韓國 컴퓨터情報學會 論文誌
第9卷 第3號, 2004. 9.

2004-9-3-1-13

수직적 재사용을 위한 방법론 연구

방 정 원*

The Study of Software Analysis Process for Vertical Reuse

Bang, Jung Won*

요 약

소프트웨어의 재사용은 생산성 향상, 소프트웨어의 신뢰성 향상, 소프트웨어 개발기간 단축, 개발비용 절감, 시스템 관련 정보 공유 등의 장점을 지닌다. 수직적 재사용은 하나의 영역 또는 응용 분야에서만 재사용함으로서, 높은 이해성으로 인해 재상용의 신뢰성을 높일 수 있는 방법이다. 소프트웨어 시스템을 개발할 때 사용되는 정보들을 재사용을 목적으로 분류하여, 결과물들을 추상화하는 상향식 방법을 이용하여 단순 코드의 재사용에서 벗어나 비형식적인 자료들을 포함해 모든 정보를 사용 가능하게 할 수 있다

Abstract

Software Reuse have the advantages of increasing productivity and software reliability, reducing the period for software development and cost, sharing the information which is related to software. Vertical Reuse is the method which reuse is restricted to a specific area and can increase the reliability of software reuse due to high understanding of related area. Bottom-up approach of Vertical Reuse classifies the related information and abstracts the results. It allows the reuse of not only pure source code but also informal documents.

▸ Keyword : 소프트웨어 재사용, 수직적 재사용, 재사용성

• 제1저자 : 방정원
• 접수일 : 2004.08.27, 심사완료일 : 2004.09.14
* 청강문화 산업대학 컴퓨터소프트웨어과 교수

I. 서 론

소프트웨어의 재사용은 프로그래머 생산성 향상과 안정적 시스템 구축을 위한 방안으로 주목받아 왔다. 소프트웨어의 재사용이란 단순히 라이브러리나 소스코드를 재사용하는 개념을 넘어 소프트웨어 시스템을 구축하는 전 과정에 거쳐 일어난다.

소프트웨어의 재사용은 생산성향상, 소프트웨어의 신뢰성 향상, 소프트웨어 개발기간 단축, 개발비용 절감, 시스템 관련 정보 공유등의 장점 을 지닌다.[1]

재사용을 재사용 방법의 관점에서 보면 순수한 의미의 일반적 재사용과 수정을 통한 재사용으로 나누어 볼 수 있다. 수정 없이 재사용 하는 경우의 핵심개념은 정보 은닉이다. 정보은닉이 잘 되어 있는 소프트웨어는 추상화가 잘 이루어져 있는데, 이는 소프트웨어 부품이 매우 복잡하여 그 구현의 세부사항을 알기 어려워도, 그 소프트웨어 부품의 기능들을 쉽게 재사용할 수 있게 한다.

수정을 통한 재사용의 경우는 수정성이 가장 중요한 요소가 된다. 수정을 위한 비용이 너무 높다면, 수정을 통해 재사용하는 것보다는 새로 작성하는 것이 낳기 때문이다. 그러나 손쉽게 수정이 가능하다면, 적은 비용과 시간으로 기존의 부품을 재사용할 수 있다. 수정성은 소프트웨어 시스템을 수정하는데 있어서의 용이성 또는 난이도를 말하며, 수정으로 인해 신뢰성이 저하될 수 있는 가능성을 뜻하는 것으로 정의된다.[2]

따라서, 수정성을 높이기 위해서는 높은 이해성과 수정으로 인해 발생하는 파급 효과의 최소화가 필요하다. 소프트웨어 수정을 위해서 가장 먼저 수행되어야 할 것이 소프트웨어에 대한 이해이고, 이를 위해 많은 시간이 소요되게 되므로 높은 이해성은 용이한 수정과 재사용을 위해 필수적이다.

이러한 소프트웨어에 대한 이해를 높이기 위해서는 소프트웨어의 여러 품질들 외에 적용업무 영역에 대한 지식이 필수적이다. 특정 영역에서의 재사용만을 고려할 때는 정보은닉이나 수정성 보다는 그 영역에의 적합성이 더 크게 작용하며, 이는 그만큼 특수성이 커진다고 할 수 있다.

이 논문에서는 재사용의 여러 가지 관점들을 살펴보고, 하나의 영역 또는 응용분야에서만 재사용함으로서, 높은 이해성으로 인해 재상용의 신뢰성을 높일 수 있는 방법인 수직적 재사용에 대하여, 재사용의 수정성 측면에 초점을 두고 살펴보고자 한다.

