2

소프트웨어 가치와 품질에 대한 인식전환

한양대학교 ▌이 욱 세*

1. 서 론

최근 급격하게 이공계 기피 현상과 함께 전산학(computer science)에 대한 학생들의 기피 현상이 두드러지고 있다. 이공계 기피 현상은 이미 여러 언론 매체를 통해 잘 알려진 사실이고, 전산학에 대한 기피 현상은 학부체제를 유지하고 있는 대부분의 국내 대학에서 관찰되고 있다. 학부 1학년을 마치고 나서 전산학과 전자공학을 선택하게 될 때 예전에는 전산학에 학생들이 몰리는 경향이 있었으나 최근에는 역으로 전산학을 기피하여 선택하는 경향이 높아지고 있다. 이공계 기피와 함께 전산학에 대한 기피로 인하여 국내 대학에서는 우수한 인력을 전산학 분야에 끌어 모을 수가 없다.

전산학의 위상이 낮아지면 향후 소프트웨어 전문 인력의 품질이 낮아지고, 결과적으로 산업 전반적으로 문제를 유발할 수 있다. 기술의 흐름을 볼 때 예전에 물리적으로 작동하는 기기들이 현재는 전자공학을 기반으로 작동하고 있고, 점차적으로 소프트웨어로 작동하는 방향으로 진화하고 있다. 예를 들어, 자동차의 경우 예전에는 모든 것이 물리적으로 작동되었으나, 현재는 자동차의 핵심 부품들이 전자적으로 제어되고 있다. 향후에는 핵심 기술이 소프트웨어로 작성되어 자동차에 장착될 것이다. 이를 개발하는 프로그래머의 능력이 우수하지 못하면 자동차의 성능과 안전성에 영향을 주고 결과적으로 경쟁력을 잃게 될 것이다.

물론 이런 추세는 시대의 흐름과 환경적 변화에 따라 급작스럽게 바뀔 수도 있지만 프로그래밍에 대한 여러 잘못된 사회적 인식들이 전망을 어둡게 한다. 아래 내용들은 대학 새내기들이 프로그래밍에 대해 갖고 있는 시각과, 전산학 전공자가 아닌 기술자들이 프로그램에 대해 갖고 있는 잘못된 시각들을 나열해 본 것이다.

- 프로그래밍은 시간이 많이 투자되지만 그 노력에 비해 보상이 적다. 프로그래머들은 모두 밤샘 작업을 한다. 그럼에도 불구하고 그 노력에 대한 대가는 다른 제품에 비해 인정받지 못한다. 물리적 장치를 개발하는 사람은 그 결과가 가시적으로 보이지만 프로그램은 가시적으로 보이지 않는다. 결과적으로 개발에 들인 노력에 비해 결과물이 평가 절하된다.
- 프로그래밍은 아무나 할 수 있다. 기계공학이나 전자공학은 학원에서 배울 수 없지만, 프로그래밍은 학원에서 쉽게 배울 수 있다. 전산학을 전공하더라도 기술 장벽이 높지 않아 비전공자가 프로그래밍 영역에 쉽게 진입할 수 있다. 예를 들어 자동차 소프트웨어는 기계공학 전공자가 쉽게 프로그래밍 기술을 습득하여 개발 가능하다.
- 프로그래밍은 다른 중요한 작업에 대한 지원 도구일 뿐이다. 휴대폰의 물리적 장치를 개발하는 것이 가치 있는 일이지 그 안에 소프트웨어를 개발하는 것은 쉽게 누구나 할 수 있는 것이다. 특히 휴대폰 소프트웨어는 물리적 장치를 개발한 전자공학 전문가가 전산학 전공보다 장치에 대한 지식이 많기 때문에 보다 쉽게 개발할 수 있다.
- 프로그램은 개발하기 쉽다. 누구에게나 단시간을 주면 개발할 수 있다. 그렇기 때문에 전체 개발 공정에 있어서 디자인과 하드웨어 개발 공정에 많은 시간을 부여하고, 프로그램 개발 시간은 크게 고려하지 않아도 된다.

