서 추출한 순환 복잡도 값을 계산했으며 HTML복잡도는 링크의 수 +1을 계산한 값이다[6].

논문의 구성은 2장에서 패러다임에 따른 복잡도, 복잡도 한계값 측정 및 복잡도 추정 메트릭에 대한 기존의 복잡도 연구 동향을 파악하고, 3장에서 제안하고자 하는 웹 소프트웨어의 복잡도에 대한 연구 내용을 제시하며, 4장에서 실험 대상 표본을 추출하고 실험과정을 제시하며 실험 결과를 통해 복잡도, NOC, NOM 빈도 분포에 의한 복잡도한계값과 NOC한계값, NOM한계값과의 상관관계를 분석 평가한다. 5장에서 결론 및 향후 과제를 제시한다.

II. 기존의 복잡도 측정

1. 패러다임에 따른 복잡도

본 절에서는 절차적 프로그래밍에서의 순환복잡도와 객체지향 프로그래밍에서의 복잡도 및 웹 환경에서의 복잡도에 대해 알아보고자 한다.

McCabe는 소프트웨어의 논리적인 복잡도를 정량적으로 측정하는 메트릭으로 순환 복잡도를 제안하였다. 순환복잡도는 프로그램의 논리 흐름을 표현하는 제어흐름 그래프를 기초로 하는 메트릭으로 구조적 프로그램에서 제어흐름의 복잡도를 효과적으로 측정한다[5].

순환복잡도 CC는 간선의 수 E, 노드의 수 N, 연결 요소의 수 p를 사용하여 다음과 같다.

$$CC = E - N + 2p \quad \text{.....[5][7]}$$

이렇듯 McCabe의 순환복잡도는 구조적 프로그래밍의 복잡도 측정에 이용되어 왔으며 제어 흐름의 복잡도를 매우 효과적으로 측정하는 메트릭으로 최대 한계값이 10으로 그 한계에 대한 어떤 제한 조건도 없다는 특징이 있다. 또한 10이라는 한계값의 사용에 의문을 갖기에는 엄청난 양(695,741개의 파일)의 측정 결과에 기초한 연구이다[2][3].

객체지향 시스템은 여러 단계의 추상화 수준을 가지며 추상화 수준에 따른 측정 메트릭으로 여러 가지가 존재하는데 그 중 하나로 메서드 수준에서의 측정이 있다. 지역변수와 같이 참조된 실행문의 집단으로 '함수', '프로시저', '모듈' 등이 그것이다. McCabe의 순환복잡도, 팬인-팬아웃같은 것이 복잡도 측정에 사용된다[6][8].

객체지향 프로그래밍에서의 복잡도는 모듈이나 메서드의 그래프로부터 계산되는 것으로 Lopez는 자바 프로그램을 위한 복잡도를 계산하기 위하여 McCabe의 순환복잡도 개체는 '한 클래스내의 메서드'로 정의하고 있다. 실험결과 메서드의 80%가 2이하의 복잡도를 가지고 90%가 5이하의 복잡도를 가짐에 따라 객체지향 프로그램의 복잡도 한계값은 5로 제안되었으며 숙련도는 복잡도 한계값과 관련이 없다고 하였다[2][3].

2. 복잡도 한계값 측정

소프트웨어 공학 전 단계에서 측정은 힘드는 작업으로 남아있으며 특히 명확하지 않은 설계 단계와 연관된 어려움을 강조하고 있다. 또한 이러한 어려움이 해결된다고 해도 측정 결과의 이용은 나타나지 않고 있으며, 측정의 이용은 측정자에게 관련 값을 분석하여 의사결정하게 하는데 이 작업이 쉽지 않은 것이다[2].

측정 이용 단계의 중요한 이슈는 과학 분야에서 별로 탐구되지 않고 있는 한계값(threshold)의 결정이다. 또한 주어진 한계값에 대한 신뢰 부족이 측정 결과의 사용을 방해할 수 있으며 실제 소프트웨어 프로젝트에서 발견되지 않은 한계값은 측정자에게 더 좋은 측정을 찾게 하고 측정 결과를 숙고하게 할 수 있다. 한계값을 결정하는 가장 일반적인 방법은 경험에 기초해 간단히 선택하는 것이다. 한계값을 결정하는 다른 방법은 주어진 측정의 실험적 빈도 분포에 대한 개별 값의 비교와 중간으로부터 극적으로 다른 경우에 집중하는 것인데 이러한 접근은 대표적 샘플의 생산이 어렵기 때문에 시간과 노력이 많이 드는 작업이다.

이와 같은 어려움에도 불구하고 한계값을 결정하는 반복적이고 재 생산적인 방법의 제안은 측정의 이용 증진, 검증 향상 및 잠재적 사용 증진을 도모할 것이다[2].

