통신 소프트웨어의 복잡도 분석 사례 연구

이재기, 신상권, 남상식, 김창봉*
ETRI, *공주대학교
전화 :042-860-4806 팩스: 042-860-6344 E-mail : jkilee@etri.re.kr

An analysis of the switching Software Matrics : Case study

Jaeki Lee, Sangkwan Shin, Sangsik Nam, Changbong Kim
ETRI, Kongju University
Tel :042-860-4806 Fax: 042-860-6344 E-mail : jkilee@etri.re.kr

Abstract

The software complexity model makes an estimated of the product software. For a practice of software managed, need to guideline of the static analysis. Especially, Software complexity model introduced for the estimation of software quantity. In case of measurement the software matrices, its need for us to analysis of software quality and products.

In this paper, we represent that the analysis of function point, control structure and interface, volume matrices in various aspect of switching software. Others, their results utilized similar of project and system development.

색인어 : 복잡성, 매트릭스, 교차수, 노드, 프로세스, 리소스, 결함(defect), function point.

I. 서 론

일반적으로 소프트웨어 복잡성은 소프트웨어 작성(coding)이나 이해가 난해한 부분이 나타나는 특성이 있을 때 크게 2가지 관점인 이론적 복잡성(theoretical complexity)과 심리적 복잡성(psychological complexity)으로 구분된다.

이론적 복잡성은 소프트웨어 고유의 성질을 반영, 소프트웨어 구현(실현)의 문제나 알고리즘의 복잡성에 대한 것이다. 이것은 소프트웨어의 본질적인 성질에 착안한 연산(혹은 계산)의 복잡성(computational complexity)과 현상적인 성질에 근거한 프로그램의 복잡성으로 구분된다.

전자는 프로그램 실현 시 수반되는 문제로써 요구사항을 반영하는데 나타나는 현상으로 이런 것들을 이론적 복잡성이라 칭한다. 다시 말해서 문제의 해결을 구하는 알고리즘을 실행 머신 위에서 실행하는데 필요한 메모리 용량이나 수행시간(execution time)이 필요하다. 소프트웨어 science 분야의 제반 계산상의 복잡성이 여기에 속한다. 후자는 설계상의 프로그램 결과가 구조적인 복잡성에 대해서 프로그램을 구현하는 것이다. 여기에는 의미적 구조와 형식적 구조에 착안한 2개의 복잡성이 존재하게 된다. 의미적 구조의 복잡성에 대한 연구 사례는 거의 없으나 형식적 복잡성에 대해서는 여러 측면에서 다수의 연구 결과가 발표되고 있다. 이것에 대한 연구 결과는 주로 프로그램에 포함된 정보량이나 조건 분기수로 표시하는 복잡성 연구가 대표적이다. 즉, 주요 소스코드에 대해 소스 비용을 계측 대상으로 삼아 예측하는데, 이러한 연구는 소프트웨어 설계 평가 모델[1]에서 다루어지고 있다.

심리적 복잡성은 프로그램의 작성이나 이해가 난해한 경우 사람에 의해 주관적인 경험을 통해서 얻는 복잡성이다. 즉, 심리적 복잡성은 여러 소프트웨어의 구현에 관계되는 시간이나 비용으로 표현되는 복잡성이다. 이것을 정리하면 아래 그림 1과 같다.

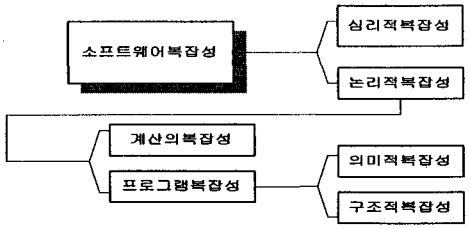

그림 1. 소프트웨어 복잡성의 분류

본 논문의 구성은 1장에서 소프트웨어 복잡성에 대해 살펴보고 2 ~ 3장에서 프로그램 복잡성, 제어구조, 데이터구조, 복합 매트릭스, 프로세스 복잡도에 의한 여러 가지 매트릭스 분석기법에 대해 알아본다. 그리고 4장에서 프로그램 복잡도에 근거한 방법 및 제어구조, 인터페이스 구조 및 function point(F.P) 등 복합 매트릭스 분석 기법을 채택하여 개발중인 ATM 교환기의 프로젝트에 적용한 결과를 분석하고 향후 고려할 사항에 대해 언급하고 맺는다.