이런 잘못된 사회적 인식은 필자가 판단하기에 모두 소프트웨어의 가치와 품질에 대한 인식 부족으로 인하여 발생한 것이라고 판단된다.[1] 프로그램은 개발

* 종신회원

1) 물론 현재의 한국의 산업구조에 기인한 것도 있다. SI와 게임, 검색 엔진 업체를 제외하고는 현재의 산업구조 상 소프트웨어는 다른 산업의 보조 역할을 하고 있다. 다른 산업의 보조 역할이 아닌, 독립적인 소프트웨어 산업이 발전하거나 소프트웨어가 다른

하기 쉽다. 하지만 품질 좋은 소프트웨어를 개발하기는 쉽지 않다. 프로그래밍은 아무나 할 수 있다. 하지만 품질 좋은 프로그램 개발은 아무나 할 수 없다. 프로그램 개발이 쉽다는 잘못된 인식으로 인하여 소프트웨어 개발 예상 시간을 짧게 잡아 프로그래머들이 밤샘작업을 할 수 밖에 없다. 현재는 소프트웨어가 하드웨어의 보조 도구로 인식되지만 향후에는 소프트웨어가 주도구가 될 것이다. 휴대폰의 경우에도 과거에는 통신 성능에 그 가치가 많이 부여되었지만 현재는 소프트웨어의 품질과 디자인으로 승부가 갈리고 있다.

필자의 판단으로는 소프트웨어의 가치와 품질에 대한 인식 수준을 높이는 것이 전산학의 위상을 높이는데 중요한 기여를 할 것으로 본다. 이러한 인식 수준을 높이는 것은 단순히 강연과 홍보를 통해서만 되는 것은 아니라고 판단된다. 소프트웨어의 가치와 품질을 높이는 법적/제도적 장치 마련, 관련된 기술의 개발, 그리고 더불어 품질 좋은 소프트웨어에 초점을 둔 교육과정을 개발하는 것이 우리가 할 수 있는 노력이라고 판단된다.

2. 소프트웨어 및 소스코드 저작권 보호

전산학의 가치를 인정받으려면 전산학의 최종 결과물인 소프트웨어의 가치를 인정받아야 한다. 소프트웨어의 가치가 인정되면 이를 개발하는 프로그래머의 가치도 인정되고, 결과적으로 전산학의 위상도 높아질 것으로 판단된다.

하지만 소프트웨어는 무형의 자산이기 때문에 가치를 인정받지 못할 확률이 높다. 자동차를 구매하면 커다란 자동차가 눈앞에 나타나고, 휴대폰을 구매하면 앙증맞은 휴대폰이 손에 쥐어진다. 하지만 소프트웨어는 CD 한 장 또는 심지어 내려받기(download)를 하면 보이지 않게 컴퓨터에 저장되어 가시적인 구매의 효과가 없다. 결국 사용자들은 눈에 보이지 않는 소프트웨어에 비싼 가격을 지불하고 싶지 않게 된다.

그러다보니 소프트웨어의 가치를 훼손시키는 불법 행위가 사회에 만연되어 있다. 그 첫 번째는 소프트웨어 불법 복제이다. 돈을 주고 구입하여야 하는 소프트웨어를, 정당한 대가를 지불하지 않고 불법으로 복사하여 사용하는 행위가 만연하다. 이러한 행위는 결국 소프트웨어 개발자의 노력과 그 결과로 나온 제품인 소프트웨어의 가치를 심각하게 훼손하는 행위이다.

다행히 정부에서 소프트웨어 불법 복제의 심각성을 인식하고 이를 근절하기 위한 노력이 계속되어 왔다. 하지만 여전히 소프트웨어 불법 복제는 꾸준히 행해지고 있다. 문제는 사용자들의 인식이 바뀌지 않는다는 것이다. 이를 근절하기 위해서는 소프트웨어 불법 복제에 대한 사회적 인식을 확산시키는 것 외에도 물리적인 장치들이 필요하다.

