□특집□

소프트웨어 역공학(Software Reverse Engineering)

김 태 희† 강 문 설††

◈ 목 차 ◈

1. 서 론

소프트웨어 유지보수는 소프트웨어 개발 생명주기(SDLC)에서 필요한 노력과 비용 중 가장 많은 부분을 차지하고 있으며, 현실적으로 많은 문제점과 이 문제점들의 해결을 위한 노력과 비용이 표면에 노출되지 못하고 있다. 현재 사용하고 있는 소프트웨어의 유지보수 비용은 소프트웨어 개발비용과 노력의 70% 이상을 차지하며, 이 비율은 더 많은 소프트웨어가 생산됨에 따라 소프트웨어 유지보수에 사용되는 노력과 자원의 양이 증가하는 심각한 현상이 유발되고 있다. 궁극적으로 일부 소프트웨어 개발 조직은 기존의 오래된 소프트웨어를 유지보수하는 데 사용 가능한 모든 자원을 소비해야 하기 때문에 앞으로는 새로운 소프트웨어를 개발하지 않는 "유지보수 전담(maintenance-bound)" 소프트웨어 개발 조직이 있을 수 있다.

소프트웨어 유지보수의 기본적인 문제점은 기존의 소프트웨어를 수정하고자 할 때에 전체 시스템에 예기치 못한 파급효과가 발생된다는 것이다. 이러한 파급효과를 통제하기 위한 관리적, 기술적인 노력이 필요하다. 이러한 노력은 단순히 소프트웨어 개발 생명주기의 유지보수 단계에만 적용되는 것이 아니라 시스템이 정의되고 설계되는 초기 단계부터 소프트웨어 유지보수성을 고려하여 적용되어야 한다.

일반적으로 소프트웨어 개발자들이 언제나 과거에 비해 향상된 품질의 소프트웨어를 개발한다고 아무도 장담하지 못할 뿐만 아니라 기존 소프트웨어와의 호환성이 100% 보장되지 않고서야 이미 구축된 데이터 및 사용자 교육 등에 미치는 영향 때문에 쉽게 소프트웨어 교체 결정을 내릴 수도 없다. 또한 기존의 소프트웨어에 대한 불만족도와 미련이 어느 정도이던 간에 새로운 소프트웨어로 변환하기 어려운 것도 기술적 현실이다.

따라서 기존의 소프트웨어를 수명연장이라는 목적을 위해 가능한 한 효과적으로 사용할 수밖에 없다. 그래서 등장한 개념이 소프트웨어 재공학(software re-engineering)이다. 소프트웨어 재공학의 개념 체계를 이루는 기술로 재구조화(restructuring)가 있고, 최근에는 역공학(reverse engineering)의 필요성도 대두되고 있다.

본 고에서는 소프트웨어 유지보수의 생산성을

† 정 회 원 : 전남대학교 전산학과 박사과정
†† 종신회원 : 광주대학교 전자계산학과 교수

향상시키고, 재사용을 지원하는 소프트웨어 역공학의 배경, 목표, 장단점, 역공학의 과정 및 도구, 역공학을 위한 적합한 시스템의 선택 및 역공학 도구 선택시 고려사항, 그리고 소프트웨어 개발의 자동화를 위한 3Rs에 대하여 개괄적으로 고찰한다.

2. 소프트웨어 역공학

2.1 역공학이란

컴퓨터 기반 소프트웨어 공학(CASE) 환경의 유용성이 널리 소개되면서 많은 조직들에서 시스템을 개발하는 방법의 재정의가 필요하게 되었다. 진정한 의미의 CASE 환경은 기존 시스템의 유지보수와 개선에 관련된 많은 문제들에도 적용되어야 한다. 역공학(reverse engineering) 기법은 소프트웨어 시스템의 개선 차원에서 사용되고 있으며, 소프트웨어 유지보수와 소프트웨어 개발이 만나는 시점에 역공학이 있다고 할 수 있다.

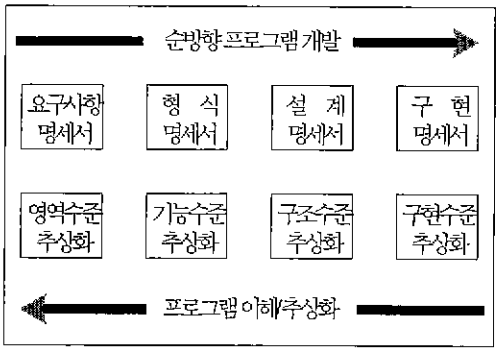
"역공학"은 원래 하드웨어 분야에서 완제품으로부터 설계 사양(design specification)을 추출하는 하드웨어 분석에서 출발하였다. 이렇게 하드웨어를 분석하는 목적은 제품의 질적 개선과 경쟁 제품을 분석하려는 것이다. M.G. Rekoff는 역공학을 "복잡한 하드웨어 시스템의 견본(specimens)을 자세히 분석하여 일련의 명세서를 개발하는 과정"이라고 정의하였다. 이러한 과정은 원래 하드웨어 시스템을 만들려고 하는 개발자보다는 유사한 제품을 만들려고 하는 개발자들이 널리 사용한다. 또한 이러한 개념을 소프트웨어 시스템에 적용함으로써 소프트웨어 시스템과 그 구조를 이해하는데 많은 도움이 된다. 그러나 하드웨어 역공학의 목적은 시스템을 복제하는 것이지만 소프트웨어 역공학의 목적은 시스템의 개선과 유지보수를 위한 설계 사양서의 정확한 이해를 지원하는 것이다.

소프트웨어 유지보수란 오류 수정, 속성이나 성능 개선, 또는 변경된 환경에 적응시키기 위해 인도된 제품을 변경하는 것이라 정의할 수 있다. 따라서 시스템 유지보수 담당자는 설계자가 아니지만 시스템을 조사하고 이해하려고 노력한다. 이러한 의미에서 역공학은 시스템을 이해하여 적절히 변경하는 소프트웨어 유지보수 과정의 일부라고 할 수 있다. 넓은 의미에서 재구조화(restructuring)도 역공학과 함께 소프트웨어 유지보수의 범주에 포함된다. 그러나 이 세 가지 과정의 각각은 새로운 시스템을 만들고 점진적으로 개발하는 범주에 포함된다.