II. Program Complexity metrics

프로그램의 복잡성을 수치로 표현하는 경우 프로그램의 신뢰성이나 보수의 용이성을 표시하는 지표와 이용을 목적으로 한다. 프로그램 복잡성의 측정치를 프로그램의 신뢰성이나 유지보수성의 지표로 이용하는데 있어서 프로그램 복잡성을 계측할 수 있는 Metrics를 제공할 때 프로그램의 신뢰성이나 보수성과의 관계를 정량적으로 명확히 밝힐 수 있어야 한다. 이런 입장으로부터 프로그램의 복잡성 Metrics에 대하여 McCabe의 척도나 Halsted의 척도 등 표에 정리된 것과 같이 많은 연구가 수행되고 있다. 대표적인 예로는 Fitzsimmons & Love[2]에 의해 Halsted의 소프트웨어 공학이론에 입각한 실험적 평가, Curtis et al.[3]에 의한 McCabe 척도에 대한 통계적 수법을 이용한 평가, 春原[8]에 의한 실용시스템의 데이터에 기초한 분석 예가 있다.

● Volume metrics

Volume metrics는 프로그램의 크기에 착안한 척도로 이 경우의 매트릭스는 프로그램 소스코드의 행수(스탭 수, LOC, 스텝 멘트 수 등), Halsted의 소프트웨어 Science 이론, 기능 수(Function Point 수)가 대표적이다. LOC는 가장 간단하게 사용되는 매트릭스이나 한편으로는 프로그램의 크기로 복잡성을 표시하는 경우 프로그램의 질적인 특성을 반영하기가 어렵다.

● Control structure metrics

제어구조 매트릭스(control structure metrics)는 프로그램의 제어 구조에 착안하여 가장 활발히 제안되고 있는 매트릭스로 대표적인 것은 McCabe cyclomatic numbe[3], chen의 최대교차수(maximal intersect number : MIN)[7], woodward의 knot count[4], zolnowski[6]의 depth of nesting 등이 있다.

● Data structure metrics

프로그램 내의 데이터 양은 프로그램의 복잡성을 나타내는 주요 요인이다. 프로그램에서 취급하는 데이터량이 많음은 프로그램의 복잡성을 증가시킨다는 것을 경험적으로 알 수 있다. 데이터 량이나 참조특성을 계측하는 매트릭스를 데이터 구조 매트릭스(data structure metrics)라고 부른다.

● composite metrics

복합 매트릭스는 프로그램의 복잡성을 전체적으로 평가하는데 이용되는 매트릭스로서 프로그램의 복잡성 요인의 도출, 프로그램의 복잡성 요인의 계측, 프로그램의 복잡성 요인의 분석, 프로그램 구조척도의 도출, 프로그램 구조척도의 적용 순서로 다양하게 평가할 수 있다.

복잡성 요인은 프로그램 구조의 데이터 구조, 데이터 참조, 처리, 제어구조, 인터페이스 등 5가지로 구분하여 도출하고 복잡성 요인에 대한 계측은 언어요소의 출현수를 계측, 언어요소의 출현 유/무 계측, 언어요소의 출현 상태를 수치화하는 경우로 구분, 평가할 수 있다. 위에서 언급한 복잡성 metrics 기법을 분류, 정리하면 그림 2와 같다.

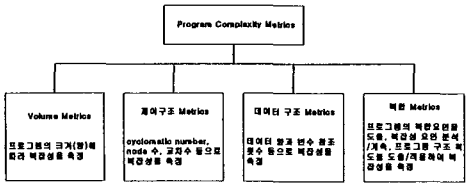

그림 2. 프로그램 복잡성 Metrics 분류

III. Process Complexity Metrics

프로세스 복잡성 매트릭스 기법은 프로세스 복잡성 요인과 프로세스 복잡성 요인 계측법으로 구분되며, 이것은 각 요인에 대한 5가지 특성에 따라 분류되어 계측된다.

● 프로세스 복잡성 요인

개발요원(개발자), 리소스, 개발기법, 툴, 발주자와의 인터페이스, 산출물에 대한 인터페이스로 구분된다.

● 프로그램 개조 및 재 이용

프로그램 개조는 프로그램 전체 에러율과 밀접한 관계를 갖으며, 개조부문의 규모에 대한 신규 작성 부문으로 측정되고 프로그램의 재 이용은 2가지 측면으로 분류되어 측정할 수 있다. 즉, 재이용된 모듈로부터 재이용 비율을 측정해 보는 시각과 재 이용한 프로그램으로 보는 시각이다. 이에 대한 연구 결과는 1991년에 Galdier, G and Basli, V. R이 모듈의 재 이용 횟수로 프로그램의 재 이용에 대한 연구 결과가 발표된 바 있다.[5]

VI. 통신 소프트웨어 분석 결과

● 시스템 프로그램 분석

프로세스 복잡성 요인 중 개발요원 특성에 대한 분석 결과는 각 기능별로 프로그램 숙련도가 아래 표 1과 같으며, 개발요원 변동율은 대략 공동개발업체 인력을 포함하여 최고 33%, 최저 15%, 프로그램 숙련자율은 53%로 파악되었다.