순환복잡도 한계값을 정의하는 방식에 다음과 같은 세 가지 연구가 존재한다.

첫째 방법으로, 복잡도 한계값은 두 가지 속성과 연계되어 결정되는데, 테스팅 노력과 오류밀도 또는 문제의 양과 관련되어 결정하는 방식이다[10]. 여기서 테스팅 노력이나 문제의 양이 많으면 순환복잡도 값이 복잡으로 정의되어 재작업이

요청되는 방식이다[5]. 그러나 이 방식은 너무 무겁고 복잡하며 개발자의 숙련도에 의해 좌우되는 문제점이 있다.

두 번째 방법으로, 16개의 산출물 샘플내에서 순환복잡도 빈도분포를 분석해서 한계값의 분포를 기초로 해서 결정하는 실험 방식이 있다[11]. 이 방식은 프로그램 언어와는 무관하며함수의 50%가 10이하의 복잡도를 가지며 90%는 80이하의 복잡도를 가진다는 실험결과를 제시하였으나 문제에 관한 레포트 수와 유지보수 노력의 상호연관성에 대한 탐구를 설명하였으나, 실험 조건이 정의되지 않았다.

세 번째 방법으로 10 이상의 복잡도를 갖는 자바 메서드의 비율을 확인함으로써 제안된 순환복잡도 한계값의 적정성을 평가하는 방식이다. 자바 프로젝트의 694 소프트웨어 샘플이 작업에 사용되었다. 순환복잡도와 연관된 개체로 자바 프로그램에서는 "클래스내의 메서드"로 정의하였으며 대부분의 산출물 샘플은 웹사이트 라벨이 부여되어 성숙된 것을 대상으로 순환복잡도의 빈도 분포를 계산하여 90% 이상이 5이하의 복잡도를 가진다는 것을 보임으로써 객체지향 자바 프로그램에서는 10의 복잡도 숫자는 별 의미가 없으며 5의 한계값이 적정함을 제안하였다. 측정의 한계값은 개발자의 숙련도와는 관계없음을 제시하였다[2][3].

3. 복잡도 추정 메트릭

한편 객체지향 시스템에 널리 사용되는 복잡도 측정에는 CK(Chidamber & Kemerer) 메트릭과 추상화 수준에 따른 측정 메트릭이 있다.

Chidamber와 Kemerer가 제안한 것으로 복잡도 측정방법에 두 가지가 존재하는데, 하나는 클래스 내 각 메서드의 복잡도 합이고 다른 하나는 메서드 복잡도를 1로 보았을 때 메서드 수의 합, 즉 클래스 당 메서드의 수가 된다고 하였다[6][12].

소프트웨어 추정 메트릭에는 다음과 같은 연구가 존재한다.

1996년 Basili, Brian & Melo는 '품질 지시자 로서의 객체-지향 설계 메트릭의 검증'에서 결함의 수는 메서드 수(NOM), 상속 트리 깊이(DIT), 결합도(CBO), 클래스 응답(RFC)등과 관련이 있다는 것을 증명하였다[10].

1998년 Chidamber, Darcy & Kemerer는 '객체-지향 소프트웨어의 관리자적 사용: 기술적 분석'에서 생산성, 재작업 노력, 설계 노력은 결합도(CBO)와 응집도 부족(LCOM)의 높은 값에 의해 영향 받는다고 제안하였다[13].

1999년 V. Misic과 D. Tesic은 '노력과 복잡도 추정 : 객체-지향 케이스 스터디'에서 노력과 총 클래스 수(NOC) 및 총 메서드 수(NOM)가 밀접한 관련이 있다는 것을 선형회귀분석을 통해 제시하였고 복잡도와 함수 수와의 상관관계를 실험적 탐구를 통해 정의함으로써 이를 설계 단계에서 추정할 수 있다고 제안하였다[8].

또한 2003년 Ronchetti 등은 '객체-지향 환경에서의 소프트웨어 규모 조기 추정'에 관한 연구에서 소프트웨어 개발 노력 대신 규모를 측정하면서 메트릭으로는 LOC를 사용하였고 CMM level 3 소프트웨어 회사에서 개발된 2개 프로젝트를 실험 대상으로 하였다. 실험을 위하여 사용된 총 6개의 메트릭 중 메서드 수(NOM)가 규모(LOC)와 상당한 관련이 있음을 통계적 분석을 통해 제시하였다[14].

이러한 연구는 대부분 객체-지향 언어인 C++ (Li & Henry의 경우만 Ada 사용)로 작성한 프로그램에 대하여 이루어졌으나 본 연구는 웹 프로그래밍 언어인 ASP로 작성된 프로그램을 대상으로 하였다.