순공학(forward engineering)과 역공학의 개념을 설명하기 위하여 생명주기 모형의 실체, 대상 시스템의 존재, 그리고 추상화 수준의 식별 등 세 개의 종속된 개념을 정확히 구분해야 한다. 생명주기 모형은 소프트웨어 개발 과정을 위해 존재해야 하고, 그 모형이 폭포수 모형이나 나선형 모형 또는 방향성 그래프로 표현할 수 있는 다른 유형으로 표현할 수 있어야 한다. 생명주기 모형은 각 단계에서 반복이나 순환(recursive)이 있을 수 있고, 방향성 그래프로 표현된 모형에서는 전진하는 방향의 작업과 후진하는 방향의 작업이 있을 수 있다.

대상 시스템은 단순한 프로그램, 부분적인 코드, 또는 대화형 프로그램, 작업 제어 명령, 단일 인터페이스, 그리고 데이터 파일 등의 복잡한 집합일 수 있다. 순공학에서 대상 시스템은 개발 과정의 결과이고, 역공학에서는 대상 시스템이 존재해야 작업이 시작된다.

생명주기 모형의 전반부에서는 일반적이고 구현-독립적인 개념을 다루고, 후반부에서는 구체적이고 구현 중심적인 개념을 다루게 된다. 생명주기의 순방향 진전은 더 구체적인 것으로 전환한다는 것을 의미하며 추상화 수준이 낮아지는 것이다. 결국 초기 단계에서는 정보를 고수준의 추상화로 표현하고 후반부에서는 저수준의 추상화로 표현한다. <그림 1>은 4 가지의 추상화 범주들이 생명주기의 다른 단계에서 사용되는 정보와 어떻게 대응되는 가를 나타내고 있다.

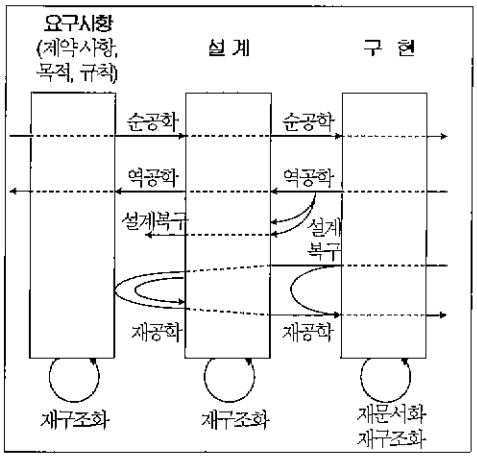


<그림 1> 순방향 프로그램 개발과 역방향 프로그램 추상화

역공학은 대상 시스템을 분석하여 시스템의 구성요소와 그 관계를 파악하고, 시스템을 다르게 표현하거나 고수준의 추상화를 추출하려는 과정이다. 역공학은 일반적으로 설계 사양서를 추출하거나 구현-독립적인 추상화된 표현을 만드는 것이다. 따라서 역공학은 대상 시스템을 변경시키거나 새로운 시스템으로 개선하는 변경이 아닌 분석 작업이다.

2.2 역공학과 관련된 정의

<그림 2>에서 처럼 분명하게 다른 추상화 수준



<그림 2> 역공학과 추상화 수준

을 갖는 생명주기의 세 단계를 사용하여 역공학과 관련된 용어들을 다음과 같이 정의한다.

■ 순공학(forward engineering)

순공학은 논리적이고 구현에 의존하지 않는 고수준의 추상화 개념으로부터 구현에 의존하는 저수준의 물리적 표현으로 진행하는 전통적인 소프트웨어 개발 과정이다. 순공학은 요구사항을 분석하여 설계하고 구현하는 일련의 과정으로써 고수준의 추상적 개념을 물리적으로 구현하는 작업이다.

■ 역공학(reverse engineering)

역공학은 시스템의 구성요소와 그들의 관계를 식별하고, 고수준의 추상화에서 시스템의 표현을 생성하기 위하여 대상 시스템을 분석하는 과정이다. 역공학은 종종 그 대상으로 기존 시스템의 기능은 포함시키지만 요구사항은 포함시키지 않는다. 역공학은 추상화의 어떤 수준이나 생명주기의 어떤 단계에서도 시작할 수 있다. 역공학은 대상 시스템의 변경이나 역공학된 대상시스템을 기반으로 새로운 시스템을 생성하는 것은 포함하지 않는다. 즉, 역공학은 시스템을 변경하거나 복사하는 과정이 아니고, 시스템을 시험하고 분석하는 과정이다. 역공학은 기존 시스템으로부터 설계 요소들을 추출하는 작업이지, 목표 시스템을 수정하거나 새로운 시스템을 생성하지 않는다.

■ 재문서화(redocumentation)

재문서화는 관련되는 추상화 수준으로부터 목표 시스템의 동일한 의미의 설명서를 생성하거나 개조하는 작업이다. 재문서화는 가장 간단하고 가장 오래된 유형의 역공학으로 재문서화된 설명서의 결과를 다른 유형(데이터 흐름, 자료구조, 제어 흐름)으로 표현한다. 재문서화를 위해 사용되는 일반적인 도구로 프린터(pretty printer), 다이어그램 생성기, 상호 참조표 생성기 등을 사용한다. 이 도구의

주요 목적은 프로그램 구성요소들 사이의 관계를 쉽게 표현할 수 있는 방법을 제공하는 것이다. 전형적으로 문서화 도구는 자료흐름도, 데이터 모델, 구조도 또는 텍스트기반 프로세스 정의를 생성하기 위하여 기존의 원시 코드를 사용한다.

■ 설계 복구(design recovery)

설계 복구는 시스템을 시험하여 영역 지식, 외부 정보 및 연역/퍼지 추론 등을 직접 획득할 수 있는 유용한 고수준의 추상화를 식별하기 위하여 대상 시스템의 관찰에 추가로 사용되는 역공학의 부분집합이다. 설계 복구는 프로그램의 기능, 작동 방법 등을 사용자가 완벽하게 이해하는 데 필요한 모든 정보를 재생성해야 한다.