- **소프트웨어 불법 복제에 대한 법적/사회적 인식 확산 필요:** 음원, 동영상에 대한 불법 복제뿐만 아니라 소프트웨어 불법 복제에 대한 사회적 인식은 정부와 민간의 홍보에 의하여 점차 확산되고 있다. 하지만 여전히 불법 복제는 만연하고 있는데, 이는 어렸을 때부터 교육이 되지 않아서 또는 관행적으로 불법 복제를 사용하는 행태가 용인되는 것에 기인한다. 대학 새내기들의 경우 과목 수강 중에 숙제를 복제하는 행위가 많이 적발되는데 이는 여전히 "복제"라는 것이 얼마나 큰 비윤리적 행위인지 인지를 못하는 것에 기인한다. 청소년 시기부터 저작권에 대한 교육이 필요하다. 이른바 전산학을 전공하고 있는 대학원 연구실에서도 여전히 소프트웨어 불법 복제는 만연하고 있다. 자신이 개발할 프로그램의 가치를 인정받기 위해서는 남이 개발한 소프트웨어의 가치를 먼저 인정하는 뼈아픈 자기반성이 필요하다.
- **소프트웨어 저작권 삽입 및 불법 사용 방지 기술[2] 개발:** 최근 출시되는 대부분의 유료 소프트웨어는 어떤 방식으로든 소프트웨어 불법 사용을 위한 방지 기술을 채택하고 있다. 가장 단순한 형태는 일련번호로, 일정한 형식을 만족하는 번호를 입력해야만 프로그램이 동작하는 것이다. 이는 일련번호가 유출되거나 일련번호 체크 루틴이 호출되지 않도록 조작하는 것을 통해 쉽게 무용화된다. 보다 진화된 방법은 전화 또는 인터넷을 통하여, 설치 기계에 특화된 일련번호를 발급하는 방식이다. 이는 특정 기계에만 특화되어 있기 때문에 다른 기계에서 사용할 수 없어 불법 사용의 확률을 떨어뜨린다. 단점은 특정 기계와 동일하게 인식되는 기계를 사용하거나, 역시 일련번호 체크 루틴이 호출되지 않도록 조작하는 것을 통해 무용화될 수 있다. 이러한 불법 사용 방지 기술이 진화할수록 악의적인 불법 사용자가 아닌 관행적인 불법 사용자를 줄이는데 일조할 수 있을 것으로 판단된다.

산업의 핵심을 담당하게 된다면 이러한 잘못된 인식은 쉽게 교정될 수 있을 것이다.

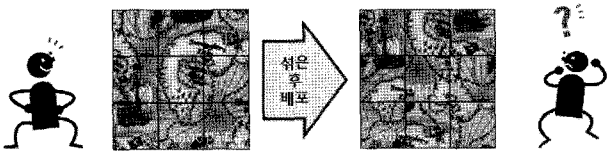
2) http://en.wikipedia.org/wiki/Copy_prevention

최종 실행 파일의 형태에 대한 불법 복제뿐만 아니라 최근에는 소스코드 도용도 심각한 문제로 대두되고 있다. 오랜 시간과 큰 비용을 투자하여 프로그램을 개발하였는데 그 소스코드가 쉽게 다른 회사에 넘어가 불법 재사용된다면 이를 개발한 프로그래머의 노력이 물거품이 된다. 이는 소프트웨어 불법 복제보다 그 피해가 심각한데, 소프트웨어 불법 복제는 단순히 하나의 상품에 대해 그 가치가 훼손된다고 본다면, 소스코드 불법 도용은 제품의 설계도가 도용된 것과 같기 때문에 그 상품의 모든 가치가 훼손된다.

특히 소스코드 도용은 기업의 생존여부에 심각한 위협을 가한다. 특정 응용 프로그램을 많은 투자액을 들여 수년간 개발한 벤처기업이 제품 개발이 완료되어 수익을 창출할 수 있는 시기에 왔을 때, 개발된 소스코드가 도용되었다고 하자. 그러면 도용한 경쟁업체가 단기간 내에 유사한 소프트웨어를 개발해 낼 수 있고, 결과적으로 원 개발업체의 수익성이 약화되어 도산하는 경우가 발생할 수 있다. 이는 단지 국내 경쟁사간의 생존경쟁 문제가 아니라 외국으로 소스코드가 도용되어 국부가 유출될 수도 있는 것이다.

