□研究論文□

韓國 軍事運營分析 學會誌
第 15 卷, 第 2 號, 1989. 12. 31.

Ada 言語를 이용한 X시스템의 設計, 具現 및 再使用性 分析에 관한 硏究 (A Study on Design, Implementation and Reusability Analysis of the X system using Ada Language)

이광건, 강석균, 박형춘, 윤창섭*

Abstract

Reusing software is a promising solution which overcomes the software crisis in the software development environment by improving the software productivity, reliability and maintainability. To develop software, some developers reuse existing softwares, but without an adequate design environment, software reuse would not be applicable.

The purpose of this paper is to design and implement the X system, and measure empirically its resuability in the development phases. The development processes of the X system used a software reuse design guidelines within the Object-Oriented Design method and the Ada programming language.

In order to measure the resuability, metrics was proposed and applied for each phase of development. The results of this paper address some reusability measurement that may be used as basic data when estimating the amount of reuse for a combined development project of military applications.

* 國防大學院

Ⅰ. 序 論

소프트웨어의 生産費用과 生産하고자 하는 소프트웨어의 數와 量은 엄청나게 增加하고 있는 반면에, 소프트웨어의 信賴度는 減少되고, 開發期間이 지연되며, 소프트웨어의 運用整備가 점점 어려워져 소프트웨어의 危機가 더욱 심화될 것으로 전망하고 있다. [1]

電算費用(cost of computing)중에서 소프트웨어 費用이 지배적인 고려요소가 되고 있다. [2] 소프트웨어의 開發과 開發完了 이후의 運用整備에 소요되는 費用增大의 원인은 소프트웨어 규모가 대형화, 복잡화되면서 開發에 투입되는 高級人力의 인건비 증대와 生産性 向上의 저조, 運用整備의 대상이 되는 소프트웨어의 量的 增加와, 誤謬修正과 調整 및 性能改善을 위한 勞東集約的인 處理過程 때문이다. [3]

소프트웨어 再使用(software reuse)은 소프트웨어의 品質과 生産性을 改善하여, 결과적으로 소프트웨어 開發期間을 단축하며, 開發과 運用整備費用을 절감하는데 寄與하여 소프트웨어 危機를 극복할 수 있는 工學的 시도로 알려져 있다. [4]

기존의 소프트웨어는 再使用性을 고려하지 않았기 때문에 再使用할 수가 없다. 갖고자 하는 소프트웨어가 再使用性을 염두에 두지 않고 設計되었다면 비록 소프트웨어의 코딩과 文書化가 잘되어 있고, 지금까지 알려진 基本的인 최적의 設計原則을 충실히 준수하였다 하더라도 소프트웨어 構成品 간의 相互 接續關係가 그것을 再使用하기에 부적합하므로 再使用의 기회는 적다. [5]

본 論文에서는 再使用性을 지원하는 對象中心設計 技法과 再使用에 대한 設計指針을 적용하여 A集團의 X시스템을 Ada프로그래밍 言語環境에서 設計 및 具現하고, 소프트웨어 壽命週期(software life cycle) 段階別로 再使用性을 측정하고자 한다.

再使用性의 測定結果는 앞으로 예상되는 軍事分野의 소프트웨어를 통합 開發시에 再使用의 정도를 예측하는 基礎資料가 될 수 있을 것이다.

Ⅱ. 再使用性(reusability)의 槪念

1. 定 義

再使用性이란 소프트웨어 構成單位(component)가 갖고 있는 能力으로써 그 構成單位가 최초에 開發되었던 應用分野(application)가 아닌 새로운 分野에서 반복적으로 使用되는 能力을 말한다. 효율적으로 再使用되기 위하여 새로운 應用分野의 要求事項에 적합하도록 기존의 소프트웨어 構成單位를 修正하여 再使用할 수도 있다. [4]

再使用性과 移植性(portability)의 槪念이 가끔 相互 同一한 意味로 혼용되고 있으나 移植

性은 특정 應用分野 전체 소프트웨어가 갖추고 있는 能力으로써 최초의 開發目的 環境이 아닌 새로운 環境에서 반복 再使用되는 能力을 뜻하며, 새로운 運用環境에 적합하도록 修正作業이 이루어질 수도 있다.〔4〕 즉, 전체 소프트웨어(entire application)가 새로운 環境으로 移植된다고 하고, 소프트웨어의 構成單位가 새로운 應用分野의 소프트웨어 構成單位로 再使用된다고 하여야 한다.

2. 소프트웨어 再使用의 고려 대상

一般的으로 소프트웨어 再使用의 대상을 開發 이후에 運用 중인 원시코드를 反復使用하는 狹義의 의미로 인식하고 있다. 그러나 소프트웨어 再使用에 고려되는 대상은 〔5〕 소프트웨어 開發方法論(methodology), 要求事項과 設計에 관한 明細書(specification), 원시코드와 모듈(module), 文書(documentation), 소프트웨어 道具 및 開發環境(tools and support environment), 分析資料(analysis data), 試驗資料(test information), 運用整備에 관한 資料(maintenance information base) 등 이다.

위에서 제시한 소프트웨어의 무엇을 再使用할 것인가를 決定하기 위하여 소프트웨어 壽命週期의 全體 段階에서 生産되는 모든 生産物과 그 生産物이 만들어지는 處理過程도 그 對象으로 고려될 수가 있다.〔6〕

3. 再使用의 잇점과 障礙要素

소프트웨어를 再使用함으로 얻어지는 잇점은 아래와 같다.〔4〕

- 開發 및 運用整備 費用의 節減
- 生産性 向上과 開發期間의 短縮
- 소프트웨어 信賴度의 改善
- 資源의 效率的 活用

소프트웨어 壽命週期의 段階別 生産物을 再使用하게 되면 그 자체를 直接開發하는 것보다 경제적인 면에서 費用節減의 잇점이 있고, 이미 기존의 應用分野에서 使用되는 과정에서 信賴度가 개선된 生産物을 再使用하게 되므로 信賴度와 品質改善의 효과도 기대할 수 있다. 硏究報告書〔4〕에 의하면 設計, 코딩, 試驗, 文書化 作業努力의 60%를 節減할 수 있었고, 또한 프로그래머가 learning curve에 의하여 세번 이상만 동일한 論理構造(logic structure)를 反復使用하게 되면 50%의 生産性을 向上한다고 보고 되고 있다. 또한 프로그래머가 작성하지 않았던 프로그램이라도 프로그램을 修正할 때에 再使用의 設計指針에 의한 一貫性있는 프로그램 形式(consistent programming style)을 이해하고 있다면 運用整備段階에서도 효율적이라고 하였다.