II. 재사용의 여러 가지 관점

소프트웨어의 재사용은 무엇을 재사용 하는가에 따라 다음과 같이 나누어 볼 수 있다[3].

재사용하는 주체에 따라 아이디어/개념 재사용, 부품 재사용, 절차/기술 재사용으로 나누어 볼 수 있다.

아이디어의 재사용은 어떤 부류의 문제들에 대한 일반적 해결방법과 같은 개념들을 재사용하는 것을 말한다. 예를 들면, 정렬알고리즘의 재사용과 같은 것을 들 수 있다. 부품 재사용은 자료구조와 알고리즘을 추상화시켜 표현하는 컨테이너로서의 소프트웨어 부품의 재사용을 말하는데, 응용 시스템, 서브시스템, 모듈 또는 객체, 함수 또는 서브프로시저의 재사용을 말한다. 객체 지향 기술은 부품 재사용의 가장 전형적인 기술이다. 절차/기술 재사용은 소프트웨어를 개발하는 제반과정을 재사용함으로서 기술과 축적된 관련지식을 재사용하는 것을 말한다.

재사용의 범위에 따라 나누어 보면 수직적 재사용, 수평적 재사용으로 나누어진다. 수평적 재사용은 일반적으로 사용될 수 있는 부품을 이용해서 여러 가지의 영역의 응용에서 재사용 하는 것을 말한다. 수직적 재사용은 하나의 영역 또는 응용 분야에서만 재사용함으로서, 높은 이해성으로 인해 재상용의 신뢰성을 높일 수 있는 방법으로 최근에는 단순 코드의 재사용에서 벗어나 디자인과 관련된 기록, 테스트 계획서같은 비형식적인 자료들을 포함해 모든 정보를 사용 가능하게 하는 것이다. 어떤 부품을 재사용할지를 결정하려면, 그 부품을 왜 그렇게 디자인 했는지를 알아야 하는데, 일반적으로 소스코드에는 이러한 내용이 없기 때문이다. 따라서 이러한 정보들을 사용 가능하게 함으로서 재사용을 보다 효과적으로 할 수 있게 된다.

기술적 관점에서는 구성적 재사용과 생성적 재사용으로 나누어 볼 수 있다. 구성적 재사용에서는 각각의 재사용 가능 부품들이 원자적으로 재사용되고, 재사용 도중에 수정되지 않는다. 따라서 각각의 부품들은 재사용 이후에도 그 형

태와 정체성을 유지 한다. 생성적 재사용은 재사용 부품들이 재사용 되어 지는 프로그램 안으로 흡수되는 형태가 된다[4]. 대표적인 예로서 컴파일러를 들 수 있는데, 이러한 생성적 재사용의 한계는 재사용의 범위가 주어진 응용 영역 내의 한정된 분야의 문제들로 제한된다.

재사용의 방법에 따라서는 수정 없는 재사용과 수정을 통한 재사용으로 나누어 볼 수 있다. 수정 없는 재사용은 소프트웨어 부품을 있는 그대로 재사용하는 것으로, 수정을 통한 재사용 보다 우수한 소프트웨어 품질과 신뢰성을 갖는다. 수정 없는 재사용을 위한 부품을 만드는 핵심은 정보 은닉이며, 이는 객체 지향 개념의 핵심이다. 이러한 재사용은 개발에 필요한 노력과 시간을 크게 줄일 수 있고, 따라서 생산성을 크게 높일 수 있다. 수정을 통한 재사용은 소프트웨어 부품을 목적에 맞게 수정하여 재사용하는 것으로, 프로그램이 진화함에 따라 이러한 형태의 재사용이 점점 많아지게 된다. 그러나 수정이 용이 하지 않은 경우 생산성이 떨어지고 신뢰도가 떨어지므로, 해당 소프트웨어에 대한 이해가 필수적이다.

III. 수직적 재사용

특정 영역에 관한 지식은 여러 개의 시스템이 개발된 후 경험이 축적된 후 비 체계적인 방법으로 구체화 되어 왔다. 예를 들어, 보고서 작성 프로그램의 경우 여러 개의 시스템이 개발된 후 이러한 기능이 여러 업무에서 필요함에 따라 일반적인 기능들을 추출해 내어 자동화 프로그램을 생성하였다. 특정 분야에서 유사한 시스템 간에 객체와 동작등을 찾아내어 재사용을 체계화 하는 것은 수직적 재사용의 효율성을 높여 준다.