III. 웹 소프트웨어의 복잡도

선행연구인 웹 프로그래밍의 복잡도 한계값 분석에서는 웹 소프트웨어를 서버, 클라이언트, HTML 세 요소의 결합으로 보고 서버, 클라이언트 복잡도는 메서드나 모듈의 제어 흐름에서 추출한 순환 복잡도 값을 계산했으며 HTML복잡도는 링크의 수 +1을 계산한 값이다. 웹 소프트웨어의 복잡도 CCWA는 서버의 VB 스크립트 복잡도 C(SSS)와 클라이언트의 자바 스크립트 복잡도 C(CSS), HTML 복잡도 C(HTML)의 합으로 계산된다. 즉 웹 소프트웨어 복잡도 계산식은 CCWA = C(SSS) + C(CSS) + C(HTML)을 사용한다[8].

실제 기업에서 사용되고 있는 10개의 웹 프로젝트에서 4,237개의 ASP파일에 대해 복잡도 빈도분포를 측정한 결과 90%가 50이하의 값을 보임에 따라 한계값이 50으로 제안되었다.

본 연구는 복잡도 한계값 측정의 세 번째 방식인 Lopez의 제안 중 객체지향 언어인 자바 프로그램을 확대하여 웹 어플리케이션에서 복잡도, NOC, NOM 한계값을 추출하여 복잡도와 NOC, 복잡도와 NOM의 상관관계를 분석하고자 한다. 실험을 위해 10가지의 웹 사이트 프로젝트에서 웹 어플리케이션의 구조적 특성상 서버와의 복잡도 영역 레벨 차이가 5이상인 어플리케이션의 복잡도, NOC, NOM 빈도 분포를 파악하여 상관관계를 분석한다.

기존에 이미 구조적 언어(C 언어)와 객체지향 언어(C++ 언어)의 실험 분석에 의하여 검증된 복잡도 인디케이터

(NASA의 SATC(Software Assurance Technical Center)에서 개발한 유지보수성 측정에 사용하기 위한 그래픽 템플릿으로써 영역을 0~6으로 구분하여 영역4,5,6은 결함가능성이 높은 프로그램으로 특별관리가 필요하다고 분석함[7][15].)에 따라 서버측 복잡도 C(SSS)와 웹 어플리케이션의 복잡도(CCWA)의 영역 차이가 5(즉 복잡도 차이가 20)이상인 결함 가능성이 높은 어플리케이션을 추출하여 복잡도, NOC, NOM의 빈도 분포를 파악함으로써 복잡도, NOC, NOM의 한계값 분석을 통한 상관관계를 파악하고자 한다.

논문의 목표는 웹 소프트웨어의 복잡도한계값과 NOC한계값, 복잡도한계값과 NOM한계값의 상관관계를 분석하여 상관관계가 있다면 설계단계에서 웹 프로그래밍 언어로 작성한 소프트웨어의 복잡도를 미리 추정할 수 있을 것이며 결함에 대한 위험도를 줄임으로써 품질 향상에 기여할 수 있을 것이다.

Ⅳ. 실험 및 결과

객관적 자료를 위해 실제 사용되고 있는 웹 사이트 프로그램 파일의 수집을 하였으며 먼저, 실험대상 표본을 추출하고 둘째 실험 과정을 설명하고 셋째 액셀 쉬트의 피벗 테이블과 히스토그램을 통한 통계 분석과 그 결과를 제시하여 웹 환경에서의 복잡도, NOC, NOM의 한계값 및 상관관계에 대해 제안한다.

1. 실험 대상 표본 추출 및 과정

본 연구는 웹 환경에서 ASP 어플리케이션의 복잡도 , NOC, NOM의 빈도분포에 따른 한계값을 추출하여 복잡도와 NOC, 복잡도와 NOM의 상관관계를 분석하는 것이 목적으로 실제 사용되고 있는 10개의 웹 사이트 프로젝트 14,944개의 파일 중 4,237개의 파일에서 웹 어플리케이션 구조에 따른 복잡도, NOC, NOM을 산출하여 빈도 분포를 분석한다.

실험대상 시스템은 ASP로 작성된 웹 어플리케이션으로 서버측은 VBScript, 클라이언트측은 JavaScript로 작성된 시스템이다. 〈표1〉은 실험 대상 시스템의 종류 및 총 파일수와 복잡도를 추출한 ASP파일 수, 웹의 통합복잡도와 서버와의 복잡도 영역 차이가 5이상인 높은 복잡도를 가진 파일을 대상으로 NOC, NOM의 빈도분포를 통한 상관관계를 분석한다.