소프트웨어 再使用은 經濟的, 技術的인 관점에서 바람직한 방향이면서 동시에 기존의 傳統的인 開發原則의 修正과 變化를 초래하게 한다. 하나의 開發組織에서 再使用 槪念을 적

용하고 이를 開發段階에서 실행하는데는 初期에 많은 어려움이 따른다. 再使用에 따른 장애요소를 열거하면 아래와 같다. [5]

- 소프트웨어 開發管理者와 開發擔當者들의 抵抗
- 再使用 技術適用 동기의 결여
- 再使用에 부적당한 기존 소프트웨어의 設計
- 標準化의 不在
- 社會的, 法的 障礙 등이다.

Ⅲ. 再使用性을 위한 設計指針

1. 對象中心設計

소프트웨어의 再使用을 위한 소프트웨어 開發方法論이 對象中心 開發方法論이며, [7, 8, 9, 10] 對象中心 프로그래밍 言語를 使用한다. 對象中心의 言語(object-oriented language)는 情報隱蔽(information hiding), 資料의 抽象化(data abstraction), 動的連結(dynamic binding), 相續性(inheritance)의 네가지 특성을 지원할 수 있어야 한다. [8, 11] 對象中心設計란 對象(object)이라는 槪念에 입각하여 시스템을 분할하는 소프트웨어 開發方法論이다.

對象中心設計는 기존의 전통적인 機能中心設計와는 근본적인 차이가 있다. 機能中心設計의 대표적인 것은 Yourdon의 Structured analysis and Design (SASD) [12]인데 機能中心設計에서 시스템의 분할은 機能을 中心으로 분할한다. 즉, 問題領域에서 주요 機能을 추출하고 그 機能에 해당하는 모듈에서 다루어야 할 알고리즘의 抽象化에 중점을 두고 있으며 프로그래밍 言語는 機能 분할에 적합한 FORTRAN, C, COBOL과 같은 言語이다. 일반적으로 機能中心設計는 아래와 같은 制限事項이 있다.

- 資料의 抽象化(data abstraction)와 情報隱蔽(information hiding)를 효과적으로 처리하지 못한다.
- 問題領域(problem space)에서 同時性을 가진 自然的인 現象을 다루기에 不適合하다.
- 問題領域의 변경사항에 대하여 적응력이 결여되어 있다.

反面에 對象中心設計 方法은 위에서 언급한 制限事項들을 다룰 수 있는 특성을 가지고 있다.

對象中心 設計方法은 機能中心設計 方法과는 완전히 다른 方法으로 문제를 인식하고 設計한다. 즉 실세계에서 일어나는 對象을 中心으로 시스템을 모형화한다. 對象中心設計는 對象을 中心으로 시스템을 분할하기 때문에 資料를 抽象化할 수 있으며, Ada, C++, SM ALLTALK와 같은 對象中心 프로그래밍 言語를 사용한다. [8] 對象中心으로 設計한 소프트웨어는 시스템을 변경시 변경에 해당하는 對象만을 수정하면 되므로, 변경에 의한 波及效

果를 局部化(locality)시킬 수 있다는 장점을 갖는다.〔7〕

다음은 이상에서 살펴본 對象中心設計가 갖는 잇점을 간략히 要約한 것이다.

- 프로그래머의 生産性向上과 運用整備費用의 減少로 전체 소프트웨어의 壽命週期費用을 줄일 수 있다.
- 시스템의 변경시 波及効果가 局部化되므로 프로그램의 變更과 擴張이 容易하다.
- 실세계의 현상과, 이에 대응하는 프로그램의 모형을 자연스럽고 直接的으로 연관하여 시스템을 開發할 수 있다.

2. 對象中心의 開發過程

對象中心 設計로 시스템을 분할하는 基準은 시스템을 구성하는 각각의 모듈이 問題領域에서 일어나는 對象 또는 對象의 集合을 표현하도록 하는 것이다. 對象中心設計의 일반적인 過程은 1) 對象과 對象의 속성(attributes)들을 식별, 2) 對象의 施行作業(operations)을 식별, 3) 對象과 對象間의 관계(visibility)를 규정, 4) 對象의 接續關係(inteface)를 설정, 5) 對象을 실제로 具現(implement)하는 것이다.〔2, 7, 13〕

본 論文에서는 자료흐름 分析(data flow analysis)모델을 利用한 對象中心設計 方法으로써, 기존의 Booch〔7〕, Stark〔14〕, Jalote〔15〕등이 제시한 設計方法을 개선하여 對象과 對象의 作業을 구체적으로 식별할 수 있는 設計方法〔16〕을 適用하였는데 要約하면 자료흐름도(DFD : Data Flow Diagram)내의 자료저장소(data store)와 자료흐름(data flow)을 對象으로 식별하고, DFD의 變換(process)을 對象의 作業으로 식별한 다음 DFD 내에 산재해 있는 여러개의 對象들을 同一한 것인가, 다른 對象의 一部分인가의 관점에서 통합하여 전체 시스템을 분할하는 設計方法을 適用하였다.

3. Ada 言語의 特性

Ada 프로그래밍 言語는 美國 國防省에서 소프트웨어의 危機를 극복하기 위하여 開發되었으며, 國防에 관한 모든 시스템에 적합할 뿐만 아니라 상업, 硏究分野 등에도 적합한 言語이며 특히 소프트웨어 工學의 원리인 資料의 抽象化(abstraction), 情報隱蔽(information hidding), 모듈화(modularity) 및 局部化(locality) 槪念을 지원하고 簡潔性, 信賴性, 効率性, 維持補修性를 向上시키며, 人間活動에 맞는 프로그램 작성(programming as a human activity)을 지원한다.〔2, 17〕

Ada 言語의 特性은 모든 프로그램 변수의 명확한 선언, 변하지 않는 type의 特性, 구문의 自然語的 表現(English like)으로 오류 可能性을 최소화 하고 있다.〔9, 10, 18〕 또한 Ada 言語는 독립적으로 여러 사람, 여러 장

소에서 分散開發한 소프트웨어 構成品(component)을 조립하여 완전한 하나의 프로그램으로 구축이 可能하게 하며, 이러한 分散開發의 槪念은 다양한 프로그램 단위와 個別 컴파일(separate compliation)로 지원된다.