3.1 상향식 분석 방법

소프트웨어 시스템을 개발할 때 사용되는 정보들을 재사용을 목적으로 분류하여, 결과물들을 추상화하는 방법이다[5]. 기존 시스템의 소스 코드, 디자인 스펙, 디자인 결정 자료, 특정 분야 관련 지식, 시스템 요구사항들을 모아 도서관의 서지 분류 기법을 기본으로 분류하여 추상화 한다. 이러한 것들은 영역 모델이나 표준사항 또는 재사용 가능한 컴포넌트로 구성되어진다.

전반부에서는 특정 분야와 영역의 경계에 관한 정의, 정보 수집, 접근 방법에 대한 결정들로 이루어진다. 그 다음 단계에서는 객체 및 동작을 인식해서 추상화하고 분류하게 되는데, 이 때 영역 모델, 프레임, 분류 언어들을 만들어 낸다. 후반부에서는 위에서 만들어진 결과물을 가지고 재사용 가능한 컴포넌트들을 뽑아 체계화하고, 모듈화하고, 표준화 한다. 이 컴포넌트들과 함께 영역 표준 사항들을 만들어 내는데, 이것은 재사용 지침으로 컴포넌트 리스트, 통합 방법 등에 대한 내용을 담고 있다.

이들 단계간의 정보 전달은 특정 영역에 커스터마이즈된 요구사항 자료, 영역 분류, 영역 프레임을 통해 이루어진다. 영역 모델과 영역 언어가 되는 경우도 있는데, 반드시 있어야만 하는 것은 아니다. 모델링이 어려운 영역도 있고, 영역 언어를 만들어 내는 것은 무척 어려운 일이어서 경우에 따라서는 불가능하기도 하다.

이렇게 분류된 영역은 각 객체나 기능들을 노드로 하고 이들 간의 서로 다른 관계들을 선분으로 하는 계층구조를 갖는다. 간단한 영역의 경우는 모듈 인터페이스 언어로 나타내거나 트리 형태로 나타내기도 한다. 복잡한 영역의 경우는 세맨틱 넷이 되기도 한다.

이러한 방법은 기 개발된 시스템을 전재로 재사용을 주 목적으로 사용된다.

3.2 하향식 분석 방법

소프트웨어 개발자가 소프트웨어를 재사용 하는데 있어서 가장 어려운 점은, 재사용 가능한 소프트웨어의 수가 적고, 있다 하더라도 서로 다른 여러 개의 소프트웨어를 통합하기가 어렵다는 것이다. 하향식 접근 방법은 소프트웨어의 프레임뿐만 아니라 디자인에 따라 어느 정도의 구현까지도 제공하며 사용자가 원하는 사항들을 시스템과 구현 제약 사항을 근간으로 하는 소프트웨어의 요구사항으로 대응시키는 전반적인 프로세스다[6].

제일먼저 이루어져야 하는 일은 영역을 결정하는 것이다. 사용자가 무엇을 원하는지에 초점을 맞추어 무엇을 완성할 지를 결정한다. 그 다음에는 문제 영역에 초점을 두고, 영역의 개념과 요구사항을 정의하고, 다듬는 과정을 반복한다. 요구 사항 분석이 완료되면, 영역의 디자인과 구현 제약 사항을 정의하고 다듬는다. 이 때에는 문제의 영역보다는 해결 방안에 더 초점을 맞추어 분석을 진행한다. 이렇게 나온 세부 사항들을 가지고 영역의 구조와 모델을 개발한다. 상위 수준의 디자인 단계와 유사하며 주로 모듈과 모델의 접

속부와 의미를 정의 한다. 마지막으로 재사용 가능한 컴포넌트들을 만들어내고, 재사용 가능한 코드나 기록 등을 만들어내고 수집한다.

이러한 방법은 기존의 시스템이 있는 경우 이를 사용 할 수는 있으나 재사용 가능한 자원이나 정보 지식들을 찾아내는 주요한 수단으로 사용하기 보다는 객관화된 요구사항으로부터 이끌어 낸 영역의 구조를 검증하는데 사용한다. 즉, 각 단계마다 끝에 검증과정을 두고 있는데, 이 때 검증하는 자료로 기존의 시스템이 사용된다.