표1. 실험대상 시스템 구성 및 클래스 추출 파일 수
Table1. Description of source system & no.of files

프로젝트	총파일수	복잡도추출파일수(*.asp등)	클래스추출파일수	프로젝트내용
P01	831	243	8	대기업의 전산 자원 관리 시스템
P02	263	222	12	인터넷 경매 시스템
P03	757	340	4	게시판 중심의 대학 홈페이지
P04	1,658	658	81	상업용 패키지- 팀 다이어리
P05	1,232	407	9	복리후생관리시스템
P06	748	315	23	수발주관리시스템
P07	980	142	4	영업관리시스템
P08	6,106	1,456	70	고객지원센터 시스템
P09	622	269	8	품질관리 시스템
P10	1,747	185	5	프로젝트 관리 시스템
합계	14,944	4,237	224	

실험 대상 시스템은 웹 어플리케이션과 서버와의 복잡도 영역 레벨 차이가 5이상인 복잡도가 높은 어플리케이션을 실험대상 표본으로 추출하여 복잡도 빈도 분포를 분석한다.

측정을 위한 실험 과정이 [그림1]에 나타나 있으며 상세 절차는 다음과 같다.

1) ASP 소스파일을 파서를 통해 토큰을 분리한다. ASP 소스를 구성하는 VBScript, JavaScript, HTML 토큰을 인식하여 분리한다.
2) 파싱처리후 LOC(Lines of Code)와 순환복잡도(CC)를 측정하여 결과를 레파지토리에 저장한다
3) 다시 소스파일을 파싱처리하여 NOC(Number of Classes), NOM(Number of Methods)을 측정하여 결과를 레포지토리에 저장한다.
4) 레포지토리에 저장된 정보를 주어진 질의를 통해 복잡도, NOC, NOM의 빈도 분포를 위해 액셀 쉬트에 옮겨 피벗테이블로 통계처리한다.
5) 피벗테이블의 내용을 토대로 한계값을 추출하고 상관관계를 분석, 평가하고 결과를 출력한다

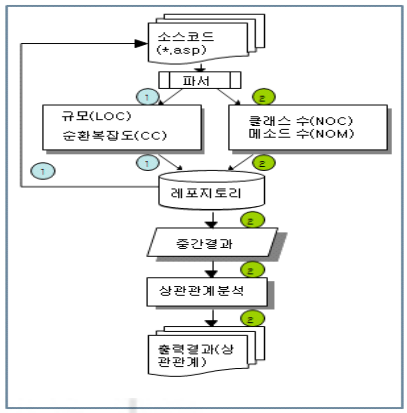

그림 1. 측정 실험 과정
Fig. 1. Measurement experiment process

실험 대상 소스 파일의 통계처리를 위하여 피벗테이블과 히스토그램을 이용하여 한계값을 측정한다. 웹 복잡도와 서버 측 복잡도 영역 레벨 차이가 5이상(복잡도 차이 20 이상)인 프로그램의 복잡도, NOC, NOM 빈도 분포 측정으로 복잡도와 NOC, 복잡도와 NOM의 상관관계를 분석한다.

2. 실험 결과 분석

10개 프로젝트의 통합된 복잡도, NOC, NOM 빈도분포를 측정한 후 각 프로젝트별 특성을 파악하기 위하여 프로젝트별 NOC, NOM 빈도 분포를 측정한 결과 빈도가 특별히 넓게 분포되어 있는 2개의 프로젝트를 발견하여 비슷한 분포를 보이는 8개의 프로젝트에 대한 같은 실험을 진행한다.

〈표 2〉는 10개 프로젝트의 복잡도 빈도 분포를 피벗테이블로 나타낸 것으로 1열은 웹 어플리케이션 순환복잡도CCWA를 나타내며 2열(요약)은 1열의 순환복잡도(CCWA)를 가진 메서드의 개수, 3열(비율)은 1열의 복잡도(CCWA)를 가진 메서드의 비율, 4열(누적갯수)은 1열의 복잡도(CCWA)보다 작거나 같은 복잡도(CCWA)를 가진 메서드 개수, 5열(누적비율)은 1열의 복잡도(CCWA)보다 작거나 같은 복잡도(CCWA)를 가진 메서드의 비율을 나타내고 있다.

복잡도 21~23인 메서드는 9.4%를 차지하며 50이하의 복잡도를 가지는 메서드는 62.9%, 73이하의 복잡도를 가지는 메서드는 90.2%로 복잡도 한계값은 73으로 본다.

표2. 복잡도 빈도분포 (10프로젝트)
Table2. frequency distribution of complexity(10proj.)