Ada 프로그램 단위는 明細部分과 實行部分이 분할(separate)되어 있다. 明細部分은 사용자에게 필요한 情報를 보여주는 部分으로 다른 프로그램 단위에서 사용할 수 있도록 한다. 實行部分은 具現의 세부사항을 포함하고 있고 사용자로부터 은폐되어 있는 部分이며, 알고리즘을 기술하고 있다. (Ada manual [17] 참조)

분할은 또한 하나의 프로그램 단위가 물리적으로 연속일 필요가 없이 個別 컴파일이 가능하도록 하므로 대형 프로그램을 보다 작은 규모의 단순한 프로그램으로 분리할 수 있도록 한다. 그리고 프로그램 단위의 라이브러리를 생성할 수 있도록 하므로, 소프트웨어의 開發을 단순화시킨다. 또한 프로그램의 校正과 更新(correction and update)을 效率的으로 하며, 컴파일하는 費用의 節減이 가능하게 한다. [2, 9, 10]

Ada 言語의 프로그램 단위는 副프로그램(subprogram), 패키지(package), 타스크(task), 一般化 프로그램 단위(generic program units) 등이 있다.

가. 副프로그램(subprogram)

副프로그램은 procedure와 function으로 분류되는데, 이는 전통적인 프로그래밍 言語에서 다루는 procedure, function과 유사하다. (Ada manual [17] 참조)

나. 패키지(package)

패키지는 논리적으로 관련된 실체(logically related entities) 또는 계산자원(computational resources)을 내포(encapsulate)하고 있으며 形式(type), 副프로그램, 타스크 그리고 다른 패키지까지도 포함할 수 있다.

패키지의 첫번째 이용분야는 논리적으로 관계되는 변수, 상수, 형식들을 하나의 패키지 안에 정의하는 경우이다. 變更事項이 발생할 경우 그 패키지만을 變更하면 되므로 管理가 용이하다.

두번째 이용분야는 논리적으로 관련된 프로그램 단위(procedure, function)들을 집단화하는 것이다.

세번째 이용분야는 抽象化 자료형식(abstract data type)을 정의하는 경우이다.

네번째 이용분야는 상태를 갖는 실체와, 한 상태에서 다른 상태로 變更되는 조작을 제어하기 위해 즉, 상태기계(state machine) 또는 오토메타(automata)를 抽象化하는 데에 패키지를 이용하는 경우이다.

다. 一般化 프로그램 단위(generic program units)

Ada 言語에서 프로그램을 因數化(parame-

terization)하는 메카니즘이 一般化 프로그램 단위이며, 一般化 프로그램 단위는 一般化 패키지와 一般化 副프로그램이 있다. 단순히 자료의 形式이 다르고 알고리즘이 같은 경우 각각의 프로그램을 작성하지 않고 一般化 프로그램 단위로 작성하여 對象의 부류(class of object)를 표현할 수가 있도록 하고 있다.

Ada 言語의 一般化 프로그램 단위들은 하나의 틀(template)이기 때문에 직접 실행시킬 수 없으며 translation time에 실례화(instantiation)를 통하여 object code로 실행될 수가 있다.

一般化 프로그램 단위들의 形態는 프로그램 단위인 副프로그램이나 패키지의 明細部分 앞에 'generic'으로 시작하는 generic part가 있으며, 明細部分과 實行部分은 프로그램 단위와 같다.

이러한 一般化 프로그램 단위는 再使用 소프트웨어 구성단위로 사용된다. 자료형식이 다르고 알고리즘이 같을 경우에 여러개의 구성요소로 만드는 것보다 一般化 단위로 매개변수화하여 크고, 복잡한 시스템을 간단하게 만들 수 있다. 開發者는 一般化 프로그램 단위의 잇점을 이용하여 새로운 코드를 적게 작성하도록 하여 生産性을 向上시킬 수 있고 소프트웨어의 信賴性과 品質의 向上을 달성할 수 있다.

Ada 프로그래밍 言語에서 再使用性을 직접 지원하는 特性을 要約하면 아래와 같다.

- 副프로그램, 패키지, 타스크 등 프로그램 단위가 다양하여 이들을 利用한 프로그램 구성에서 再使用이 용이하다.
- 可視的(visible)인 明細部分과 具現의 상세부분인 實行部分이 분리되어 있으므로 모듈의 세부적인 具現에 관계없이 다른 모듈과 접속이 가능하다.
- 컴파일 時間에 모듈을 호출하는 실매개변수와 형식 매개변수의 形式(type)을 확인(check)하여 接續關係를 명확히 하고, 오류발생을 최소화 한다.
- 一般化 프로그램 단위는 소프트웨어 구성단위를 하나의 틀로 인수화하여 再使用을 直接的으로 지원한다.
- 個別컴파일로 프로그램 library를 구축할 수 있다.

4. 再使用 소프트웨어 設計指針

가. 모델의 使用

일반적으로 모델(model)은 공통된 관점을 정의하는 데에 사용된다. 소프트웨어 開發시 분석가들은 分析 모델을 사용하여 소프트웨어의 要求事項을 分析하고 정의할 수도 있고, 이러한 要求事項들을 기존의 能力과 비교하여 얼마 만큼이나 일치 하는가를 판단한다. [19]

소프트웨어의 再使用을 고려시에 해결할 문제 중 하나는 기존의 소프트웨어를 再使用하

기 위하여 새로운 시스템을 어떻게 設計하는가 이고 또 다른 문제는 開發하고자 하는 소프트웨어가 지금까지는 具現되지 않았지만, 그러나 장차 예상되는 여러 형태의 새로운 시스템에서 再使用할 수 있는 소프트웨어를 어떻게 設計할 것인가 이다. 이러한 문제들이 모델의 사용으로 해결 가능하다. 再使用 가능한 소프트웨어를 具現하기 위한 모델을 형성하는 過程은 실세계의 문제를 인식하고 그 문제와 관련되는 발생 가능한 문제들을 추출하여 이를 一般化하고, 一般化된 機能의 모델을 형성하는 것이다. [19]

나. 階層構造의 設計

階層構造란 시스템을 순서화된 階層으로 분할(partitioned)한 구조를 의미하여 분할된 階層은 그 階層에 적합한 機能만을 실행한다.