■ 재구조화(restructuring)

재구조화는 기존의 소프트웨어에 수정을 가함으로써 이해하기 쉽고 변경이 용이하며 미래의 변화에 품질상 하자를 유발시킬 가능성을 줄이는 작업이며, 대상 시스템의 외적인 동작이 변경되지 않도록 상대적으로 동일한 추상화 수준에서 하나의 표현 유형으로부터 다른 표현 유형으로 변환하는 것이다. 재구조화는 구조적 설계의 전통적 의미에서 구조를 개선시키기 위하여 코드를 변경하는 것이다. 즉, 비구조적인 복잡한 프로그램을 구조적인 프로그램으로 변경시키는 것이 재구조화의 가장 큰 목적이다.

■ 재공학(re-engineering)

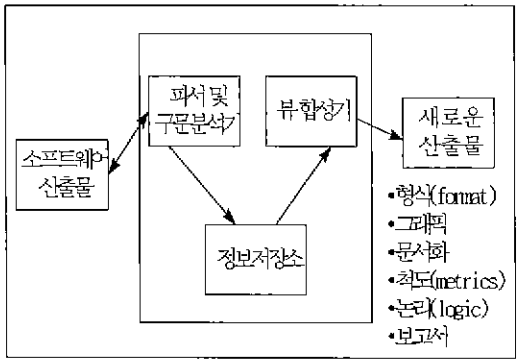
재공학은 대상 시스템을 새로운 요구에 맞도록 분석하고 이해하여 새로운 형태로 변경시켜 구현하는 작업이다. 기업의 재공학이라 함은 현재의 시스템을 평가하고 미래의 요구에 맞게 변화하는 것처럼 소프트웨어 재공학은 먼저 추상화된 정보를 얻기 위한 역공학 작업, 새로운 요구에 적합하도록 변경하여 구현시키는 순공학, 그리고 비구조적인 코드를 구조적인 코드로 변경시키는 재구조화를 포함한다.

2.3 역공학의 목표

소프트웨어 시스템을 위한 역공학의 주된 목적은 소프트웨어 유지보수와 새로운 시스템의 재개발을 위하여 대상 시스템에 대한 전체적인 이해도를 높이는 것이다. 역공학의 주요 목표는 다음과 같다.

■ 복잡한 시스템의 관리

시스템의 규모와 복잡도를 효율적으로 관리할 수 있는 보다 좋은 개발 방법이 요구되고 있다. 이러한 속성들을 효율적으로 관리하기 위한 방법으로 자동화를 지원하는 개발 도구를 들 수 있다. CASE 환경과 함께 역공학의 방법 및 도구는 시스템 진화과정에서 개발 과정과 제품을 제어할 수 있도록 관련되는 정보를 추출하는 방법을 제공한다. <그림 3>은 역공학, 재공학 및 재구조화를 위한 일반적인 도구의 구조를 보여주고 있다.



<그림 3> 역공학, 재공학 및 재구조화를 위한 도구의 구조

■ 다른 뷰의 생성

그래픽 표현은 시스템의 이해를 돕는 방법으로 오래 전부터 사용되어 왔다. 그러나 그래픽 표현을 생성하고 저장하여 유지보수

하는 과정은 매우 어려운 일이다. 한편, 역공학 도구는 다른 표현으로부터 그래픽 표현을 생성하고 재생시키는 것을 가능하게 한다. 대부분의 설계자들을 자료흐름도와 같은 하나의 관점으로 작업을 수행하지만 역공학 도구는 작업 과정의 검토와 검증을 지원하기 위하여 제어흐름도, 구조도 및 개체관계도 등의 다른 관점으로부터 추가적인 뷰를 생성할 수 있다.

■ 잃어버린 정보의 복구

대규모로 장기간 지속되는 시스템이 점진적으로 개발된다면 시스템 설계 과정에서 정보를 잃어버리는 경우가 종종 발생한다. 변경이 빈번하게 발생하므로 코드 수준에서보다 고수준 추상화 단계에서 변경은 문서화에 반영하기 어렵다. 이러한 경우에 설계 내력(history)을 보존하기 위한 대안이 없었지만, 역공학(특히, 설계 복구)은 기존 시스템으로부터 필요한 정보를 보존하는 방법으로 매우 유용하게 사용할 수 있다.

■ 부작용(side effects)의 탐지

초기 설계와 계속적인 변경으로 시스템이 초기에 의도했던 방향으로 구현되지 않는 부작용이 발생하면 사용자의 요구분석 결과와 역공학에 의하여 추출된 정보를 비교함으로써 그 원인을 찾아낼 수 있다. 역공학은 순공학에서 획득할 수 없는 정보를 제공할 수 있으므로 사용자에게 버그(bugs) 등이 보고되기 이전에 이상(anomaly)과 문제들을 탐지할 수 있다.

■ 고수준 추상화의 합성

역공학은 고수준의 추상화 한계를 넘는 여러 가지 표현을 생성하기 위한 방법과 기술을 요구하고 있으며, 이러한 과정을 완전하게 자동화시키는 방법에 대해 많은 논쟁이 발생하고 있다. 역공학은 고수준의 추상화로 표현된 정보를 여러 가지의 유형으로 추출할 수 있으므로 시스템의 이해에 많은 도움을 제공한다.

■ 재사용을 지원

소프트웨어 재사용 환경으로 이동되는 과정에서 중요한 문제는 대 규모의 소프트웨어 부품들을 효율적으로 관리하는 것이다. 역공학은 현존 시스템으로부터 재사용 가능한 소프트웨어 부품을 추출하는 데 도움을 제공한다.

2.4 역공학의 장 · 단점

역공학은 소프트웨어 시스템의 유지보수, 시스템의 전환(migrations)과 변환, 그리고 재사용 가능한 소프트웨어 부품을 식별하기 위하여 사용한다. 역공학은 기존 시스템을 문서화시키고, CASE 도구를 사용하여 설계 명세서 수준에서 시스템을 유지보수함으로써 소프트웨어 유지보수를 지원하고 있다. 소프트웨어를 유지보수하는 사람은 소프트웨어 구조와 구성요소들의 상호관계를 조사하기 위하여 구조도와 개체관계도 등과 같은 그래픽 표현을 사용함으로써 다양한 수준에서 시스템을 고찰할 수 있다. 소프트웨어를 유지보수하는 사람은 설계 명세서 수준에서 시스템을 시험할 수 있을 뿐만 아니라 설계 명세서에 대한 변경도 유지보수할 수 있고, 새로운 코드를 자동적으로 생성하기 위하여 CASE 코드 생성기를 사용한다. CASE 도구를 이용한 설계 명세서 수준에서 시스템의 유지보수는 소프트웨어 유지보수 생산성을 안정적으로 향상시킬 수 있다.