Ⅳ. 소프트웨어의 재사용성

기존의 소프트웨어 시스템은 많은 부품들로 이루어져 있고, 이중에는 재사용성이 높은 부품도 있고, 그렇지 않은 것도 있다. 재사용성이 높은 부품을 추출해 내기 위해서는 그에 적합한 기준을 알아야 한다. 이러한 기준에 따라 재사용성이 높은 부품들을 추출해내고, 또는 기존의 부품들을 재사용성이 높아지도록 가공해서 그 기능과 의미에 따라 분류하고 문서화 하면, 우수한 소프트웨어 재사용 라이브러리를 만들 수 있다[8]. 재사용성을 파악하면, 재사용 부품의 저장소나 라이브러리 구축과 이용에 직접적으로 이용될 수 있고, 또 소프트웨어의 생명주기의 상위 단계로 피드백 되어 재사용 중심의 소프트웨어 개발을 가능하게 한다.

수정성은 소프트웨어 시스템을 수정하는데 있어서의 용이성 또는 난이도를 말하며, 수정으로 인해 신뢰성이 저하될 수 있는 가능성을 뜻하는 것으로 정의된다. 수정성을 높이기 위해서는 높은 이해성과 수정으로 인해 발생하는 파급효과의 최소화가 필요하다.

재사용성을 염두에 두고 수정성이 높은 정보들을 발견하고, 이를 사용해서 소프트웨어 시스템을 구성한다면, 그만큼 그 시스템의 이해성, 유지보수성을 높일 수 있고, 개발에 필요한 시간과 노력과 비용의 절감등 재사용으로 인해 발생하는 여러 가지 이점들을 쉽게 얻을 수 있다. 즉, 재사용의 주요 목적인 요구 사항, 제반 명세서, 지식, 정보에 대한 이해와 설명에 필요한 노력을 절감시킬 수 있다. 그 외에도 재공학 과정에서 재사용성이 높은 부품의 사용을 유도할 수 있으므로, 더 우수한 재 공학을 가능케 한다.

수정성이 나쁘더라도 정보 은닉이 잘 되어 있다면, 특정한 요구사항에 잘 부합하는 경우, 수정 없이 재사용하기가 매우 좋다. 정보은닉은 가능한 한 많은 정보 즉 논리 제어와 자료 구조 등의 세부사항을 다른 외부의 부품들이 알지 못하게 하는 것을 의미한다. 정보 은닉을 높이기 위해서는 부품의 자기 완결성과 높은 응집도, 되도록 적은 매개 변수가 필요하다. 자기 완결성이 필요한 이유는, 그 부품에서 사용하는 자료 구조와 제어는 외부 부품과의 의존성이 적을수록 정보 은닉에 유리하기 때문이다. 그리고 응집도는 정보 은닉과 밀접한 관계가 있어서, 정보 은닉이 잘 되어 있을수록 응집도가 높아지게 된다. 왜냐하면, 응집도가 높다는 것은 그 부품의 각각의 부품들 모두가 매우 가까운 논리적 관계를 가지고 있다는 것을 의미하는데, 정보 은닉이 잘 되어 있을수록 외부에서 알 필요가 있는 정보가 적고, 그 만큼 부품내부의 각 부분들이 밀접하게 잘 연결되어 있게 되므로 응집도가 높아지게 되는 것이다. 물론 응집도가 높다고 해서 항상 정보 은닉이 잘되어 있다고 볼 수는 없다. 응집도의 정의상 전역 변수를 사용함으로써 정보 은닉의 정도가 낮은 경우에도 응집도는 높을 수 있기 때문이다[7].

소프트웨어의 여러 부품들은 구체적 필요성이 제기될 때만, 재사용 방법을 결정하고, 재사용될 수 있다. 여기서 명확히 해야 하는 것은 그 범위 또는 영역에 관한 것이다. 수직적 즉, 특정영역에서의 재사용에서는 소프트웨어의 여러 품질들 외에 영역에 대한 지식이 매우 중요하다. 즉 특정영역에서의 재사용만을 고려할 때는, 정보은닉이나 수정성보다는 그 영역에의 적합성이 더 크게 작용하며, 이는 그만큼 특수성이 커진다고 할 수 있다. 따라서 재사용성이 더 높은 아이디어/설계 등의 정보의 재사용에 있어서는 영역에의 적합성이 재사용을 높이는 가장 중요한 요소가 되며 이와 더불어, 수정성, 정보 은닉성 등에 대하여도 살펴보아야 한다.