階層構造의 設計概念은 <그림. 1>에서 제시

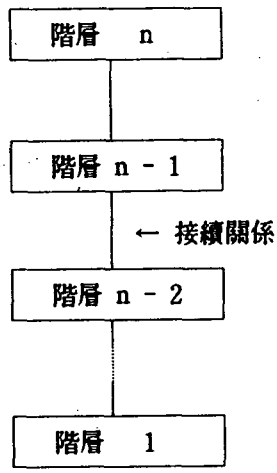

그림. 1 계층구조의 설계

한 바와 같이 소프트웨어가 수행할 전체의 機能을 階層으로 設計하고 階層間의 接續關係를 설정하는 것이다. [5]

소프트웨어를 階層構造로 설계하게 되면 個別階層이 독립적으로 분할된 機能을 수행함으로 再使用하려고할 때에는 이를 대치하여 再使用의 목적에 알맞도록 조정할 수가 있다.

또한 대치할 때나, 變更 및 수정을 할때에 波及效果를 국부화할 수 있으므로 수정노력을 최소화하게 하고 階層間의 接續關係를 標準化한다면 個別階層의 再使用 기회를 증진하는 효과도 있다.

다. 接續關係의 고려

소프트웨어 構成單位의 接續關係는 그 구성단위가 外部環境과 관련되는 역할을 규정하고 있고 接續關係를 이용하여 構成單位를 명확히 설명할 수가 있다.

소프트웨어 設計者는 再使用性을 제고하기 위하여 接續關係를 명확히 인식하고, 이를 設計하여야 하고, 완벽한 文書化 過程을 통하여 정의하고 필요시 참조할 수 있도록 文書로 남겨야 한다.

接續關係를 設計함에 있어 設計者는 모듈이 機能面에서 높은 凝集性(high cohesion)를 갖도록 하고 모듈간의 접속은 낮은 結合度(low coupling)를 유지하도록 設計하여야 한다. [13]

接續關係의 設計指針은 융통성과 일반성(flexibility and generality)을 갖도록 設計하

는 것이다. 과도한 접속은 오히려 再使用 가능한 경우를 제한할 수도 있다. 일반적으로 알려진 바로는 적은 수의 단순한 接續關係를 갖고 있는 構成單位가 再使用의 可能性이 더욱 높다는 것이다.

接續關係는 의도적인 사용 범위내에서 因數化되면서(parameterized) 또한 목적에 알맞게 조정(customized)될 수 있도록 設計되어야 再使用이 가능하다. 또한 接續關係는 모듈의 활용범위를 제한하여 모든 경우의 機能을 하나의 모듈로 처리하지 않도록 設計하여야 한다.

라. 效率性의 選擇

再使用을 위한 소프트웨어 設計는 과도한 一般化 작업으로 인하여 그 성능이 저하되고, 사용자 관점에서 불필요한 機能이 추가된 상태의 소프트웨어 構成品이 될 수도 있다. 效率性에 관한 否定的인 要因은 과도한 因數化, 발생 빈도가 낮은 경우의 처리를 위한 코드의 集合, 코드의 變更없이 再使用하기 위한 副프로그램 呼出構造의 추가 및 여러 機能을 하나로 결합한 모듈 등이다.

Ada 言語 環境에서 效率性을 개선하기 위한 방법은 一般化(generic)處理, inline code, constant folding과 dead code elimination, subunit化 등이 있다.〔17〕 즉 과도한 인수화는 generic으로, 발생이 희박한 경우에 해당하는 코드의 集合은 constant folding과 dead code elimination으로, 副프로그램의 呼出構造의 추가는 inline으로, 여러 機能을 수행하는 副프로그램의 非效率性은 subunit化 처리로 效率性을 개선할 수 있다.

Ⅳ. 연구대상 시스템의 設計 및 具現

1. 연구대상 시스템의 要求事項 分析

연구대상으로 삼은 시스템은 A 집단에서 a_i (i=1, 2, 3) 組織別로 각각 별도로 적용하고 있는 人事業務 중에서 공통성이 비교적 많다고 생각되는 進級資料評價 업무에 관한 부분만을 다루는 시스템이다.

進級資料評價 업무를 X 시스템이라 하였을 때에 Xa_i는 a_i 組織에서 실행되는 進級資料評價 업무를 말한다.

Xa_i시스템은 a_i組織에서 실행되는 進級評價에 요구되는 進級對象者 서열명부를 만들기 위하여 個人進級記錄을 각각 점수로 계산하고 點數別로 서열명부를 작성하는 소프트웨어 시스템이다. 要求事項 分析은 a_i組織의 關係規定을〔20, 21, 22〕 사용하였으며, Yourdon의 SASD를 이용한 構造的 分析 및 下向式 構造的 設計(top-down structured design) 技法을 이용하였다.〔12〕

본 論文에서는 人事業務에 대한 데이타 베이스(DataBase)가 구축되어 있다고 가정하였다. 設計시에 데이타 베이스를 억세스(access)

하는 모듈의 機能은 제외하였으며, 要求事項 分析結果의 DFD와 資料辭典 및 소단위명세서는 參考文獻[23]에 제시되어 있다.

2. 시스템의 設計

X 시스템의 設計는 a. 組織別 要求事項 分析에서 사용한 자료흐름도, 資料辭典, 要求事項明細書를 基準으로 對象中心設計 技法[16]을 適用하여 Ada 프로그래밍 環境하에서 設計하였다. <그림. 2>는 設計된 Booch-도형이며, 근무평정을 계산하기 위하여 '평균', '구간점수변환', '등급점수'의 3개의 프로그램 단위를 이용하고 있음을 나타내고 있다.