역공학은 고수준의 표준화된 프로그래밍 언어 및 데이터베이스 등과 같은 새로운 기술로 이전시키고, 퍼스널 컴퓨터 및 워크스테이션과 같은 다른 운영 환경으로 전환시킴으로써 기존 시스템의 유용한 생명주기와 가치를 연장시켜 준다. 역공학 과정에 따라 기존 시스템에서 획득한 지식은 새로운 시스템을 개발하는 과정에서 재사용할 수 있다. 예를 들어, 기존 시스템의 설계는 새로

운 대체 시스템의 구축을 위한 프로토타입으로 활용할 수 있다. 기존 시스템의 기능과 대체 시스템 사이의 차이가 60~75% 정도 존재하기 때문에 기존 시스템의 구성요소를 재사용하여 개발 비용과 위험을 안정적으로 줄일 수 있다.

역공학의 장점은 소프트웨어 유지보수 단계에만 한정되는 것은 아니다. 역공학은 직·간접적으로 소프트웨어 개발을 지원한다. 역공학은 기존 시스템에 포함된 풍부한 정보를 찾아낼 수 있도록 지원하는 유용한 방법을 제공하고 있다. 예를 들어, 기존 시스템을 역공학시킴으로써 획득할 수 있는 시스템 설계 구조를 재사용하여 시스템 분석과 설계를 향상시킬 수 있다. 역공학된 설계 정보는 새로운 시스템 설계의 출발점으로 사용할 수 있으며, 새로운 시스템의 요구사항을 만족시키기 위하여 수정할 수 있다. 새로운 시스템이 구축된 후 이 시스템의 실제 설계 구조는 요구사항을 만족하고 있는가를 검사하기 위하여 원시코드로부터 역공학시킬 수 있다. 역공학된 설계 구조는 생성된 시스템이 요구사항을 완벽하게 만족하고 있는가를 결정하기 위하여 초기 설계 구조와 비교한다.

한편, 역공학이 소프트웨어의 유지보수와 재사용을 위한 프로그램의 이해 방법으로 많은 장점을 가지고 있지만 다음과 같은 문제점을 포함하고 있다.

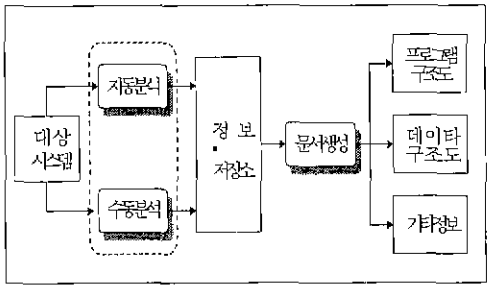
첫째, 역공학은 한시적인 기술(transient technology)로 전락할 가능성이 있다. 즉, 코드 재구조화 도구(code restructuring tool)처럼 업무 초기 단계에서만 사용되고, 향후에는 사용할 가치가 없는 한시적인 제품들이 있을 수 있다. 또한, 10~15년 정도의 기간이 지난 후 순공학의 자동화 기술 분야에 많은 발전이 이루어진다면, 역공학의 필요성은 상대적으로 줄어들게 될 것이다.

둘째, 현재 개발되어 있는 도구들과 기술들의 통합이 부족한 실정이다. 이러한 통합의 부족 현상은 다수의 사용자들로 하여금 역공학 기술을 도입하여 적용하는 것을 망설이게 한다. 즉, CASE 도구와 사용자가 선택한 역공학 도구들의 통합이 어렵거나 불가능하다.

마지막으로 기존에 개발된 코드의 규모가 방대하고, 역공학 및 CASE 환경에 대한 경험과 지식의 부족, 미흡한 표준화 작업, 빈약한 문서화 및 현대화되지 못한 기존의 시스템들이 역공학을 어렵게 만들고 있다.

3. 역공학 과정 및 도구(Reverse Engineering Process and Tools)

일반적으로 역공학의 과정(process)은 <그림 4>와 같다. 역공학은 원시코드와 시스템 구조 모델을 이용하여 대상 시스템을 분석하는 단계에서 시작하며, 시스템을 이해하기 위하여 분석하고 수집한 정보들은 정보저장소에 저장되고 다양한 유형의 문서들이 이 정보로부터 생성된다.



<그림 4> 역공학 과정

소프트웨어 역공학의 유형은 2가지의 시각에서 분류할 수 있다. 기존 프로그램에 대한 사후-개발(post-development) 분석을 수행하는 방법에 따라 정적 역공학과 동적 역공학, 분석을 수행하는 대상에 따라 데이터 역공학과 코드 역공학으로 나눌 수 있다.

3.1 정적 역공학과 동적 역공학

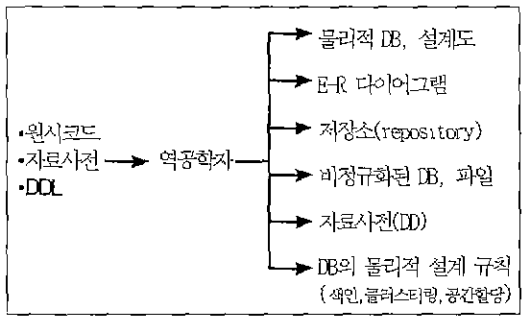
정적 역공학 도구(static reverse engineering tools)

는 가장 일반적인 도구로서 입력으로 프로그램 원시코드를 사용하고, 이를 분석한 다음 프로그램 구조, 제어 구조, 논리 흐름, 자료구조, 그리고 자료 흐름을 추출한다. 이 범주에 속하는 다른 도구들은 "프로그램 슬라이싱(slicing)"이라고 하는 방법을 적용한다. 소프트웨어 엔지니어가 관심있는 프로그램 구조의 유형을 기술하면, 역공학 도구는 관련 없는 코드를 제거하여 관심 있는 코드만 표현하게 해준다. 의존도 분석 도구(dependency analysis tools)는 이미 논의했던 대부분의 기능을 수행하지만, 이 소범주에 포함되는 도구들은 자료구조, 프로그램 구성요소, 그리고 다른 사용자-특수 프로그램 특성들 간의 관계를 보여주는 그래프적인 의존도 그림(dependency map)을 생성한다. 정적 역공학 도구들은 "코드 시각화(code visualization)" 도구라 부른다. 사실 소프트웨어 엔지니어가 프로그램을 시각화할 수 있게 함으로써 이러한 도구들은 변경된 제품의 품질을 개선시키고, 변경을 수행하는 사람들의 생산성을 향상시킨다.

