중소기업정보기술융합학회논문지 제1권 제1호 pp.19~28, 2011

융합 소프트웨어 품질의 특성, 개선 전략과 대안

민상윤, 박승훈, 이남희
(주)솔루션링크

SW Quality of Convergence Product: Characteristics, Improvement Strategies and Alternatives

Sang-Yoon Min, Seung-Hoon Park and Nam-Hee Lee
SOLUTIONLINK Co.

요 약 현시대 및 미래의 제조산업은 대부분 소프트웨어 융합 제품으로 이미 바뀌었거나 계속해서 바뀌고 있어, 소프트웨어가 부품의 의미로 사용되었던 '임베디드 소프트웨어 제품'이라는 용어도 이미 소프트웨어와 하드웨어가 동등한 역할을 하는 '소프트웨어 융합 제품'으로 불리고 있다. 이것은 단지 호칭의 유행이 아니라 제품의 기능적 진화의 대부분이 소프트웨어를 통해 중점적으로 진행되고 있음을 의미하며, 또한 소프트웨어의 품질이 융합 제품의 품질을 좌우하게 되었음을 의미한다. 융합 제품은 특성상 양산과 판매의 성격을 지닌다. 이는 양산된 제품의 소프트웨어 결함은 양산된 제품 수에 비례하여 엄청난 결함 비용을 발생한다는 것이다. 따라서 융합 산업에서 소프트웨어 품질은 기능의 경쟁력이기 이전에 사업의 안정성을 의미한다. 소프트웨어 품질 개선에 대해서는 그 동안 많은 연구와 노력이 있어왔으나, 국내나 아시아의 개발환경 하에서 실질적인 성공사례를 찾아보기는 쉽지가 않다. 아이폰의 충격과 토요타의 리콜 사건 등 많은 융합 산업의 물결 속에서 보다 현실적이고 냉정한 소프트웨어품질 관리 방안이 필요한 때이다. 본 논문에서는 융합 소프트웨어 산업의 태생적 특성을 살펴보고, 동시에 전통적인 소프트웨어 품질 개선 방법들의 현재 소프트웨어 융합산업에서의 적용 한계에 대해 논하여 본다. 그리고 융합 소프트웨어 산업의 특성을 고려한 보다 현실적인 소프트웨어 품질 개선 방안에 대해 소개하고자 한다. 본 논문에서 기술되는 전략은 프로세스 수준보다 개선 활동의 구현상에서의 구체적 전략에 해당한다. 따라서 CMMI혹은 TQM, 6시스마와 같은 품질 혁신활동과 어우러져 사용될 수 있다.

키워드 : 소프트웨어 품질, IT 융합, 보안 결함, 소프트웨어 공학

Abstract In today and near future, most of the commercially manufactured IT products will be evolved into software convergence product. Recently, the embedded software products is called as 'Software Convergence Products.' This phenomenon does not simply show the trendy fashion, but has the seriously implication that the functionalities of IT product is accomplished and evolved via software technology , not via mechanical nor electrical means. It will become true that the quality of the convergence product is dominantly governed by the software it uses. Meanwhile, we are facing with the threatening fact that software defects in the mass products will requires tremendous amounts of cost proportional to the quantity of the product. We can remind ourselves of the disasters that have been already happened, such as Automotive recalls, Smart-phone recalls, and others. In software engineering, there have been large amount of work done in software quality improvement for the past couple of decades. Software process improvement, and testings are the representative ones. But we are facing with limitations of those traditional approaches in current convergence industry; exponentially increasing software sizes and rapid changes in software technology. In this paper, we analyze the characteristics of the software convergence industry, the limitations of the traditional Software quality improvement approaches. We suggest a new approaches in software quality improvement in different angles of thought and philosophy.

Key Words : Software quality, IT convergence, Defect prevention, SW engineering

*교신저자(snag@sol-link.com)
접수일(2011년 7월 28일), 심사완료일(2011년 10월 10일)

1. 서론

가전, 자동차를 포함한 현시대 대부분의 제조산업은 많은 경우 소프트웨어를 포함한 융합 제품으로서 이미 제작되거나 혹은 전이되고 있는 상태이다.

소위 우리들이 일컫는 '임베디드 소프트웨어 제품'은 이미 예전의 Firmware 수준의 소프트웨어를 넘어서 거대한 사이즈 (수십만에서 억대의 프로그램 소스코드 라인수를 가진)의 '소프트웨어 융합' 제품으로 이미 제작되고 있다.

십년 전만해도 기껏해야 수십만 라인 정도의 C 언어로 된 프로그램이면 충분하던 핸드폰이, 스마트폰 시대로 접어들면서 천만라인을 넘어선 상황이다. 단순히 사이즈와 복잡도 측면에서만 보더라도 대형 엔터프라이즈 소프트웨어 시스템에 버금가거나 그 이상의 수준이 되었다. 이는 다시 말해 대부분의 소프트웨어 융합 제품군들의 기능적 진화는 더 이상 hardware(이하 하드웨어)적인 변화를 통해 만들어지는 것이 아니라 소프트웨어를 통해 이루어진다는 것을 의미한다. 즉, 하드웨어 기술은 기본이고 소프트웨어 기술로 제품의 경쟁력이 좌우되고 있다. 가장 대표적인 소프트웨어 융합 산업이 바로 자동차 산업이다.