CCWA	요약	비율	누적갯수	누적비율
21~23	21	9.4	21	9.4
24~26	6	2.7	27	12.1
28~30	21	9.4	48	21.4
31~33	9	4.0	57	25.4
34~37	9	4.0	66	29.5
38~40	28	12.5	94	42.0
41~43	5	2.2	99	44.2
45~47	5	2.2	104	46.4
48~50	37	16.5	141	62.9
51~53	24	10.7	165	73.7
54~56	9	4.0	174	77.7
57~59	8	3.6	182	81.3
60~63	4	1.8	186	83.0
65~67	12	5.4	198	88.4
68~70	3	1.3	201	89.7
73	1	0.4	202	90.2
74	2	0.9	204	91.1
75	2	0.9	206	92.0
77~82	5	2.2	211	94.2
83~96	3	1.3	214	95.5
106~124	3	1.3	217	96.9
131~173	4	1.8	221	98.7
237~3159	3	1.3	224	100.0
총합	224	100		

표3. NOC 빈도분포 (10프로젝트)
Table3. frequency distribution of NOC(10proj.)

NOC	요약	분포비율	누적수	누적비율
0~2	54	24.1	54	24.1
3~5	16	7.1	70	31.3
6~8	25	11.2	95	42.4
9~11	23	10.3	118	52.7
12~14	24	10.7	142	63.4
15~17	32	14.3	174	77.7
18~20	27	12.1	201	89.7
21	2	0.9	203	90.6
22	13	5.8	216	96.4
23	3	1.3	219	97.8
24~28	4	1.8	223	99.6
35	1	0.4	224	100.0
총합	224	100		

표4. NOM 빈도분포 (10프로젝트)
Table4. frequency distribution of NOM(10proj.)

NOM	요약	분포비율	누적수	누적비율
0~2	47	21.0	47	21.0
3~5	12	5.4	59	26.3
6~8	9	4.0	68	30.4
9~11	13	5.8	81	36.2
12~14	14	6.3	95	42.4
15~17	15	6.7	110	49.1
18~20	9	4.0	119	53.1
21~24	16	7.1	135	60.3
25~29	8	3.6	143	63.8
30~32	29	12.9	172	76.8
33~35	17	7.6	189	84.4
36~38	8	3.6	197	87.9
39	3	1.3	200	89.3
40	10	4.5	210	93.8
41	4	1.8	214	95.5
43~45	5	2.2	219	97.8
46~52	4	1.8	223	99.6
53	1	0.4	224	100.0
총합	224	100		

〈표 3〉은 10개 프로젝트의 NOC 빈도 분포를 피벗테이블로 나타낸 것으로 클래스 수(NOC) 21이하가 90.6%, 28이하가 99.6%로 NOC한계값은 21이다.

〈표 4〉는 10개 프로젝트의 NOM 빈도 분포를 피벗테이블로 나타낸 것으로 메서드 수(NOM) 40이하가 93.8%, 52이하가 99.6%로 NOM한계값은 40이다.

여기서 10개 프로젝트 각각의 NOC, NOM의 빈도 분포를 측정하였다. 〈표 5〉가 각 프로젝트별 클래스 수(NOC)의 빈도분포를 나타내며 〈표 6〉은 각 프로젝트별 메서드 수(NOM)의 빈도분포를 나타내고 있다. 이들 표를 보면 P02, P04를 제외한 나머지 8개의 프로젝트는 NOC의 대부분이 15이하(1건 예외)이며 NOM의 대부분은 28이하(4건예외)의 분포를 보이고 있다. P02, P04 프로젝트는 NOC 값이 13이상의 값으로 치우쳐 있거나 널리 분포되어 있었고 NOM은 25이상의 값으로 치우쳐 있거나 널리 분포되어 있었다.

표5. 각 프로젝트별 NOC 빈도분포 (10프로젝트)
Table5. frequency distribution of NOC for each projects(10pj.)

NOC	요약	p01	p02	p03	p04	p05	p06	p07	p08	p09	p10	총합	분포비율
0~2	54	2			1				51			54	24.1
3~5	16	3		1		1			11			16	31.3
6~8	25	2			7	4	4		2	6		25	42.4
9~11	23			1	2		11	3	3		3	23	52.7
12~14	24		2	2	2	4	8	1	2	1	2	24	63.4
15~17	32		6		24				1	1		32	77.7
18~20	27		1		26							27	89.7
21	2				2							2	90.6
22	13		1		12							13	96.4
23	3		1		2							3	97.8
24~28	4		1		3							4	99.6
35	1	1										1	100
총합	224	8	12	4	81	9	23	4	70	8	5	224	

표6. 각 프로젝트별 NOM 빈도분포 (10프로젝트)
Table6. frequency distribution of NOM for each projects(10pj.)