Ⅴ. 재사용성을 고려한 분석 방법

상향식 방법은 영역분석의 결과물로 나오는 영역 분류와 영역 표준 사항을 통해, 필요한 정보의 추출을 용이하게 해 재사용할 수 있는 컴포넌트와 필요한 정보를 찾아내는 것이 용이하다. 따라서 이미 많은 시스템이 구현되어 있는 영역의 경우 재사용을 주 목적으로 하는 경우에 더 적합하다고

하겠다. 또한 이는 특정 시스템에 중점을 둔다는 점을 제외하면 객체지향 분석방법과 비슷하며, 프로세스가 반복적으로 행해지므로 쉽게 접근할 수 있다[9].

하향식 방법은 재사용 가능한 자원이나 정보들을 찾아내는 주요한 수단으로 사용 하는 것이 아니라 객관화 된 요구사항으로부터 이끌어 낸 영역 구조를 검증하는 데 용이하므로 새로운 제품개발인 경우 해당 영역에서 그 제품의 가능성을 찾아내는 데 유리하다. 새로 가능성 높은 고객들로부터 인터뷰를 하거나, 관련된 제품군의 요구사항이나 제약사항들을 영역 전문가로부터 얻는 과정에서 제품의 가능성을 찾아낼 수 있다[10]. 결론적으로, 수직적 재사용을 높일 수 있는 방법으로는 상향식 방법이 적당하다. 이러한 방법은 기존이 시스템이 이미 많이 존재하여 영역 분석의 요구가 무르익은 분야에서 주로 사용할 수 있는 방법이다. 그러나 이러한 방법은 결정의 가장 주요한 요인인 디자인 결정에 대한 정보들이 회복되지 않고, 기존 시스템의 오류에 대한 수정이나 기능 확장 등으로 인한 군더더기가 포함된다는 단점을 지닌다. 재사용성을 높이기 위해서는 이러한 상향식 방법을 기조로 하되, 무엇보다 먼저 영역에의 적합성을 판단하여야 한다.

분류과정에서 정보은닉에 초점을 두어 외부와의 상관관계가 되도록 적어지도록 분류하고 문서화 하여야 한다. 정보 은닉이 잘 되어 있을수록 외부에서 알 필요가 있는 정보가 적고, 그 만큼 부품내부의 각 부분들이 밀접하게 잘 연결되어 있게 되는데, 이는 그 부품의 각각의 부품들 모두가 매우 가까운 논리적 관계를 가지고 있다는 것을 의미한다.

분류된 각 부품들은 수정성을 높이기 위해 높은 이해성과 수정으로 인해 발생하는 파급 효과가 최소화되도록 구성하여야 한다. 하부단위의 소스 코드뿐만 아니라 소프트웨어 개발과정에서 발생하는 요구사항 명세서나, 시스템 명세서, 디자인 결정과정, 디자인 명세서등의 재사용등의 모든 정보의 재사용을 염두에 두고 체계화된 표현 방법을 사용하는 것이 중요하다.

참고문헌

[1] C. Maclure, The Three Rs of Software Automation, Prentice Hall, Englewood Cliff.NJ. 1992

[2] A. Macro, Software engineering concept and management, Prentice Hall, Englewood Cliff.NJ. 1990

[3] R. Prieto-Diaz, "Status Reports : Software Reusability", IEEE Software, pp. 61-66, May 1993

[4] S.Jarzabek, "From reuse library experiences to application generation architecture

[5] Ruben Prieto-Diaz "Domain Analysis : An Introduction", ACM SIGSOFT Software Engineering Notes vol 15 no 2, pp. 47-54, Apr 1990

[6] Will Tracz "A Domain-specific Software Architecture Engineering Process Outline" ACM SIGSOFT Software Engineering Notes vol 18 no 2, pp. 40-49, Apr 1993

[7] 임 근, "이벤트 추상화를 통한 정보관계 표현", 한국컴퓨터 정보학회 논문지 vol 7 no 4, pp. 1-7, Dec 2002

[8] Scott R. Tilley, Hausi A. Muller, Mehmet A. Organ, "Documenting Software System with Views", Proceedings of 10th International Conference on Systems Documentation, pp.211-219, 1998

[9] Wim De Pauw, "Visualizing the Behavior of Object-Oriented Systems", OOPSLA 99, pp. 326-337, 1999

[10] 임 근, 권영만, "객체모델에 대한 형식명세로의 변환 방법", 한국 컴퓨터 정보학회 논문지 vol 8 no 4, pp. 21-27, Dec 2003

저 자 소 개

방 정 원

1997년 ~ 현재

청강문화 산업대학

컴퓨터소프트웨어과 교수