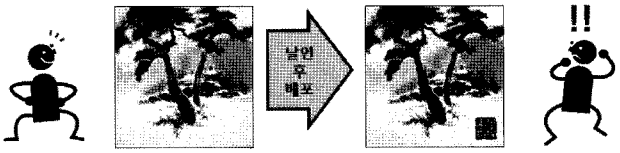
정부에서도 이러한 심각성을 인지하고 소스코드의 불법 도용을 막기 위해 필요한 기술과 제도에 대해 고려하고 있다. 하지만 아직은 초기 단계이고 소스코드의 불법 도용을 막기 위한 기술들이 보다 진화되어야 할 것으로 보인다. 소스코드 불법 도용 방지를 위한 방법과 기술은 다음과 같다.

- **소스코드 불법도용에 대한 법적 장치 확립 필요:** 현재 소스코드 불법 도용은 저작권보호법에 의해 처벌되고 있다. 하지만 저작권보호법은 소스코드가 아닌 문학, 음원, 동영상 등에 특화된 법으로 소스코드에 적합하지 않다. 저작권보호법은 표현 형태가 동일하거나 유사한 경우에만 불법 복제라고 판단한다. 즉, 소스코드가 형태가 다르지만 동일한 동작을 하는 경우에는 불법 복제라고 판단하지 않는 것이다. 도용자는 소스코드를 도용할 때 형태만 조금 바꾸면 되는 것이다. 최근에 여러 판례에 의해 그 해석 방법이 다소 현실화 되었지만 현재는 저작권 보호법 이외에 영업비밀침해와 같은 간접적인 법으로 처벌하고 있는 현실이다. 소스코드의 특수성을 고려한 법안이 절실한 형편이다.
- **소스코드 불법도용을 막기 위한 체계적인 시스템 구축 필요:** 소스코드 불법 도용은 외부에서 침투하여 복제해 가는 것이 아니라 대부분의 경우 내부자에 의해 복제가 이루어진다. 예를 들어 A 회사에서 근무하던 개발자들이 B 회사로 이직하면서 A 회사에서 개발한 소스코드를 가지고 가는 것이다. 대부분의 소프트웨어 개발회사들이 벤처 또는 중소기업으로 소스코드 불법도용에 대한 인식이 부족하여, 소스코드 도용을 방지하기 위한 체계적인 안전장치를 마련하고 있지 않다. 이에 대한 홍보와 함께 안전장치 구축에 관심을 가질 때이다.
- **소스코드 난독화(code obfuscation) 기술[1] 개발:** 소스코드 도용을 예방하기 위해 소스코드를 읽기 어렵게 만드는 기술이다. 개인/기업/국가의 생존권이 걸린 중대한 핵심 알고리즘, 보안 알고리즘, 기밀정보 등 유출 자체가 문제가 되는 경우에 적용할 필요가 있다. 예를 들어, 아래 그림에 비유한 것처럼, 보물지도를 가지고 있는데 다른 사람이 보더라도 이해하지 못하도록 지도를 퍼즐로 만들고 뒤섞어서, 다른 사람이 이 보물지도를 구하더라도 섞은 순서를 알지 못하면 지도를 이해할 수 없도록 하는 방법이다. 난독처리는 코드를 변환하여 최대한 복잡하고 혼란스럽게 만들어서 공격자가 역공학(reverse engineering)을 수행하기 힘들게 하고 역공학이 되더라도 소스를 읽기 어렵게 함으로써 소프트웨어의 보안 수준을 향상시킨다.