동적 역공학 도구(dynamic reverse engineering tools)들은 프로그램의 행위 모형(behavior model)을 구축하기 위해 감독하는 동안 획득한 정보를 사용하고 실행함으로써 소프트웨어를 감독한다. 비록 이러한 도구들이 거의 없지만 이들은 실시간 소프트웨어나 내장 시스템을 유지보수해야 하는 소프트웨어 엔지니어들에게 매우 중요한 정보를 제공해준다.

3.2 데이터 역공학과 코드 역공학

데이터 역공학(data reverse engineering)은 기존 시스템의 이해와 조직이 새로운 기술로 이전하기 위하여 사용된다. 예를 들어, 데이터 역공학은 기존 데이터베이스를 수정할 뿐만 아니라 IMS에서 DB2로 이전과 같은 새로운 데이터베이스 관리 시스템으로 이전하기 위해 사용할 수 있다. 또한 데이터 역공학은 논리적 데이터 모델을 구축하기 위한 첫 번째 단계로 사용 할 수 있다. 기존 시스템에 포함된 데이터 모델을 개선시키고 역공학시킴으로써 현재의 데이터 모델을 신속하게 구축할 수 있다. 이 데이터 모델은 기업의 전략 계획에서 요구되는 데이터 모델 구축의 출발점이 된다. 데이터 역공학은 데이터 모델링 과정을 촉진하고 개선시키기 위하여 하향식 기법과 상향식 기법의 통합을 지원한다.



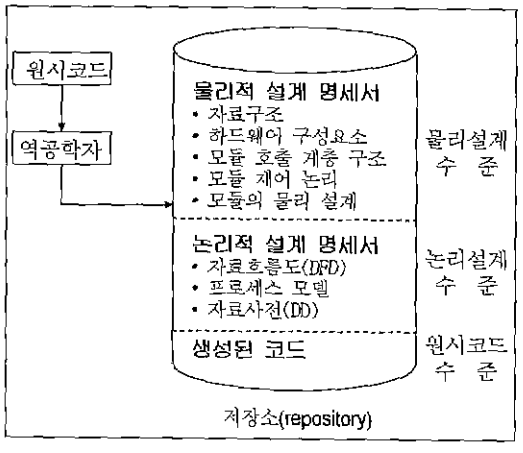
<그림 5> 데이터 역공학 과정

데이터 역공학 과정은 <그림 5>에서 처럼 물리적 데이터 기술로부터 개념적이고 논리적인 데이터 구조에 관한 정보를 추출한다. 즉, 이 과정에서는 원시코드, 자료 사전(DD), 자료정의 언어(DDL)로부터 개체, 관계, 속성 및 물리적 설계 명세서 등의 추출을 포함하고 있다. 또한 개체관계도(E-R 다이어그램) 및 다이어그램의 온라인 그래픽 표현 등과 같은 물리적 데이터베이스 다이어그램과 논리적 다이어그램의 생성, 수정, 개선 및 검증을 포함하고, 갱신된 논리적 다이어그램과 물리적 다이어그램으로부터 DDL 스키마의 재생성도 포함한다.

코드 역공학(code reverse engineering)은 원시코드 또는 논리적 설계로부터 프로세스의 설계 명세서를 추출하고, 저장소에 저장시킬 정보를 생성, 수정, 개선 및 검증하여 저장시키는 일련의 과정이다. 물리적 설계 명세서들이 저장소에 저장되어 있으면, 설계 명세서는 시스템을 문서화하고, 효과분석의 변경 및 시스템을 새로운 환경으로 전환시키기 위하여 부분 혹은 전체 프로그램을 재생성하

여 사용할 수 있다. 코드 역공학과 프로그램 코드 분석은 프로그램의 물리적 기술을 입력으로 사용하고, 프로세스의 코드 구성요소를 기술한 문서를 출력으로 사용하는 점에서 매우 유사하다.

코드 역공학 도구는 <그림 6>에서 처럼 코드 역공학 과정을 지원하는 저장소, 원시코드를 포함하는 저장소, 그리고 물리적 및 논리적 프로그램 설계 정보를 포함하는 저장소를 필요로 한다. 가장 중요한 물리적 설계 저장소는 상세한 수준의 정보, 에러 검증 및 코드 역공학 과정의 특성을 결정하는 요소들을 문서화시켜 저장한다. 원시코드는 역공학되어 물리적 설계 저장소로 저장되고, 설계 명세서는 논리적 설계 저장소로부터 역공학된다. 한편, 물리적 설계 저장소에 있는 설계 명세서들은 수정되고 변경된 내용을 검증 받아야 한다. 유지보수하는 사람 혹은 분석가는 물리적 명세서를 분석하고 그래픽 표현을 생성하여 문서화시키며, 순공학자는 변경된 다이어그램으로부터 물리적 설계에 영향을 미치는 일부 혹은 전체 원시코드를 생성하여 원시코드 저장소에 저장한다. 역공학 과정에서 중요한 것은 설계가 역공학자에 의해 물리적 수준으로부터 논리적 수준으로 유도되는 것이 아니라 순공학자에 의해 논리적 수준으로부터 물리적 수준으로 유도된다는 점이다.



<그림 6> 코드 역공학 과정

4. 역공학을 위한 시스템과 도구의 선택

4.1 역공학을 위한 적합한 시스템의 선택

역공학 프로젝트를 시작하기 전에 대상 시스템 환경과 자동화 지원의 가능성을 신중하게 고려해야 한다. 역공학 기법은 모든 시스템들에 대해 적용할 수 있지만, 어떤 경우에는 가장 적합한 기법임에 불구하고도 비용 측면에서 비효율적인 경우가 있다. 다음의 질문은 역공학을 위한 시스템의 선택 과정에서 고려해야 하는 사항들이다.

(1) 역공학 도구들이 대상 시스템의 환경에서 이용할 수 있는가?