표 1. 개발요원 특성 분석 결과 (단위 : 년)

기능 별	경험치(AVG.)	담당블럭수
호 제어(CP)	11.3	5.6
운용보전(O&M)	7.3	6.2
운영체제	7.3	1.0
DBMS	5.0	1.25
망관리(TMN)	2.5	1.0
Firmware(DC)	8.2	1.0

● DC : device controller(장치 제어계)

● 교환기 프로그램 경력 10년 이상자를 숙련자로 가정

또 전체 프로그램 규모와 프로그램 재사용 관련 프로그램 개조율과 재 이용율은 분석에 많은 어려움이 수반되나 그 결과는 표 2와 같다.

● 제어구조 및 인터페이스 구조 분석 결과

기능별 제어구조 분석결과는 표 3과 같다. 초기 시스템인 SV5.1.2 버전의 호처리 프로그램의 제어 구조는 각각의 프로토콜을 구분하지 않고 하나의 기능 블록 내에서 통합, 처리함으로써 프로그램내의 제어구조 횟수가 적었으나 소스 규모가 커지는 구조로 되었으며, 점차 시스템의 기능이 세분화 됨에 따라 각 프로토콜 별 기능을 세분화하여 처리함으로 인해 최종 목표 시스템으로 갈수록 프로그램에서 많은 제어구조를 사용하게 되었다.

표 2. 프로그램 규모, 개조 및 재 이용 현황

구분	비율(%)	전체 프로그램규모
개조율	17.2	1310 KLOC
재 이용율	16.3	(121 Blk, 147 Function)

한편, OAM 기능은 초기에 복잡한 기능을 통합함으로 인해 프로그램의 파일 개수는 적은 반면 규모나 제어구조를 많이 사용하게 되고 점차 파일수는 기능을 세분화하여 용도에 맞게 기능을 구현함으로써 평균 제어수는 줄어드는 추세이다. 이 이유는 시스템의 설계 방법이 분산환경 및 객체지향 개념을 도입하고 CHILL에서 C/C++로 전환함으로 C-언어의 특징을 살려 파일 소스 규모는 다소 변동이 없으면서 다양한 기능을 구현할 수 있는 장점을 살렸기 때문이다.

변경 횟수는 호처리 기능이 별다른 차이가 없으나 OAM 기능은 많은 차이를 보인다. 그 이유는 객체지향개념의 설계방법 도입과 분산구조 환경으로 복잡한 기능을 분리함으로써 중앙집중식 관리의 부담을 해소하고 사전 구현 기능에 대한 모델링을 수행함으로 에러를 줄일 수 있었다.

표 4에 언급된 분석 결과는 소스 프로그램의 readability를 향상시키기 위한 주석라인(comment line)수는 대체로 1 ~ 2% 정도로 분석되었으며 F.P는 운용기능이 호처리나 보전기능, 데이터 처리 기능에 비해 상대적으로 많이 사용되었다. 특히 운용기능은 다른 기능 범주에 비해 다양한 상태관리나 일정주기로 처리하는 일이 많은데다가 시스템 운용에 필요한 OAM 기능 등을 지원하는 경우의 수가 많아 F.P의 사용이 타 기능에 비해 많다.

호처리 기능은 주로 main 시스템 내에서 IPC(inter processor communication) 처리로 기능 등을 수행하는 관계로 constant나 구조체(structure) 구문을 많이 사용하는 것으로 분석되는데 그 이유는 다양한 프로토콜 처리 등을 위해 프로그램에서 미리 선언해 놓은 값들을 불러 사용하기 때문이다. 각 기능들을 처리하기에 적합하도록 파라미터 값들을 선언하고 프로그램에서 반복 사용하기 때문인 것으로 분석된다.

보전기능은 초기버전에서 global 변수, F.P, constant 선언, 구조체문, 매크로 선언 등이 많이 사용되었으나 최종 버전에서는 현격히 줄어드는 현상을 보인다. 그 이유는 초기에 시스템 형상 및 기능 정립이 확실하게 정해지지 않은 상태에서 임의로 선언하여 사용

해 왔으나 점차 시스템의 형상 확정과 완성도가 높아짐에 따른 기능의 재정립이 이루어져 불필요하게 사용되었던 변수, 구조체문, 매크로 선언 등을 정리하여 프로그램의 완성도를 높였던 것으로 분석되었다.