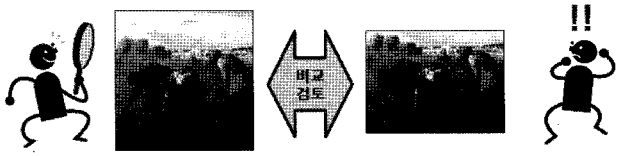


- **소스코드 저작권 삽입 및 탐지 기술:** 좀 더 민감하지 않은 경우를 위한 것으로, 소스코드에 저작권을 탑재하여 추후 도용당하더라도 소스코드에서 저작권을 추출하여 도용을 증빙함으로써 손해 배상을 받을 수 있도록 하는 방법이다. 예를 들어, 아래 그림에 비유한 것처럼, 그림에 자신의 싸인을 하거나 낙관을 찍음으로써 도난당하더라도 그 그림이 자신의 것임을 증명할 수 있도록 하는 것이다. 이러한 기술을 소프트웨어 워터마킹(software watermarking)[2]이라고 한다. 소프트웨어의 소스 코드 혹은 바이너리 코드에 저작권 정보 등을 삽입하는 기술로서 소프트웨어의 불법적인 사용과 유통을 막을 수 있고, 더 나아가 콘텐츠에 대한 정당한 대가를 보장 받을 수 있다. 소프트웨어 워터마킹은 소프트웨어에 대한 복사를 방지하기 위한 것이 아니라, 데이터 구조나 알고리즘 같은 소프트웨어의 소유권을 증명하기 위한 것이며, 추후 코드 도용에 관한

분쟁이 발생했을 경우, 개발자는 포함되어 있는 워터마크를 이용하여 프로그램이나 모듈에 대한 저작권을 주장할 수 있다.



- **소스코드 복제 감지 기술**[3]: 앞서의 소스코드 도용 방지기술을 사용하지 못하고 또한 저작권 탑재도 못한 채 소스코드가 도용당했을 경우에 필요한 기술이다. 이 경우에는 소스코드 자체를 제외하고는 도용 사실을 증빙하기 어렵다. 아래 그림에 비유한 것처럼, 그림에 도장을 찍지 못하고 도난당했을 때 또는 도장을 찍었는데도 불구하고 도용자가 도장을 제거했을 경우이다. 방법은 원본 그림과 같은 그림이 있는 경우 그림에 사용된 재료, 스타일, 그림에 나오는 개체들, 그 형태의 유사성 등을 근거로 그림이 도난당했음을 증빙하는 것이다. 소스코드에 대해서도 이러한 유사도 감정 기술을 사용하여 도용을 증빙할 수 있다.



이러한 소프트웨어 및 소스코드에 도용에 대한 법적/제도적 장치 및 관련 기술이 발달하면 불법 복제가 줄어들어 소프트웨어 가치 향상에 큰 도움이 되리라고 믿는다.

3. 소프트웨어 품질 지표와 측정 도구

소프트웨어 및 소스코드의 저작권이 보호된다고 해도, 정작 알맹이인 내용이 부실하다면 소프트웨어가 가치 있는 산출물이라고 사회적이라고 인식되기 어렵다. 예를 들어 건물에 대한 소유권 등에 대한 법률이 잘 정비되어 있다고 하더라도, 건축기술이 부족하여 부실 공사가 만연하게 된다면 건축학에 대한 위상이 높아질 수가 없다.

소프트웨어의 내용이 알차기 위해서는 소프트웨어 품질에 대한 특별한 관심이 필요하다. 같은 목적의 건물을 짓기 위해 공사를 하더라도 건축 기술자와 비전문가가 지은 건물에는 분명 큰 차이가 있다. 예를 들면, 지진이 발생하였을 때 기술자가 지은 건물은 튼튼하게 버틴 반면 비전문가가 지은 건물은 무너질 가능성이 높다. 건축에서는 이러한 건물의 품질에 대해 다각도로 평가를 한다. 예를 들면, 설계도면을 검수받고 건물이 완공되었을 때 설계도면대로 지어졌는지 검사 과정을 거쳐 건물의 품질을 평가한다. 하지만 소프트웨어의 품질에 대해서는 현재 법적 장치와 기술이 부족하다.

소프트웨어 품질에 대한 인식 부족으로 전문 프로그래머가 대우 받을 수 없는 것이다. 전문 프로그래머가 작성한 오류 없이 제대로 동작하는 소프트웨어와 초급 프로그래머가 작성한 비정기적으로 비정상 종료하는 프로그램을 똑같이 취급한다고 가정하자. 즉, 두 프로그래머가 같은 대우를 받는다고 하자. 누가 전문 프로그래머가 되려고 할까? 누가 전산학의 위상을 높게 평가할까?