역공학자가 자동화 도구의 지원없이 기존 시스템을 분석하여 이해하는 데 요구되는 노력은 새로운 시스템을 개발하기 위해서 요구되는 노력과 거의 비슷하다. 따라서, 가장 좋은 방법은 기존 도구들이 요구되는 환경에서 적절하게 사용할 수 있어야 한다.

(2) 코드는 구조적으로 작성되었는가?

역공학 도구들이 원시코드로부터 유용한 설계 명세서를 추상화시키 위해 우선 체계적으로 설계되고, 이 설계는 코드로 표현되어야 하며, 코드는 구조적 방법으로 작성되어야 한다. 코드가 구조적이지 못하면 코드 재구조화 도구를 사용해야 하며, 코드 분석 도구는 "good" 코드와 "bad" 코드를 식별하기 위해서 사용된다. 따라서 코드의 분석과 재구조화는 프로젝트에서 부가적인 비용을 필요로 한다.

(3) 프로젝트에서 구조적 기법과 CASE 도구들을 사용하고, 팀원들이 순방향 기법들을 명확하게 이해하고 있는가?

역공학 프로젝트로부터 최대의 이익을 얻기 위해서 대상 시스템은 개발 방법론과 엄격한 구성관리 기법을 활용해야 한다. 만약 이러한 방법론과 기법을 활용하지 않으면 시스템의 초기 상태와 설계 문서들이 조잡하게

되어 매우 빈번하게 초기 상태로 되돌아 가야 한다.

(4) 업무 규칙(논리)이 동적인 요소를 포함하고 있는가?

역공학의 가장 좋은 대상 시스템은 핵심 요구사항들이 높은 재사용성을 갖도록 안정적이어야 한다. 자주 변경되는 업무 규칙을 갖는 시스템은 초기 설계 구조가 자주 변경된다.

(5) 기업경영에 시스템이 얼마나 중요한 역할을 수행하는가?

역공학이 필요한 경우 가장 현명한 방법은 역공학으로부터 대부분의 이익을 얻을 수 있는 프로그램들을 식별하는 것이다. 즉, 이익의 80% 정도가 역공학의 결과로부터 생길 수 있는 80/20 규칙을 종종 적용해야 한다.

(6) 시스템 개발에 참여한 초기의 분석가, 설계자 및 프로그래머가 조직에 아직 남아있는가?

역공학 도구들이 많은 양의 입력을 요구하는 상태이면 적절한 도구를 선택해서 활용해야 한다. 따라서 프로젝트에 참여한 팀원의 전문지식이 많으면 많을수록 성공할 가능성이 높게 된다.

(7) 다수의 언어들이 시스템에서 어떻게 지원하는가?

하나의 시스템이 종종 3개 혹은 그 이상의 다른 언어를 포함하는 경우가 있다. 어셈블리어 루틴과 다수의 JCL 문장을 포함하는 시스템이 코볼로 작성될 수 있다. 도구들은 이러한 경우를 해결할 수 있어야 한다.

(8) 다른 시스템의 시스템 인터페이스를 어느 정도 이용할 수 있는가?

역공학이 순공학의 중간 결과를 이용하기 때문에 이들 인터페이스의 무결성을 유지보수하기 위해 숙달되어 있어야 한다. 코드의 변경이 없다고 예상되더라도 인터페이스의 한 측면으로부터 인터페이스 설계 정보를 추상화시키는 것은 매우 어려운 일이다.

(9) 내장(in-house) 서브루틴을 어느 정도 사용할 수 있는가?

표준 서브루틴이 시스템에서 사용되었다면, 어떤 한 시스템의 요구사항과 함께 데이터와 프로세스를 포함할 것이고, 수작업으로 중복된 코드의 식별을 요구한다. 보다 많고 복잡한 내장 서브루틴을 포함하면 수작업을 위한 요구사항이 많아진다.

(10) 작업환경에 영향을 미치는 역공학 방법을 어떻게 도입할 것인가?

새로운 도구와 기법들의 등장은 유지보수팀의 작업 방법에 있어서 획기적인 변화를 가져온다. 개발자와 유지보수자의 역할에 분석가, 설계가 및 프로그래머의 역할도 어느 정도 포함하게 될 것이다.

4.2 역공학 도구의 선택

특정 프로젝트를 위해 사용할 도구 혹은 도구들을 선택하기 전에 프로젝트의 전체 목표와 중간 목표를 식별하는 것이 매우 중요하다. 완벽한 재공학 과정이 계획된 경우와 유지보수를 지원하기 위해 선택적인 프로그램 역공학이 계획된 경우의 요구사항은 분명하게 다를 것이다. 하나의 역공학 방법이 결정되어야 하고, 최종 산출물과 중간 산출물도 결정된다. 다음은 역공학 도구를 선택하는 과정에서 고려해야 하는 사항들이다.

(1) 언어의 적용 범위(language coverage)
사용자들이 필요로하는 언어와 언어의 특성을 도구가 지원하는가?

(2) 지원되는 기술(techniques supported)
기존 개발 방법에서 사용한 설계 기법과 설계 산출물들의 유형을 도구가 지원하는가?

(3) 지원되는 저장소(repository support)
도구에서 사용하는 저장소의 유형은 어떤 것이고, 메타 데이터 계층을 지원하는 방법

은? 그리고 저장소가 기존 CASE 도구에서 사용가능한가?

(4) 환경(environment)
도구가 어떤 환경에서 실행되는가? 대부분의 역공학 도구와 CASE 도구는 퍼스널컴퓨터와 워크스테이션에서 실행되고 있으며, 일부 도구들이 미니 혹은 메인프레임 환경에서 실행된다.

(5) 도구 판매자의 지원(vendor support)
도구 판매자의 지원, 교육 및 자문에 대한 서비스 수준과 비용이 어느 정도인가?

(6) 고객의 요구 수용(tailoring)
도구가 사용자 또는 판매자들에 의해서 고객의 요구를 수용할 수 있는가?

(7) 참조 사이트(reference sites)
적합한 참조 사이트를 방문할 수 있는가?

(8) 추상화의 유형 및 수준(type and level of abstraction)
생명주기 각 단계의 역으로 진행하기 위해 제공되는 지원 수준은? 대부분의 도구들은 코드로부터 내부 설계로의 역변환만을 자동으로 지원하며, 나머지 일부 도구들은 고수준의 추상화를 지원한다.