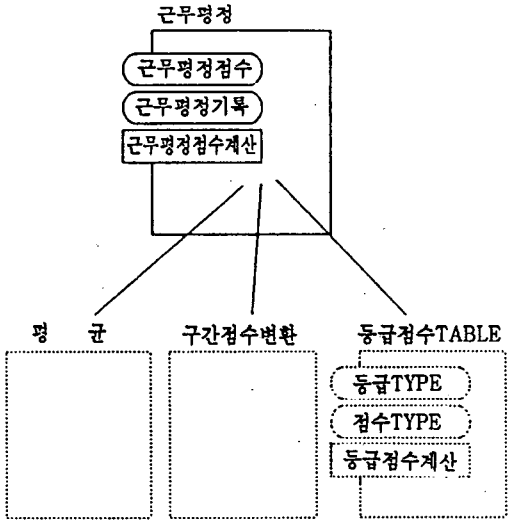

그림 2 Booch-도형의 예

'구간점수변환'은 점수 範圍別로 점수를 계산하는 기능이다. 구간점수변환의 明細部分은 아래와 같으며 一般化 값 매개변수(generic value parameter)를 사용하고 있다. 즉 2번 line에서 12번 line까지의 값 매개변수는 상수로 취급하며 사용자가 상수의 범위를 부여할 수 있다.

```
1  generic
2   BYEONHWAN_JEOMSU_1 : INTEGER :=95;
3   BYEONHWAN_JEOMSU_2 : INTEGER :=88;
4   BYEONHWAN_JEOMSU_3 : INTEGER :=85;
5   BYEONHWAN_JEOMSU_4 : INTEGER :=75;
6   BYEONHWAN_JEOMSU_5 : INTEGER :=65;
7   GUGAN_JEOMSU_1 : FLOAT :=100.0;
8   GUGAN_JEOMSU_2 : FLOAT :=90.0;
9   GUGAN_JEOMSU_3 : FLOAT :=88.0;
10  GUGAN_JEOMSU_4 : FLOAT :=80.0;
11  GUGAN_JEOMSU_5 : FLOAT :=70.0;
12  GUGAN_JEOMSU_6 : FLOAT :=60.0;
13 procedure GUGAN_JEOMSU_BYEONHWAN(GUGAN_
14 JEOMSU :in FLOAT; JEOMSU :out INTEGER);
```

設計過程 및 設計의 최종적인 生産物 즉, Booch-도형 및 프로그램 明細部分(specification)은 參考文獻[23]에 제시되어 있다.

3. 시스템의 具現

X시스템 具現은 設計段階의 최종적인 生産

```
1 procedure GUGAN_JEOMSU_BYEONHWAN(GUGAN_
2 JEOMSU:in FLOAT; JEOMSU:out INTEGER)is
3  begin
4   if GUGAN_JEOMSU <= GUGAN_JEOMSU_1 and
5    then GUGAN_JEOMSU >= GUGAN_JEOMSU_2
6      then JEOMSU := BYEONHWAN_JEOMSU_1
7    elsif GUGAN_JEOMSU < GUGAN_JEOMSU_2
8     and GUGAN_JEOMSU >= GUGAN_JEOMSU_3
9      then JEOMSU := BYEONHWAN_JEOMSU_2
10   elsif GUGAN_JEOMSU < GUGAN_JEOMSU_3
11    and GUGAN_JEOMSU >= GUGAN_JEOMSU_4
12    then  JEOMSU := BYEONHWAN_JEOMSU_3
13   elsif GUGAN_JEOMSU < GUGAN_JEOMSU_4
14    and GUGAN_JEOMSU >= GUGAN_JEOMSU_5
15    then  JEOMSU := BYEONHWAN_JEOMSU_4
16   else   JEOMSU := BYEONHWAN_JEOMSU_5
17   end if;
18  end GUGAN_JEOMSU_BYEONHWAN;
```

物인 프로그램 明細部分을 토대로 實行部分(body)을 Ada 言語로 再使用性을 고려하여 구현하였다. 아래의 프로그램은 구간점수변환을 具現한 예로 實行部分을 나타내고 있다. 위의 프로그램 예에 대하여 아래의 프로그램 1-5라인은 一般化 프로그램 단위를 실례화(instantiation)시키는 것이며, 특히 3-4 라인은 明細部分의 값 95, 88, 85, 75, 65를 각각 35, 34, 33, 31, 31로 사용자가 상수의 값을 부여하고 있다. 또한 6-9라인은 실례화된 프로그램을 호출하여 원하는 계산을 하고 있음을 보여주고 있다.

```
1  procedure PYEONGGYUN_TO_JEOMSU is new
2                GUGAN_JEOMSU_BYEONHWAN;
3  procedure PYEONGGYUN_TO_GEUNMU_JEOMSU
4     is  new GUGAN_JEOMSU_BYEONHWAN(35,
5                          34, 33, 31, 31);
6 PYEONGGYUN_TO_GEUNMU_JEOMSU(PYEONGGYUN,
7                          INTEGER(JEOMSU));
8 PYEONGGYUN_TO_JEOMSU(PYEONGGYUN,integer
9                   (GEUNMU_JEOMSU(1)));
```

위의 예에서 X 시스템의 一般化 프로그램 단위는 再使用 可能하도록 설계되었으며, 실례화를 통하여 하나의 組織에서 한번 開發한 소프트웨어 構成品을 다른 組織에서 수정없이 반복적으로 사용할 수 있도록 具現하였다.

연구대상 시스템은 개인용 컴퓨터(PC)상에서 JANUS/Ada 202* 컴파일러로 具現하였고, 원시코드는 參考文獻[24]에 지시되어 있다.

* R.R. Software Inc.의 등록상표

V. X 시스템의 再使用性 分析

X 시스템은 a_i 組織들이 공통으로 실행하는 進級資料評價 시스템이기 때문에 그 意味(semantic)면에서 類似한 업무로 보이겠지만, X 시스템을 a_i 組織 전체의 공동 노력으로 기존의 言語(a_i 組織이 사용하는 COBOL 言語를 基準)를 사용하고 기존의 設計方法을 이용하여 開發한다면 기존의 言語로 具現하는 과정에서 再使用性이 희박한 것으로 판단되었다.[25]

본 章에서는 再使用性을 고려하여 設計 및 具現된 X 시스템을 소프트웨어 壽命週期(life cycle) 段階別로 나누어 再使用性을 分析하고자 한다.