소프트웨어 품질에 대한 지표를 제시하고 이를 측정하는 도구를 개발하는 것이 필요하다. 제대로 된 품질 측정 도구와 지표가 없으면 품질 좋은 소프트웨어와 품질 나쁜 소프트웨어를 구분할 방법이 없는 것이다.

- **소프트웨어 품질에 대한 제도적 표준 마련 필요:** 많은 소프트웨어 공학 연구자들이 소프트웨어의 품질에 관하여 연구하여 왔고 ISO 9126과 같은 표준도 존재한다. 하지만 이러한 표준안들이 현장에서 적용되는 사례가 많지 않다. 현실을 고려하여 측정가능한 요소들로 소프트웨어 품질에 대한 지표를 마련하고 이를 운용하여야 한다. 예를 들면 정부 조달 소프트웨어는 품질 등급 얼마 이상만 납품가능하다와 같은 강제 조항을 두어 품질에 따라 소프트웨어를 차별할 필요가 있다.
- **소프트웨어 품질 검사 기술 개발:** 앞서 언급한 소프트웨어 품질에 대한 지표가 잘 운용되려면 이를 객관적으로 계측할 수 있는 도구가 반드시 필요하다. 최근의 프로그래밍 언어 분야에서 현장에 적용할 수 있을 만큼 발달된 정적 분석(static analysis) 기술[4]과 동적 분석(dynamic analysis) 기술을 조합하여 품질을 측정할 수 있는 도구를 개발하는 것이 필요하다. 정적 분석 기술은 프로그램을 실행시켜 보지 않고 소스코드만을 보고 실행가능성을 예측하여 추정하는 기술을 말한다. 예를 들면 특정 오류가 발생할 수 있는지 가능성을 계산하여 탐지하는 기술이다. 정적 분석 도구의 장점은 모든 가능성을 계산한다는데에 있고 단점은 정확도가 상황에 따라 떨어질 수 있다는 데에 있다. 예전에는 그 계산량이 많아 현실적인 도구로 구현되지 못하였는데 최

근에 그 구현기술이 발달하여 상업용 소프트웨어[3]로 판매되고 있을 정도로 소프트웨어 품질 측정에 활용하기 충분한 상태가 되었다. 동적 분석 기술은 테스트의 한 종류로 프로그램을 실행하면서 관심 있는 정보를 모니터링하여 프로그램을 테스트하는 것이다. 정적 분석이 계산하기 어려운 정보를 실제 실행을 통해 관찰하는 기술이다. 이 두 기술을 조합하면 충분히 소프트웨어의 품질 측정을 위한 유용한 도구 개발이 가능하리라 믿는다.

이러한 품질 지표와 측정도구가 갖추어지면 소프트웨어를 품질에 따라 등급을 부여할 수 있고 결과적으로 프로그램 개발 인력의 옥석을 가릴 수가 있다. 특정 제품의 가치를 높이는 일은 산출물의 품질 제어가 있다고 판단된다. 예를 들어 한우도 등급을 부여하면서 그 품질을 높이려는 노력이 수반되고 결과적으로 고부가가치를 창출할 수 있는 특등급 한우를 생산할 수 있듯이, 소프트웨어 또한 객관적인 등급 부여를 통해 그 가치를 높일 수 있으리라 믿는다.