(9) 통합(integration)
선택할 도구와 같은 회사 또는 다른 회사에서 개발한 다른 제품들의 통합 방법은? 만약 재공학 프로젝트를 수행하려고 한다면, 역공학 도구에 의해 저장소에 저장되는 객체들을 관리하기 위한 순공학 CASE 도구들이 있어야하고, 이 순공학 도구들로부터 코드를 자동으로 생성할 수 있어야 한다.

(10) 업체의 표준화(industry standards) 지원
저장소, 네트워킹, 사용자 인터페이스 등의 표준화를 어느 정도 지원하고 있는가?

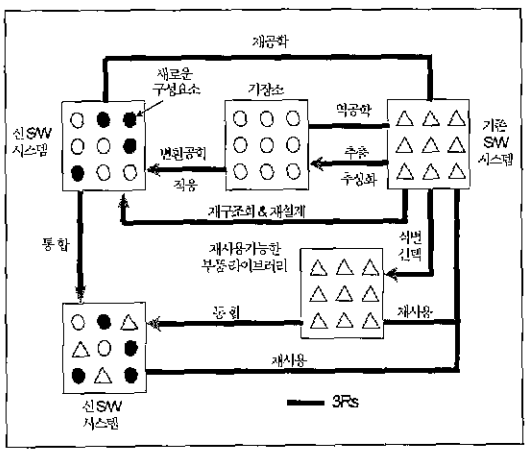
5. 소프트웨어 자동화를 위한 3Rs

3개의 소프트웨어 기술 용어, "재사용(reuse)", "재공학(re-engineering)", "역공학(reverse-engineering)"[3Rs]의 의미와 관계를 정확하게 정의할 필요가 있다. 이 세 용어 사용의 혼란은 소프트웨어 관리자들 뿐만 아니라 소프트웨어 공학의 전문가들 사이에서도 발생하는 일반적인 현상이다. 일부 CASE 기술 옹호자들은 "저장소(repository)", "재사용", 그리고 "재공학"을 3Rs로 정의하고 있지만, "역공학"을 "저장소" 대신 3R에 포함시켜야 한다. 저장소의 개념은 이들 세 가지 공학의 목표 중의 어떤 하나를 구현하는 수단이기 때문에 다른 2개의 Rs와 한 부류로 할 수 없다. 비록 3Rs에 대한 단 하나의 정확한 정의가 소프트웨어 공학 사회에 존재하지 않지만, 구별이 어려운 이들 세가지 소프트웨어 기술의 상호관계를 이해할 수 있도록 설명하고자 한다.

정보시스템 공학은 대부분 4가지 방법, 즉 순공학(forward engineering), 역공학(reverse engineering), 재공학(re-engineering), 변환공학(transverse-engineering)으로 관리한다. 역공학, 재공학 및 변환공학은 그들 활동을 위한 자원으로 이미 개발된 기존 시스템들에 공통적인 기초를 두고 있다. 일반적으로 목표 지향 변환(goal-oriented transformation)은 이들 기술을 토대로 자원을 활용한다. 역공학은 코드로부터 고수준 명세서를 유도하고, 기존의 코드로부터 명세서를 유도한다. 변환공학은 유도된 명세서를 특정의 응용 설계 명세서로 변환한다. 이 초기 영역은 각 공학 기술이 상호 종속되고, 목표와 기법들로 확장되는 역할을 함으로써 불명료하게 만든다. 따라서 변환공학 활동은 역공학의 실제(practices)가 되며, 역공학 활동은 재공학의 예제(exercises)로 합병된다.

3Rs의 정의는 다음과 같다. "재사용(reuse)"은 새로운 컴퓨팅 시스템에 적용하기 위하여 정확한 수집, 재구성, 적응을 위한 재사용 가능한 소프트웨어의 식별에 초점을 둔 소프트웨어 공학 활동으로 정의한다. "재공학(re-engineering)"은 널리 보급된 프로그래밍 표준화에 순응하고, 쉬운 유지보수를 위한 고급 언어(high order language)로 구현,

다른 하드웨어 플랫폼들에 대한 재조정(rehost) 및 다른 컴퓨터 시스템 구조들에 대한 재설정(retarget) 등의 4가지 목표를 위해 기존의 시스템을 변환시킨 결과로 설계되는 소프트웨어 공학 활동으로 정의하며, 일반적으로 재공학은 기존의 "bad" 시스템을 새로운 "good" 시스템으로 변환시킨다. "역공학(reverse-engineering)"은 기존의 소프트웨어 시스템들로부터 설계명세서, 소프트웨어 부품들의 컴퓨터 지원 추출에 적합한 소프트웨어 활동으로 정의하며, 역공학은 기존의 "good" 소프트웨어 시스템들로부터 추상 명세서의 유도와 변환 공학 단계들을 포함한다. <그림 7>은 이들 세 가지 개념과 기술들 사이의 상호관계를 나타낸다. 3Rs에 대한 적절한 노력은 소프트웨어 개발 기간과 비용 측면에서 많은 절약을 가져올 것이다.



<그림 7> 재사용, 재공학 및 역공학의 상호 관계

비록 많은 소프트웨어 엔지니어들이 이 정의에 동의하지 않지만, 이들 세 용어를 위한 핵심 개념의 이해는 이들 활동에 더욱 많이 관계된다. 재공학은 재설계(redesign), 재개발(redevelopment) 활동을 강조하고, 역공학은 변환 기법들을 강조한다. 가장 핵심적인 소프트웨어 재사용은 재사용 가능한 코드와 명세서의 식별과 이러한 식별을 기초로 유용한 라이브러리를 구축하는 것이다. 일부 학자들은 역공학이 다른 두 기술을 지원한다고 생각하지만, 실은 그렇지 않다. 역시 재공학은 간단하게 역공학된 부품들 없이 가능하다 예를 들어 초기 시스템에 사용된 같은 구현 언어로 된 원시 코드의 재공학은 "기능개선(perfective)" 소프트웨어 유지보수 과정에서 실제로 보여준 재구축과 재설계 활동들을 반영한다. 그러나 역공학된 부품들의 재공학은 대부분의 재공학 활동으로 대체된다. 이것은 비용절감, 설계의 복구, 재문서화, 일정 절약의 기초에서 가능하다. 일단 특정의 프로젝트에 적용을 위해 기존 소프트웨어 시스템들로부터 역공학된 사용 가능한 부품과 재공학된 이들 부품이 성공하면, 이 성공은 분명한 소프트웨어 재사용을 위해 평가된다. 비록 역공학과 재공학 기술의 필요 없이 쉽게 재사용될 수 있도록 작성된 소프트웨어일지라도 소프트웨어 재사용은 역공학과 재공학 기술에 의존하고 있다.