● function Point(F.P) 분석에 의한 소프트웨어 Benchmark

Function Point Analysis(FPA) 방법에 의한 소프트웨어 프로젝트 규모나 비용, 인도시기 등을 분석하는 방법이 많이 제기되고 있다.[9] 이에 대한 대표적인 방법이 Putnam에 의해 1975년 제기되어 이를 보완하여 프로젝트 분석 및 프로그램 분석에 활발히 활용되고 있다. 이점에 착안하여 ATM Project에서 수행된 결과를 분석하면 표 5과 같다.

분석한 결과 프로젝트의 F.P 평균치를 적용하는 경우의 work effort(person-hours)는 비슷한 수준이나 delivery speed는 60% 정도에 지나지 않는다. 다시 말해서 개발기간이 길어져 적기에 소프트웨어를 인도하지 못할 위험성을 안고 있는 격이다. 대체적으로 project duration도 50 ~ 70%인 것으로 분석되었다. 전체적인 개발경험은 풍부하나 동일 환경에 대한 경험이 많은 편으로 디버깅 환경 및 개발환경의 개선, 새로운 개념의 개발경험 등이 필요한 것으로 밝혀졌다.

V. 결 론

교환 소프트웨어 개발 관리에 주로 사용되어 왔던 방법은 주로 개발된 프로그램을 동적인 환경에서 실행시켜 얻어진 결과를 토대로 소프트웨어 개발 프로젝트를 평가하는 소프트웨어 신뢰도 성장 모델을 사용하여 왔다. 이 방법은 개발된 프로그램에 대한 분석이 용이치 않아 프로젝트 관리 방법에는 잘 활용되지 못했다.

본 논문은 이러한 단점을 보완하고 개발된 프로그램에 대한 정적 분석과 동적 분석을 병행하여 소프트웨어 품질을 향상시키고 소프트웨어에 대한 신뢰도를 향상시킬 수 있는 방법의 일환으로 교환 시스템에 대한 소스 프로그램을 복잡도 모델의 기초 이론에 입각하여 프로그램의 규모를 분석함으로써, 교환 소프트웨어의 품질 척도와 신뢰도를 향상시킬 수 있는 틀을 만들었다. 이러한 분석 정보들은 프로젝트 관리 즉, 전체 개발 진행 현황 파악을 위한 유용한 정보로 활용이 가능하다.

향후 연구방향으로는 다양한 품질 척도 방법과 프로젝트 견적 모델인 개발비용을 기초로 한 연구생산성 평가 모델 개발과 이에 대한 적용방법의 연구가 필요하다.

참 고 문 헌

[1] 山田 戊, 高橋 宗雄, ソフトウェアマネジメントモデル入門 - ソフトウェア品質の可視化と評價法, pp.25-46, 共立出版社, 1993.

[2] Fitzsimmons, A. and Love, T., "A review and evolution of software science", ACM Computing surveys, Vol. 10, No.1, pp. 3-18, 1978.

[3] Curtis, B., Sheppard, S. B., Milliman, P., Borst, M. A. and Love, T., "Measuring the psychological complexity of software maintenance tasks with the Halsted and McCabe metrics", IEEE Trans. Software Engineering, Vol. SE-5, No. 2, pp. 96-104, 1979.

[4] Woodword, M. R., Hennell, M. A. and Hedly, D., "A measure of control flow complexity in program text:, IEEE Trans. Software Engineering, Vol. SE-5, No. 1, pp. 45-50, 1979.

[5] Galdier, G. and Basili, V. R., " Identifying and qualifying reuseable software components" IEEE Computer, Vol. 24, No. 2, pp. 61-70, 1991.

[6] Zolnowski, J. C. and Simmons, D. B., "Taking the measure of program complexity", Proc. National Computer Conf., pp. 329-336, 1981.

[7] Chen, E. T., "Program complexity and programmer productivity", IEEE Trans. Software Engineering, Vol. SE-4, No. 3, pp- 187-194, 1978.

[8] Sunohara, T., Takano, A., Uehara, K. and Ohkawa, T., "Program complexity measure for software development management", Proc. 5th Int. Conf. Software Engineering, pp. 100-106, 1981.

[9] Yen Cheung, Rob Willis and Barrie Milne, "Software benchmarks using function point analysis", Benchmarking : An International Journal, Vol. 6 No. 3, pp. 269-279, 1999.