4. 품질 좋은 소프트웨어 개발 능력을 가진 인력 양성

학생들을 상담하다 보면 가끔 방학을 이용하여 학원에서 프로그래밍 언어 또는 기술을 익히는 것이 어떤지 자문을 요청하는 경우가 있다. 그러면 대학에서 교육을 하는 교수로써 학원에서 가르치는 내용과 대학에서 가르치는 내용이 어떻게 다른지, 어떻게 달라야 하는지에 대해 자문하게 된다. 이에 대한 필자의 대답은, 학교는 기본을 가르쳐야 하고 학원은 현재를 가르쳐야 한다. 프로그래밍 언어를 예로 들면, 학교에서는 프로그래밍 언어가 갖추어야 하는 구성 요소와 각 구성 요소의 기능들에 대해 논해보고, 현재까지 널리 활용된 프로그래밍 언어를 살펴보고, 더 좋은 프로그래밍 언어를 어떻게 설계할 수 있는지에 대해서 교육한다. 하지만 학원은 현재 널리 쓰이고 있는 프로그래밍 언어의 구체적인 내용들, 즉 당장 특정 기능을 구현하려면 어떤 라이브러리를 사용해야 하는지를 학습한다. 그러면 어느 교육이 학생에게 더 이득이 되는 것인가? 단기적으로는 학원에서 배운 것이 더 쓸모 있을지 몰라도 장기적으로는 학교에서 배우는 기초가 필요한 것이다.

기초가 없으면 품질 좋은 소프트웨어 설계 및 구현은 어렵다고 믿는다. 기초가 왜 필요한가? 어떤 운동을 배우더라도 기초가 튼튼해야 한다고 한다. 처음 폼을 잘못 잡으면 어느 정도 이상 능력을 향상시킬 수가 없다고 한다. 현재의 프로그래밍 언어 습득에 급급하다가는 당장은 프로그래밍을 잘 할 것처럼 보이지만, 새로운 개념이 나오면 전혀 이해를 하지 못하여 적응할 수 없다. 예를 들어, 운영체제를 배운 사람과 운영체제를 전혀 배우지 못한 사람에게 인터럽트 및 이벤트 등을 설명하여 이해시키는 것은 커다란 차이가 있다.

필자의 짧은 소견으로는, 대학 교육에서는 당장 필요한 기술을 교육하기 보다는[4] 품질 좋은 소프트웨어를 설계·구현할 수 있도록 기초 교육을 강화해야 한다고 본다. 즉, 전통적인 전산학 교육과정이 그 해결책이라고 판단된다. 최근의 대학의 교육과정을 살펴보면 현재 추세를 지나치게 반영하여 중심 없이 교과과정을 운영하는 경향을 관찰할 수 있다. 멀티미디어가 유행을 타면 멀티미디어 프로그래밍 관련 교육, 내장형 시스템이 유행을 타면 그 관련 교육, 유비쿼터스 시스템이 유행을 타면 그 교육으로 교과과정을 마구 뒤흔들어 기초에 상관없이 교육하는 경우를 많이 볼 수 있다. 필자의 판단으로는 그러한 시대의 흐름은 튼튼한 기초 위에 반영이 되어야지, 기초 없이 그 응용분야들을 학생들이 이해할 것이라고 판단되지 않는다.

이러한 기초지식 바탕위에 추가적으로 실무적인 교육을 해야 한다면 소프트웨어 공학(software engineering) 관련 지식을 교육해야 한다고 믿는다.[5] 품질 좋은 대규모 소프트웨어 개발에 있어 필수적이라고 판단되기 때문이다.

- **소프트웨어 개발 프로세스에 대한 교육 필요:** 목표가 주어졌을 때 주먹구구식으로 프로그램을 개발하는 것이 아니라, 설계와 분석을 거쳐 구현으로 이

3) 국내외에 최근 소프트웨어 오류 또는 보안취약점 감지 상용 도구가 많이 개발되고 있다. Sparrow, Fortify, Coverity 등이 있다.

4) 최근 산업 현장에 바로 적용 가능한 실무교육을 대학교육의 방향으로 설정하는 경향이 있는데, 필자는 그 방향이 옳지 않다고 본다. 물론, 대학에서 4년간 배운 인력이 당장 현장에서 (그 대상 프로그래밍 환경에 대한 경험이 없기 때문에) 프로그래밍을 할 수 없다는 불만에도 일리가 있다. 하지만 대학은 배출하는 인력의 장기적인 능력개발에 그 초점을 두어야 한다. 당장은 적응하더라도 환경이 바뀌면 적응하지 못하는 인력은 자연스럽게 퇴출될 것이다. 적응력을 높이기 위해서는 기초교육이 필수적이다. 이는 필자의 사견이 아니라 실제 현장 개발 책임자들이 기초만 제대로 교육해 달라는 요청을 하는 경우를 종종 보았다.