저장소의 의미는 단어 "저장소"가 사용된 문장에 종속되고, 다양화될 수 있기 때문에 "저장소"의 의미를 잘못 이해해서는 안된다. 단어 "저장소"가 재사용에 사용될 때, 저장소는 재사용 가능한 부품을 효율적으로 검색하기 위해 디렉토리에 저장되는 라이브러리의 의미이다. 한편, 단어 "저장소"가 재공학(또는 역공학)에서 사용되면, 재공학된 명세서/코드의 컴퓨터 지원 관리에 적합한 데이타베이스관리 구조 아래의 데이터베이스를 의미한다.

6. 맺음말

소프트웨어 개발 생명주기(SDLC)의 마지막 단계인 유지보수는 소프트웨어를 개발하는 과정에서 사용되는 비용의 대부분을 차지하고 있다. 더 많은 프로그램이 개발됨에 따라 소프트웨어 유지보수 단계에서 사용되는 노력과 자원의 양이 증가하는 심각한 현상이 발생하고 있다. 궁극적으로 일부 소프트웨어 조직들은 유지보수-전담 조직이 되고, 그들의 모든 자원들이 자신의 오래된 소프트웨어들을 유지보수하는 데 사용되기 때문에 새

로운 프로젝트를 착수할 수가 없게 된다.

소프트웨어 유지보수의 과정에 대한 기술과 관리의 접근 방법은 별다른 변화 없이 구현될 수 있으나 소프트웨어 공학과정에서 수행된 업무들은 유지보수성을 정의하고, 어떠한 유지보수 접근 방법의 성공에 중요한 영향을 미친다.

역공학과 재공학 도구 및 기법들은 1990년대 후반부에 소프트웨어 유지보수와 재사용을 위한 프로그램의 이해 과정에서 매우 중요한 역할을 수행하게 될 것이다. 역공학은 다른 문서를 사용할 수 없을 때, 프로그램의 원시코드로부터 설계 정보를 추출하고, 재공학은 얻어진 정보를 이용하여 보다 좋은 품질을 얻기 위해서 그 프로그램을 다시 구성한다. 이렇게 하여 장래에 보다 좋은 소프트웨어 유지보수성을 갖도록 할 수 있다.

참고문헌

[1] Ahrens, J.D., Prywes, N., and Lock, E., "Software Process Reengineering: Toward a New Generation of CASE Technology," Journal of System Software 30(2), pp.71~84, 1995.

[2] Biggerstaff, T.J., "Design Recovery for Maintenance and Reuse," IEEE Computer 22(7), pp.36~49, July 1989.

[3] Byrne, E.J., "Software Reverse Engineering: A Case Study," Software - Practice and Experience 21(12), pp.1349~1364, December 1991.

[4] Chikofsky, E.j., and Cross, J.H., "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software 7(1), pp.13~17, January 1990.

[5] Fiore, P., Lanubile, F. and Visaggio, G., "Analyzing Empirical Data from a Reverse Engineering Project," Working Conference on Reverse Engineering, pp.106~114, July 1995.

[6] Frazer, A., "Reverse Engineering - hype, hope, or here?" in Software Reuse and Reverse Engineering in Practice, Hall, P.A.V.(ed.), pp.209~243, Chapman&Hall, 1992.

[7] Hall, P.A.V., "Software Reuse, Reverse Engineering and Re-engineering," in Software Reuse and Reverse Engineering in Practice, Hall, P.A.V.(ed.), pp.3~31, Chapman&Hall, 1992.

[8] Harandi, M.T., and Ning, T.Q., "Knowledge-Based Program Analysis," IEEE Software 7(1), pp.74~81, January 1990.

[9] Jarzabek, S, "Domain Model Driven Software Reengineering and Maintenance," Journal of System Software 20(1), pp.37~51, 1993.

[10] McClure, C., *The Three Rs of Software Automation : Reengineering, Repository, Reusability*, Prentice Hall, 1992.

[11] McGill, R., "Reverse Engineering - not yet?" in Software Reuse and Reverse Engineering in Practice, Hall, P.A.V.(ed.), pp.245~252, Chapman&Hall, 1992.

[12] Pressman, R.S., *Software Engineering : A Practitioner's Approach*, McGraw-Hill, 1992.

[13] Rugaber, S. and Clayton, R., "The Representation Problem in Reverse Engineering," Working Conference on Reverse Engineering, pp.8~16, May 1993.

[14] Selfridge, P.G., Waters, R.C. and Chikofsky, E.J., "Challenges to the Field of Reverse Engineering," Working Conference on Reverse Engineering, pp.144~150, May 1993.

[15] Tomic, M., "A Possible Approach to Object-Oriented Reengineering of COBOL Programs." ACM SIGSOFT Software Engineering Notes 19(2), pp. 29~34, 1994.

[16] Waters, R.C. and Chikofsky, E.J., "Reverse Engineering: Progress along Many Dime nsions, "Communication of the ACM 37(5), pp.23~24, 1994.

[17] Wilkening, D.E, and Loyall, J.P., "A Reuse

Approach to Software Reengineering," Journal of System Software 30(3), pp.117~125, 1993.

김 태 희

1991년 동신대학교 전자계산학과(공학사)
1993년 전남대학교 대학원 전산통계학과(이학석사)
1996년 전남대학교 대학원 전산통계학과(박사수료)

1993년 3월 ~ 현재 동신대학교 전자계산학과 시간강사
관심분야 : 소프트웨어공학, 객체지향시스템

강 문 설

1986년 전남대학교 계산통계학과(이학사)
1989년 전남대학교 대학원 전산통계학과(이학석사)
1994년 전남대학교 대학원 전산통계학과(이학박사)

1983년 9월 ~ 1985년 12월 제1군수지원단 전산실
1989년 4월 ~ 1994년 8월 전남대학교 전산학과 조교 및 강사
1994년 9월 ~ 현재 광주대학교 전자계산학과 전임강사
관심분야 . 소프트웨어공학(재사용, 재공학, 역공학), 객체지향시스템, 정보검색