1. 計劃段階의 再使用性

가. 再使用性 分析對象

計劃段階에서 再使用性 分析對象은 확장된 자료흐름도(expanded DFD)의 최하위 수준(primitive)의 bubble에 대한 소단위명세서(mini-specification)이다. 따라서 〈도표. 1〉과 같이 총 71개 bubble에 대한 소단위명세서를 分析對象으로 한다.

도표. 1 a_i 組織別 자료흐름도 分析表

구 분	총 계	a_1	a_2	a_3
分析對象 Bubble수	71	29	22	20

나. 再使用性 基準(criteria)

소단위명세서를 MS_{ijk}로 정의하면 다음과 같다. MS_{ijk} 여기에서

i : a_i 組織番號($1=a_1$, $2=a_2$, $3=a_3$)

j : a_i 組織의 Xa_i을 나타낸 j번째 소단위명세서 番號

k : MS_{ij}내에서 알고리즘($k=1$), 入力($k=2$), 出力($k=3$)

1) 適用 基準

a_i 組織의 MS_{ij}을 基準으로 $a_{i'}$ 組織의($i=i'$) $MS_{i'j'}$을 비교해 보았을 때에 同一한 알고리즘, 入力, 出力을 만족하는 MS_{ij}만이 再使用 가능한 것으로 한다. 이들의 관계를 표현한다면 〈그림. 3〉과 같이 $MS_{ij}RMS_{i'j'}$이다.

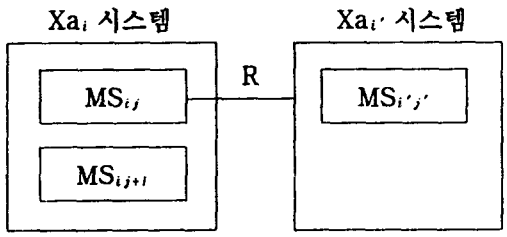

그림. 3 相異한 組織間의 소단위명세서의 同一性 관계

2) 適用 尺度(metric)

$T_i = \{MS_{ij}\}$

T_i는 a_i 組織의 Xa_i시스템에 존재하는 小單位明細書 集合이다.

$$R.MS_i = \{x | x \in T_i \text{ and } y \in T_{i'} \text{ and } x\,R\,y \text{ and } i = i'\}$$

R. MS는 a 組織의 소단위명세서 집합 T_i과 a ($i = i'$)조직의 소단위명세서 집합 $T_{i'}$ 중에서 동일성의 관계를 만족하는 소단위명세서 집합을 말한다. $|T_i|$, $|R.MS_i|$는 각각 T_i집합과 $R.MS_i$집합의 cardinality이다.

$$a_i \text{ 組織의 再使用性 尺度} = \frac{|R.MS_i|}{|T_i|}$$

a_i 組織의 再使用性 尺度는 a_i 組織의 전체 소단위명세서의 수($|T_i|$)중에서 再使用으로 대치한 소단위명세서 수($|R.MS_i|$)와의 비율로 한다.

다. 再使用性 分析 結果

計劃段階에서 알고리즘, 入力, 出力의 동일성을 基準으로 再使用性을 分析한 結果는 다음과 같다.

X 시스템을 가지고 각각 a_i 組織을 基準으로 $a_{i'}$ 組織間의 再使用性은 14-35% 정도로 낮은 편이다. 왜냐하면 기존의 言語(COBOL의 경우)는 알고리즘이 같다 하더라도 入力과 出力 資料構造가 相異하다면 이들 간의 再使用은 기대하기 어려운 실정이기 때문이다.

2. 計劃段階의 再使用性

가. 再使用性 分析對象

設計段階에서 再使用性 分析對象은 Ada 開發環境에서 設計段階의 최종산물인 Ada 프로그램 構成單位의 明細部分으로 한다. 여기서

構成單位란 패키지, 副프로그램, 일반화 프로그램 단위 등을 말하는데, 이 중 패키지는 하나 이상의 副프로그램으로 구성될 수 있다. 따라서 分析對象을 構成單位와 副프로그램별로 나누어 고려한다.

(1) 構成單位를 基準으로 明細部分을 조사해 본 結果 〈도표.2〉와 같이 48개의 明細部分을 分析對象으로 한다.

(2) 副프로그램을 基準으로 明細部分을 조사해 본 결과 〈도표.2〉와 같이 69개의 明細部分을 分析對象으로 한다.

도표.2 a_i 組織別 明細部分 수

구 분	총계	a_1	a_2	a_3
構成單位 기준	48	18	15	15
副프로그램기준	69	25	22	22

나. 再使用性 基準

構成單位別 明細部分을 CS_{if}라 정의하면 다음과 같다.

CS_{if}여기에서

i : a_i 組織番號(1=a_1, 2=a_2, 3=a_3)

f : a_i 組織의 Xa_i을 나타낸 f번째 明細部分 番號이다.

1) 適用 基準

CS_{if}중에서 아래 基準을 모두 만족하는 CS_{if}만을 再使用 가능한 것으로 한다.

(1) 構成單位 明細部分의 명칭이 同一하고, with되는 구성요소가 있으면(primitive한 구성요소가 아닌 경우) with되는 구성요소 명칭과 갯수가 同一한 경우이다. 단 여기서 命名된 명칭은 A集團 전체가 共同으로 設計함으로서 一貫性 있게 작성되어 있고, 명칭이 同一하다는 것은 그 수행 機能이 同一함을 뜻한다.

(2) 매개변수 리스트(parameter list)가 同一하고, 形式(type, subtype)이 同一하면 再使用性이 있는 것으로 基準한다. 여기서 매개변수 리스트가 同一하다는 意味는 매개변수의 명칭, mode(in, out, in out) 등이 모두 同一한 경우이고, 形式이 同一하다는 意味는 정의된 자료구조와 項目(item)이 同一한 경우다.