5) 물론, 각 대학의 교육목표에 따라 어떠한 내용을 실무교육으로 강조할 지는 다를 수 있다. IEEE CS와 ACM에서 제안하는 교과과정안(http://www.acm.org/education/curricula-recommendations)에 따르면 전산학의 교과과정을 크게 5가지로 분류하고 있다. 전통적인 전산(CS), 컴퓨터공학(CE), 소프트웨어공학(SE), 정보공학(IE), 정보시스템(IS)으로 분류하고 있다.

루어지고, 또는 전체 개발 프로세스가 순환적으로 작동되는 등, 체계적인 소프트웨어 개발에 대한 교육이 필요하다. 체계적인 개발 프로세스가 없이는 품질 좋은 소프트웨어 개발은 불가능하다.

- **소프트웨어 품질 관리에 대한 교육 필요:** 프로그램을 작성할 때 단지 구현 편의성이나 성능만을 고려하는 것이 아니라, 품질에 대해 고려하면서 작성하는 방법에 대한 교육이 필요하다. 오류 유발 가능성, 위험한 구문 사용 방지, 예외 처리의 필요성, 특정 부품이 만족해야 하는 명세 등을 고려하면서 프로그래밍을 하여야만 결함이 없는 소프트웨어 개발이 가능하다. 또한 개발된 제품에 대해 품질 분석 및 평가 방법을 익혀 더 질좋은 소프트웨어 개발이 가능하도록 교육이 필요하다.

5. 결 론

필자의 사견으로는 현재의 이공계 기피나 전산학 기피는 단기적인 시대의 흐름이라고 본다. 최근에 IT 거품으로 인하여 전산학이 지나치게 높은 가치를 부여 받았던 것의 일종의 반작용이라고 판단된다. 기술의 흐름 상 소프트웨어가 전 산업에 미치는 파급효과가 점점 더 커질 수밖에 없기 때문에 이러한 단기적 경향은 사라지리라고 믿는다. 하지만 소프트웨어의 사회적 영향력이 커질수록 이러한 단기적인 위상 저하도 우려스럽다.

학계에서는 이를 단순히 무시할 것이 아니라 현재의 우리의 모습을 비판하고 개선하는데 관심을 기울이는 것이 옳다고 본다. 언제나 역사에서는 일신우일신(日新又日新)하지 않은 국가나 조직은 쇠락하기 마련이었다. 지금이 우리의 모습을 보면서 반성하고 우리를 개조할 수 있는 좋은 기회를 맞이한 것이다. 필자가 판단컨대 소프트웨어의 가치와 품질을 높이는 일이 바로 지금 해야 하는 일인 것이다. 법적/제도적 장치, 필요한 기술의 개발, 그리고 튼튼한 교육과정 수립을 통하여 질좋은 인력을 양성하도록 노력하는 것이 필자가 생각하는 우리가 할 일이라고 판단된다.

참고문헌

[1] C. Collberg, C. Thomborson, and D. Low, A taxonomy of obfuscating transformations. Technical Report 148, Dept. of Computer Science, University of Auckland, New Zealand, July 1997.

[2] C. Collberg, and C. Thomborson. Software watermarking: models and dynamic embeddings. *ACM Symposium on Principles of Programming Languages*, pp. 311-324, March 1999.

[3] K. Kontogiannis, R. D. Mori, E. Merlo, M. Galler, and M. Bernstein. Pattern matching for clone and concept detection. *Automated Software Engineering*, 3(1/2):79-108, 1996.

[4] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*, Springer, 1999.

이 욱 세

2004 교수, 한양대학교 컴퓨터공학과
2003~2004 박사후연구원, 서울대학교 컴퓨터공학부
1997~2003 박사, KAIST 전산학과
1995~1997 석사, KAIST 전산학과
1991~1995 학사, KAIST 전산학과
관심분야 : 프로그램 분석, 소스코드 유사도 감정, 컴파일러, 함수 언어
E-mail : oukseh@hanyang.ac.kr