NOM	요약	p01	p02	p03	p04	p05	p06	p07	p08	p09	p10	총합	분포비율
0~2	47				1				46			47	21.0
3~5	12			1					11			12	26.3
6~8	9	1			1	3	4					9	30.4
9~11	13	2		1	5				5			13	36.2
12~14	14	2			2		1	2	1	6		14	42.4
15~17	15	1					12	1			1	15	49.1
18~20	9						5		1	1	2	9	53.1
21~24	16	1			2	4	1	1	4	1	2	16	60.3
25~29	8		2		5	1						8	63.8
30~32	29		1		28							29	76.8
33~35	17		6		11							17	84.4
36~38	8				8							8	87.9
39	3				3							3	89.3
40	10				9	1						10	93.8
41	4				4							4	95.5
43~45	5	1	1	2	1							5	97.8
46~52	4		1		1				2			4	99.6
53	1		1									1	100
총합	224	8	12	4	81	9	23	4	70	8	5	224	

10개 프로젝트 중 큰 값의 클래스 수(NOC), 메서드 수(NOM)를 가지는 P02, P04 두 프로젝트를 제거하고 8개 프로젝트의 복잡도와 NOC, NOM의 빈도분포를 다시 분석하였다. <표 7>은 8개 프로젝트의 복잡도 빈도분포를 나타내고 <표 8>은 8개 프로젝트의 NOC 빈도분포, <표 9>는 8개 프로젝트의 NOM 빈도 분포를 나타내고 있다.

8개 프로젝트의 복잡도는 52이하의 복잡도가 92.4%를 차지하여 10개 프로젝트의 90.2%가 73이하의 복잡도를 가지는 것과 큰 차이를 보인다. 8개 프로젝트의 NOC는 12이하가 94.7%이며 15이하가 99.2%로 한계값은 12로 볼 수 있다. 또한 NOM은 21이하가 92.4%이며 28이하가 95.4%로 한계값은 21로 볼 수 있다.

표7. 복잡도 빈도분포 (8프로젝트)
Table7. frequency distribution of complexity(8proj.)

CCWA	요약	비율	누적갯수	누적비율
21~23	20	15.3	20	15.3
24~26	6	4.6	26	19.8
28~30	20	15.3	46	35.1
31~33	8	6.1	54	41.2
34~38	5	3.8	59	45.0
39~42	19	14.5	78	59.5
45~47	3	2.3	81	61.8
48~50	29	22.1	110	84.0
51	2	1.5	112	85.5
52	9	6.9	121	92.4
53	1	0.8	122	93.1
54~65	4	3.1	126	96.2
95~290	4	3.1	130	99.2
3159	1	0.8	131	100.0
총합	131	100		

표8. 각 프로젝트별 NOC 빈도분포 (8프로젝트)
Table8. frequency distribution of NOC for each projects (8proj.)

NOC	요약	p01	p03	p05	p06	p07	p08	p09	p10	총합	분포비율
0~2	53	2					51			53	40.5
3~5	16	3	1	1			11			16	52.7
6~8	18	2		4	4		2	6		18	66.4
9~11	21		1		11	3	3		3	21	82.4
12	16			4	7	1	2	1	1	16	94.7
13	1				1					1	95.4
14	3		2						1	3	97.7
15~17	2						1	1		2	99.2
18~20	0									0	99.2
21~23	0									0	99.2
24~28	0									0	99.2
35	1	1								1	100
총합	131	8	4	9	23	4	70	8	5	131	

표9. 각 프로젝트별 NOM 빈도분포 (8프로젝트)
Table9. frequency distribution of NOM for each projects (8pj.)

NOM	요약	p01	p03	p05	p06	p07	p08	p09	p10	총합	분포비율
0~2	46						46			46	35.1
3~5	12		1				11			12	44.3
6~8	8	1		3	4					8	50.4
9~11	8	2	1				5			8	56.5
12~14	12	2			1	2	1	6		12	65.6
15~17	15	1			12	1			1	15	77.1
18~20	9				5		1	1	2	9	84.0
21	11			4	1	1	3	1	1	11	92.4
22	2	1							1	2	93.9
24	1						1			1	94.7
25~29	1			1						1	95.4
30~32	0									0	95.4
33~35	0									0	95.4
36~38	0									0	95.4
39~41	1			1						1	96.2
43~45	3	1	2							3	98.5
46~52	2						2			2	100
53	0									0	100
총합	131	8	4	9	23	4	70	8	5	131	

여기서 한가지 알 수 있는 것은 복잡도 한계값이 커지면 NOC, NOM 한계값도 따라서 커지며 복잡도, NOC, NOM 한계값이 낮은 8개의 프로젝트는 내부 관리 소프트웨어이며 복잡도, NOC, NOM 한계값이 상대적으로 높은 2개의 프로젝

트는 매매나 판매등 외부 서비스 소프트웨어라는 것이다.