2) 適用 尺度

$S_i = \{CS_{if}\}$

S_i는 a_i 組織의 Xa_i시스템에 존재하는 小單位明細書 集合이다.

$$R.SP_i = \{x \mid x \in S_i \text{ and } y \in S_{i'} \text{ and } x\,Ry \text{ and } i \neq i'\}$$

$R.SP_i$는 再使用 가능한 明細部分의 集合을 意味한다. $|S_i|$, $|R.SP_i|$는 각각 S_i集合과 $R.SP_i$集合의 cardinality이다.

$$a_i\text{組織의 再使用性 尺度} = \frac{|R.SP_i|}{|S_i|}$$

副프로그램을 基準으로 할 경우도 위에서 제시한 適用 基準과 尺度를 똑같은 요령으로 適用하여 계산하기로 한다.

다. 再使用性 分析 結果

設計段階에서 再使用性을 分析한 結果는 첫

째 構成單位別 明細部分을 단위로, X 시스템을 a_i 組織을 基準으로 $a_{i'}$ 組織間의 再使用性은 44-60% 정도 이었고, 둘째 副프로그램 明細部分을 단위로, X 시스템의 a_i 組織을 基準으로 $a_{i'}$ 組織間의 再使用性은 44-50% 정도이었다. 따라서 X 시스템을 A 集團 전체가 共同開發한다면 設計段階에서 각 副프로그램 및 一般化 프로그램에 대한 設計文書 작성시 再使用性 基準에 적용되는 것은 同一한 設計文書로 볼 수 있기 때문에 새로운 文書를 작성할 필요가 없다. 또한 構成單位의 單位試驗(unit-test)을 최소화하기 때문에 設計過程의 努力을 節減할 수 있다.

3. 코딩段階의 再使用性

가. 再使用性 分析對象

코딩段階에서 再使用性 分析對象은 構成單位의 明細部分을 實行部分까지 具現한 프로그램의 원시코드(source code) 라인수이다. (pretty printer에 의해 계산된 원시코드 라인수 중에서 공란(space)과 주석(comment)부분은 제외한 라인수이다.) 分析對象은 〈도표 3〉과 같이 39개 構成單位의 원시코드 라인수이다.

도표. 3 a_i 組織別 Xa_i 시스템을 具現한 構成單位 수

구 분	총계	a_1	a_2	a_3	공통
構成單位수	39	13	11	10	5
원시코드수	195	621	598	512	220

나. 再使用性 基準(수정없이 재사용하는 경우)

원시코드 라인수를 SCL_{ih}라 정의하면 다음과 같다. (i는 組織番號, h는 프로그램 구성단위 番號이다.)

1) 適用 基準

SCL_{ih}중에서 아래 基準을 만족하는 SCL_{ih}만을 再使用 가능한 것으로 한다.

(가) 一般化 프로그램을 실례화하여 사용할 경우 一般化 프로그램 사용 횟수 만큼 원시코드 라인수가 再使用性이 있는 것으로 판단한다. 단 一般化 프로그램을 再使用하기 위하여 실례화하는데 필요한 라인수 만큼은 제외한다.

(나) 패키지, 副프로그램 등을 再使用할 경우는 사용 횟수에는 관계없이 再使用된 원시코드 라인수 만큼 再使用性이 있는 것으로 基準한다. 이들의 관계를 그림으로 표시하면

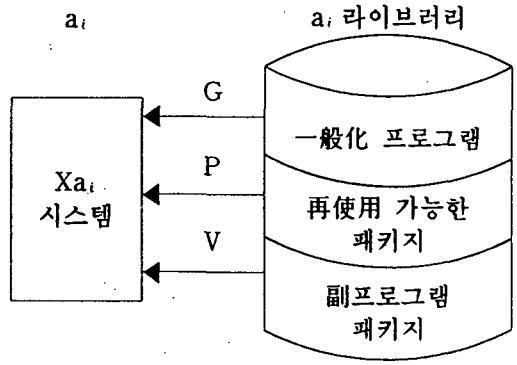

그림. 4 Xa_i 시스템의 프로그램 再使用에 대한 모형도

은 G의 경우고, (2)는 P의 경우다.

2) 適用 尺度

再使用性의 適用 尺度는 전체 SCL_{ih} 중에서 再使用性의 基準을 만족하는 SCL_{ih}의 비율로 한다.

Xa_i 시스템의 원시코드 再使用性 尺度 =

$$\frac{\sum_{q=1}^{t}(G(q) * N(q)) + \sum_{r=1}^{u}P(r) + \sum_{s=1}^{v}(V(s)-M(s))}{ULOC + \sum_{q=1}^{t}(G(q) * N(q)) + \sum_{r=1}^{u}P(r) - \sum_{q=1}^{t}L(q) + \sum_{s=1}^{v}V(s)}$$

TLOC : a_i 組織에서 Xa_i 시스템을 具現하는데 필요한 물리적인 構成單位 원시코드 라인수

q : 一般化 프로그램 단위의 종류

r : 再使用되는 패키지의 종류

G(q) : q번째 一般化 프로그램 라인수
$1 \le q \le t$

N(q) : q번째 一般化 프로그램 반복사용 빈도

P(r) : r번째의 再使用된 패키지의 라인수
$1 \le r \le u$

L(q) : i번째 一般化 프로그램을 실례화 하는데 필요한 라인수 合

다. 適用 基準(일부 수정하여 재사용하는 경우)

일부 수정작업을 통하여 再使用性의 정도를 측정하기 위한 適用基準과 尺度는 다음과 같이 정의한다.

1) 適用 基準

전자의 경우에서 추가로 다음과 같은 基準이 추가된다.

(1), (2)는 전자의 경우를 適用한다.

(3) a_i 組織에서 a_i' 組織의 構成單位의 원시코드를 10라인 이내로 變更(삭제, 추가)하여 再使用할 수 있으면 再使用性이 있는 것으로 基準한다. 즉 그림으로 표시하면 〈그림. 4〉에서 V의 경우다.