자동차 부품사 중 전세계의 대표적인 기업으로 Bosch사를 예로 들 수 있다. Bosch사는 이미 소프트웨어의 중요성과 이에 대한 R&D가 90년대 초반부터 본격적으로 이루어지고 있으며, 그림 1과 같은 소프트웨어 중심의 철학 하에 Bosch의 제품은 소프트웨어 제품이라는 것에 큰 전략을 두고 있다[1].

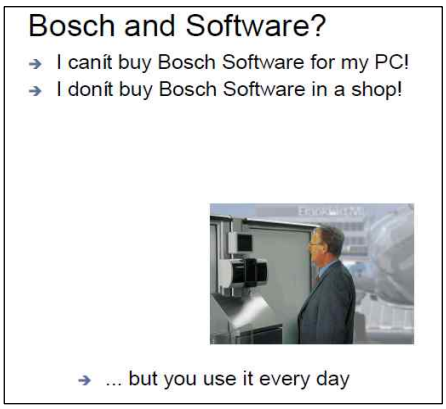

그림 1. Bosch사의 슬로건
Fig 1. Bosch's slogan

소프트웨어 융합 제품은 제품이라는 특성상 양산과 판매라는 유전적 성격을 가지게 된다. 양산 (Mass Production)은 말 그대로 수십만 대에서 수천만 대의 동일 제품이 제조되고 판매되는 것을 의미한다. 양산은 이와 같이 매출증대 측면에서 거대한 매력을 갖는 특성이 있는 반면에, 제품 불량 발생 시 엄청난 비용을 지불해야 한다는 위협적인 특성도 존재한다. 소프트웨어 융합 제품에서 결함 발생은 일반 컴퓨터 어플리케이션 소프트웨어와 같이 버그 fixing & patching 비용으로 끝나는 것이 아니라, 극단적인 경우에는 출시된 제품에 대한 전체 리콜까지도 갈 수 있다. 더구나 융합 제품군의 특성에 따라서는 자동차, 항공기, 의료기와 같이 안전에 위협을 줄 수 있는 경우에는 사고 비용까지 지불하게 되면 막대한 손실로 이어져 회사의 존패에 대한 위협으로 작용할 수도 있다.

소프트웨어 품질 개선에 대해서는 그 동안 많은 연구와 노력이 있어왔다. 그러나 실제 필드에서 많은 실질적인 성공사례를, 특히 국내와 아시아에서는 찾아보기가 쉽지 않다. 특히, 태생적으로 하드웨어 기술로부터 발전한 국내 및 아시아권의 제조업 분야에서는 소프트웨어품질 개선에 대한 성공 사례를 찾기가 더욱 어렵다.

소프트웨어 융합을 중심으로한 제조업에서의 소프트웨어 품질 저하는 제품 개발 일정 지연과 이로 인한 고객 신뢰도 하락, 제품 개발에 대한 직,간접적 실패비용에 대한 부담, 양산 후 결함 발생 시 엄청난 보상 비용 발생과 이로 인한 회사의 경쟁력 상실 등 직접 혹은 간접적인 치명적 피해를 유발하게 된다.

본 논문에서는 소프트웨어 융합 산업의 태생적 특성을 살펴보고, 동시에 전통적인 소프트웨어 품질 개선 방법들의 현재 융합 소프트웨어 산업에서의 적용 한계에 대해 논하여 본다. 그리고 소프트웨어 융합 산업의 특성을 고려한 보다 현실적인 소프트웨어 품질 개선 방안에 대하여 소개하고자 한다. 먼저 2장에서는 제조업 관점에서 소프트웨어의 특징에 대해 살펴보기로 한다. 3장에서는 이러한 특징들을 기반으로, 어떻게 하면 보다 현실적인 소프트웨어 품질 향상을 소프트웨어 융합 산업에서 성취할 수 있는 가에 대한 요건들을 정리하여 본다. 4장에서는 이러한 성공 요건들을 기반으로 품질 개선 추진 전략과 방법론에 대해 제안하고자 한다. 마지막으로 5장에서는 본 접근에 대한 적용 기대효과를 정성적으로 요약하고 더불어 향후 연구 방향성에 대하여 논하여 보고

자 한다.

2. 소프트웨어 융합 제품에서의 소프트웨어 특성

본장에서는 통상적인 제조업에서 소프트웨어 융합 제품을 개발, 관리 시 반드시 올바르게 숙지하고 있어야 할 소프트웨어의 특성들에 대하여 살펴본다.