표10. 프로젝트 특성에 따른 비교
Table10. Comparison of projects properties

	10개 프로젝트	8개 프로젝트
복잡도한계값	68	52
NOC한계값	21	12
NOM한계값	40	21
프로젝트 특성	내부관리 프로그램 및 매매사이트	내부관리프로그램사이트

〈표 10〉의 프로젝트 특성에 따른 비교에서도 알 수 있듯이 복잡도 한계값이 높은 10개 프로젝트는 NOC, NOM 한계값도 크고 복잡도 한계값이 낮은 8개 프로젝트는 NOC, NOM 한계값도 작다. 복잡도와 NOC, 복잡도와 NOM은 상관관계가 있음을 보였고, 특히 NOC와 NOM은 밀접한 상관관계를 보였는데, NOM 한계값은 NOC 한계값의 두 배 정도의 값을 가짐을 알 수 있다.

복잡도, NOC, NOM한계값이 낮은 8개 프로젝트는 기업 내부 관리 프로그램이며 NOC, NOM한계값이 높은 2개 프로젝트는 인터넷 경매사이트와 판매용 패키지인 팀다이어리 사이트로 상업용 서비스 소프트웨어가 내부관리 소프트웨어보다 복잡도가 훨씬 높은 분포임을 알 수 있다.

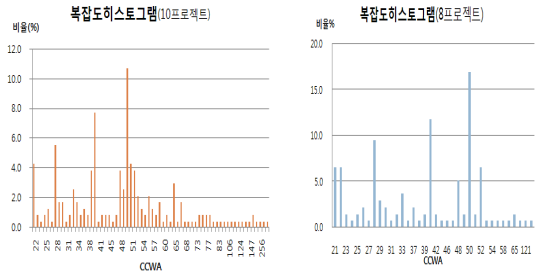

그림 2. 복잡도빈도분포 히스토그램(10개 & 8개프로젝트)
Fig. 2. frequency distribution of complexity(10 & 8proj.)

〔그림2〕는 복잡도 빈도 분포를 나타내는 히스토그램으로 X좌표는 복잡도이며, Y좌표는 같은 복잡도를 가진 메서드의 비율을 나타낸 것이다. 10개 프로젝트와 8개 프로젝트에 대한 빈도분포에서 한계값은 현저한 차이가 있으나 가장 높은 빈도를 나타내는 복잡도는 두 집단 모두 50의 값을 가지는 비율이 10.9와 16.9로 가장 높은 빈도를 보이고 있다. 그러나 이 부분은 연구의 범위를 벗어나므로 고려 대상에서 제외하기로 한다.

본 연구의 표본은 복잡도가 현저히 높은 프로그램을 위주로 한 것이기 때문에 일반적인 웹 어플리케이션 보다 위험도가 크므로 고려해야 할 사항이 더 많이 요구되며, 이 부분에 대해서는 향후 웹 어플리케이션의 복잡도와 관련된 숨어 있는 특성은 없는지 관련성을 찾아보는 노력이 필요할 것이다.

V. 결론

본 연구는 서버, 클라이언트, HTML이 복합된 구조의 웹 환경에서 통합 어플리케이션의 복잡도, NOC, NOM의 빈도분포를 파악하고 한계값에 대한 복잡도와 NOC, 복잡도와 NOM의 상관관계를 분석한 결과 상관관계가 있는 것으로 나타났다. 실험 측정 결과 10개 프로젝트에서 복잡도 한계값이 68일 때 NOC한계값은 21, NOM한계값은 40이었고, NOC와 NOM의 빈도분포가 낮은 8개 프로젝트의 복잡도 한계값이 52일 때 NOC한계값은 12, NOM한계값은 21의 값을 가짐으로써 복잡도 한계값이 크면 NOC, NOM의 한계값도 크고 복잡도 한계값이 하락하면 NOC, NOM 한계값도 하락함을 알 수 있었다. NOM한계값은 NOC한계값의 2배 정도의 값을 나타내었다. 또한 복잡도, NOC, NOM의 한계값이 낮은 8프로젝트는 기업 내부 관리 프로그램이며, 복잡도, NOC, NOM의 한계값이 높은 2프로젝트는 인터넷 경매 사이트와 상업용 패키지인 팀 다이어리 사이트로 상업용 서비스 소프트웨어가 내부 관리 소프트웨어보다 복잡도도 높고 클래스 수나 메서드 수가 훨씬 많이 소요됨을 알 수 있었다. 이런 결과는 내부 관리 소프트웨어보다 외부 서비스 소프트웨어의 클래스 수나 메서드 수가 더욱 많이 소요되며 따라서 복잡도도 높아지므로 품질관리에 특히 주의해야 함을 뒷받침하고 있다.