2) 適用 尺度

Xa_i 시스템의 원시코드 再使用性 尺度 =

$$\frac{\sum_{q=1}^{t}(G(q) * N(q)) + \sum_{r=1}^{u}P(r)}{TLOC + \sum_{q=1}^{t}(G(q) * N(q)) + \sum_{r=1}^{u}P(r) - \sum_{q=1}^{t}L(q)}$$

ULOC : a_i 組織에서 Xa_i 시스템을 具現하는데 唯一(unique)하게 필요한 물리적인 構成單位의 원시코드 라인수.

q : 一般化 프로그램 단위의 종류

r : 再使用되는 패키지의 종류

s : 一部分 수정하여 再使用되는 패키지의 종류

G(q) : q번째 一般化 프로그램 라인수
$1 \le q \le t$

N(q) : q번째 一般化 프로그램 반복사용 빈도

P(r) : r번째의 再使用된 패키지의 라인수
$1 \le r \le u$

L(w) : i번째 一般化 프로그램을 실례화

하는데 필요한 라인수 합

V(s) : k번째 프로그램을 일부 수정하여 재사용 가능한 프로그램 라인수 $1 \leq s \leq w$

M(s) : k번째 프로그램을 再使用할 수 있도록 수정한 라인수

라. 再使用性 分析 結果

코딩段階에서 再使用性을 分析한 結果는 첫째 수정없이 再使用된 프로그램의 원시코드 라인수는 a_i 組織을 基準으로 a_i' 組織間의 再使用性이 37-38% 정도이었고, 둘째 일부분 수정하여 再使用된 원시코드 라인수를 포함한 X시스템의 再使用性은 a_i 組織을 基準으로 a_i' 組織間의 再使用性이 45-63% 정도이었다. 따라서 X시스템을 A集團 전체가 再使用性을 고려하여 Ada 言語 開發環境에서 共同開發한다면 높은 再使用性으로 인하여 生産性 및 品質의 向上을 기할 수 있다.

V. 結 論

본 論文의 結果는 하나의 프로젝트를 대상으로 分析한 結果이고 전체 시스템에 대한 再使用性의 分析은 아님이 분명하다. 그러나 분석대상으로 삼았던 X 시스템을 組織의 個別努力으로 開發할 때 보다는 統合努力으로 再使用性을 設計 基準으로 適用하여 開發하였을 때에는 높은 再使用性의 정도를 기대할 수가 있었다.

따라서 앞으로 A集團의 경우 類似한 업무는 統合努力으로 開發을 시도하는 것이 소프트웨어의 品質 보장과 生産性 向上으로 電算資源의 효율적 활용을 기대할 수가 있다.

참고문헌

〔1〕 E. Horowitz and B. Munson, "An Expansive View of Reusable Software", IEEE Trans. on Software Eng., Vol. SE-10, No. 5, Sept., 1984.

〔2〕 G. Booch, Software Engineering with Ada, 2nd ed. The Benjamin/Cummings Publishing Company Inc., 1986.

〔3〕 B. W. Boehm, "Software and its impact : A Quantitative Assessement", Datamation, May 1973, p. 13.

〔4〕 R. G. Lanergan, C. A. Grasso, "Software Eng. with Reusable Design and Code", IEEE Trans. on Software Eng., Vol. SE-10, No. 5, Sept. 1984, pp. 498-501.

〔5〕 W. Wong, "A Management Overview of Software Reuse", NBS Special Publication 500-142, U. S., Department of Commerce, Sept. 1986.

〔6〕 윤창섭, "소프트웨어 재사용과 설계에 관한 고찰", 한국군사운영분석학회지, Vol. 15, No. 1, June 1989, pp. 1-13.

〔7〕 G. Booch, "Object Oriented Development", Tutorial : Object Oriented Computing, Vol. 2, Implementation, The Computer Society of IEEE, 1987. pp. 5-15.

〔8〕 B. J. Cox, Object-Oriented Programming An Evolutionary Approach, Addison-Wesley Publishing Company, 1986, pp. 13-28.

〔9〕 R. St. Dennis, "Reuasble Characteristics of Reusable Ada Software", pp. vi. 2-41—vi. 2-50.

〔10〕 R. St. Dennis, "Measurable Ada Software Guidlines", Tutorial : Software Resue : Emerging Technology, The Computer Society of IEEE, 1988, pp. 62-67.

〔11〕 G. A. Pascoe, "Elements of Object-Oriented Programming", Tutorial : Object Oriented computing, Vol. 1 : concepts, The Computer Society of IEEE, 1987, pp. 15-20.

〔12〕 T. De Marco, Structured Analysis and System Specification, YOURDON Inc., 1979.

〔13〕 R. S. Pressman, Software Eng. A Practitioner's approach, 2nd ed., McGraw-Hill, 1987.

〔14〕 E D. Seidewitz, M. Stark, "Towards A General Object-Oriented Software Development Methodology", Tutorial : Object Oriented Computing, Vol. 2, Implementation, The Computer Society of IEEE, 1987, pp. 16-29.

〔15〕 P. Jalote, "Functional Refinement and Nested Object for Object-Oriented Design", IEEE Trans. on Software Eng. Vol. 15, No. 3, March 1989, pp. 264-270.

〔16〕 김규석, 윤창섭, 자료흐름분석 모델을 이용한 대상의 식별과 대상중심 설계규칙에 관한 연구, 국방대학원, 1989.

〔17〕 U. S., D. O. D. Reference Manual for the Ada programming Language, July 1980.

〔18〕 D. N. Card, V. E. Church and W. W. Agresti, "An Empirical Study of Software Design Practices", IEEE Trans. on Software Eng., Vol. SE-12, No. 2, Feb. 1986.

〔19〕 C. Ausnit, C. Braun, S. Eanes, J. Goodenough, and R. Simpson, "Ada Reusability Guideline", ESD-TR-85-142, SoftTech, inc., April 1985.

〔20〕 육군본부, 육군규정 113, 진급관리규정(군사비밀구분 : 평문), 1984.

〔21〕 해군본부, 해군규정 제 3 권, 진급규정(군사비밀구분 : 평문), 1987, pp. 427-478.

〔22〕 공군본부, 공군규정 36-7, 장교인사 장교진급(군사비밀구분 : 평문), 1983, pp. 1-32.

〔23〕 박형춘, 윤창섭, Ada 중심의 소프트웨어 재사용성을 위한 설계에 관한 연구, 국방대학원 1989.

〔24〕 이광건, 윤창섭, Ada 언어의 재사용성을 이용한 소프트웨어 구현에 관한 연구, 국방대학원 1989.

〔25〕 강석균, 윤창섭, Ada 언어를 이용한 재사용성 분석에 관한 연구, 국방대학원, 1989.