이 실험은 웹 어플리케이션과 서버와의 복잡도 영역 레벨차가 5(복잡도 20의 차이)이상인 복잡도가 매우 높은 위험도가 큰 어플리케이션에 대한 실험으로써 기업에서 실제 사용하고 있는 검증된 모집단임에도 불구하고 제한적인 측면이 있다.

향후 복잡도가 특별히 높은 소프트웨어 위주가 아닌 일반적인 웹 소프트웨어의 복잡도에 대한 정량적 추정이 가능한 메트릭의 연구가 필요할 것이다.

참고문헌

〔1〕 Habra, N., Abran A., Lopez, M. & Paulus, V., "Towards a framework for Measurement Lifecycle," University of Namur, Technical Report, TR37/ Apr. 2004

〔2〕 Miguel Lopez, Naji Habra, "Relevance of the

Cyclomatic Complexity Threshold for the Java Programming Language," In Proceedings of the Software Measurement European Forum(SMEF 2005), Mar. 2005

〔3〕 JeeHyun Kim, "Relevance of the Cyclomatic Complexity Threshold for Web Programming," Journal of the Korea Society of Computer and Information, vol. 17, no. 6, Jun. 2012

〔4〕 Miguel Lopez, Gregory Seront, Naji Habra, "A Quantitative Analysis of the Cyclomatic Complexity of the Java Development Framework J2SDK," In Proceedings of SMEF2006, May 2006

〔5〕 T. J. McCabe, , "A Complexity Measure," IEEE Transactions on Software Engineering, Vol. 2, No. 4, Dec. 1976

〔6〕 JeeHyun Kim, HaeYoung Yoo, "A Study of Estimation for Web Software Size," Journal of the Korea Information Processing Society, vol. 12-D. no. 3, May 2005

〔7〕 JeeHyun Kim, Chel Park, "Implementation of the Module Risk Levels Measurement Tools for Web Application," Journal of the Korea Society of Computer and Information, vol. 7, no. 2, Jun. 2002

〔8〕 V.B.Misic, D.N.Tesic, "Estimation of effort and complexity: An object-oriented case study," Journal of Systems and Software, Jan. 1999

〔9〕 Chel Park, HaeYoung Yoo, "Analysis of Cyclomatic Complexity for Web Application", Journal of the Korea Information Processing Society, v.11-D, n.4, pp.865-872, Aug. 2004

〔10〕 V.R. Basili, L.C. Briand, and W.L. Melo, "A Validation of Object Oriented Design Metrics as Quality Indicators," IEEE Trans. Software Eng., vol. 22, no. 10, pp. 751-761, Oct. 1996

〔11〕 Stark G., Durst, R.C., " Using Metrics in Management Decision Making," Computer (IEEE), 1994

〔12〕 Victor Laing and Charles Coleman, "Principal Components of Orthogonal Object-Oriented Metrics," www.gsfc.nasa..gov/support/OSMASA SMSEP01/,Oct. 2001

〔13〕 Chidamber, S.R., D.P. Darcy, and C.F. Kemerer, "Managerial Use of Object-Oriented Software : An Explanatory Analysis," IEEE Transactions on Software Engineering, 24(8), Aug. 1998

〔14〕 Marco Ronchetti, Giancarlo Succi, Witold Pedrycz, Barbara Russo, "Early estimation of software size in object-oriented environments : a case study in a CMM level 3 software firm, " www.unibz .it/web4archiv/objects/pdf/cs_library/1/, 2003

〔15〕 Linda Rosenberg, Ph.D., Lawrence Hyatt, "Developing a successful metrics program," International Conference On Software Engineering(IASTED) SanFrancisco CA, Nov. 1997

〔16〕 Britoe e Abreu, P., Poels, G., Sahraoui H.A., and Zuse, H., "Quantitative Approaches in Object-Oriented Software Engineering," Kogan Page Science, 2004

〔17〕 Abran, A., Lopez, M., and Habra, N., "An Analysis of the McCabe Cyclomatic Complexity Number," in IWSM Proceedings, Berlin, 2004

〔18〕 Thomas A. Powell , "WebSite Engineering," Prentice Hall PTR, 1998

저 자 소 개

김 지 현

1978: 이화여자대학교 수학과 이학사

1994: 단국대학교 전자정보전공 경영학석사

2005: 단국대학교 전산통계학과 이학박사

현재: 서일대학교 컴퓨터소프트웨어과 교수

관심분야: 웹공학, 데이터베이스, 품질 관리

Email : jhkim@seoil.ac.kr