2.1 소프트웨어 프로젝트의 난이도

제조업 기반의 소프트웨어 융합 제품을 개발, 양산하는 기업들의 공통점 중의 하나가 소프트웨어 개발에 대하여 너무나 쉽게 생각하고 있다는 것이다. 실제로 많은 상위 관리자들은 소프트웨어 개발 경험이 없거나, 혹은 예전의 펌웨어 개발 경험을 기반으로 바라보고 있다 보니, 소프트웨어 개발자들과의 마찰이 많이 발생하고 있다. 국내 대표적인 가전 제조업체들이나 기타 소프트웨어융합 제품군들의 제조업체들의 실무자급 인력구성은 이미 소프트웨어 인력이 하드웨어 인력을 수적인 측면에서 넘어 선 상태이다. 인력의 양적 투자를 통한 성과 달성은 가장 기초적인 요건임과 동시에 가장 원시적인 방법이라는 양면성을 가진다. 가장 먼저 소프트웨어개발은 지식산업이라는 점을 이해하여야 하며, 오늘날의 소프트웨어는 그 복잡도나 사이즈 측면에서 이미 주먹구구식으로 관리하기에는 그 수준을 훨씬 넘어 섰다고 할 수 있다. 이는 비단 국내 뿐만이 아닌 해외에서도 큰 문제로 대두되고 있다. 아래 그림 2에서 보는 것과 같이 소프트웨어 프로젝트의 초과비용 발생률은 50%가 넘으며, 성공율은 30% 전후이며 2000년도 중반부터는 그 성공률이 지속적으로 하락하고 있다[2,3].

2.2 하드웨어 대비 소프트웨어 구성비

융합 제품은 말 그대로 물리적인 제품의 형태로 개발 및 양산이 이루어지기 때문에 가시적 관점에서 그 특성이 인지되는 경우가 많다. 특히 구성 부품에 대한 비율적인 측면에서 보면 융합 제품의 소프트웨어는 Cross Compile되어 마이크로 컨트롤러의 메모리 속에 저장되기 때문에 하나의 부품으로만 인지되는 경우가 많다. 그러나 하드웨어 부품과의 올바른 구성비를 보고자 할 경우 소프트웨어는 프로그램의 구성 단위수로 비교되어야 공정할 것이다. 아래 그림 3은 몇 가지 대표 제품군들에 대한 하드웨어 대비 소프트웨어 부품/모듈 구성 수에 대한 비교이다.

그림 2. 소프트웨어 Project 성공률 통계
Fig 2. Success rate statistics of software project

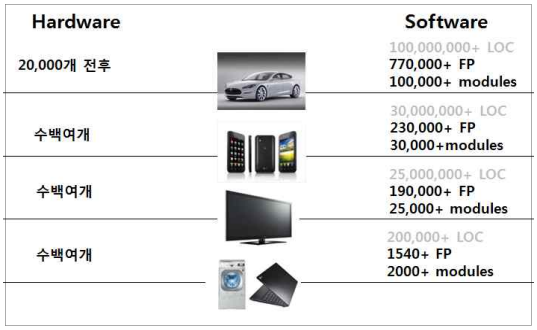

그림 3. 하드웨어/소프트웨어 부품/모듈 수 비교
Fig 3. Comparison between the number of hardware and software modules

그림 3의 부품/모듈 수에 대한 데이터는 필드인터뷰를 통한 데이터이며 제품 모델에 따라 몇 배 이내의 오차가 있을 수 있다. 몇 배의 오차가 중요하지 않은 것은 각 제품군에 대하여 하드웨어 부품 수 대비 소프트웨어 모듈

수를 비교하면 적게는 수십 배에서 많게는 수백 배에 이르기 때문이다. 소프트웨어의 크기를 정하는 단위는 LOC (Lines of Codes)부터 FP (Function Point: 소프트웨어 기능 단위로 C 언어의 경우 약 130 LOC 전후임), 그리고 Modules로 나누어 비교하였다.

상기 비교 데이터에서 유추하여 볼 수 있는 것은 결함이 단위 구성요소에서 발생한다고 할 경우, 소프트웨어 융합 제품의 경우 결함의 근원 자체의 수가 하드웨어보다 소프트웨어가 원천적으로 많다는 것이다. 이는 더 이상 융합 제품에서 소프트웨어가 단일 부품이 아니며, 하드웨어보다도 수적인 측면이나 양적인 측면에서 더 큰 비중을 차지한다는 것을 의미한다.

2.3 품질 안정성

품질의 안정성 (Stability)측면에서 소프트웨어는 하드웨어와 비교하여 볼 때 많은 차이를 가지고 있다. 하드웨어 백그라운드를 가진 많은 관리자들이 소프트웨어 품질 문제가 발생할 경우 품질의 안정성 측면에서 불만을 토로하는 경우가 빈번하다. 이는 생태계적이고 유전적인 차이가 있기 때문이다.

2.3.1 하드웨어 품질의 안정성

하드웨어 산업은 소프트웨어 산업에 비해 훨씬 오래된 역사를 가지고 있다. 따라서 부품의 표준화 및 규격화가 성숙되어 있으며 이에 따라 단일 부품에 대해서도 다양한 품질 수준의 부품들이 존재한다.

또한 부품 개발업체들의 해당 부품에 대한 전문성이 지속적으로 축적됨으로 인해 부품의 기술적, 품질적 안정성이 높다. 시간이 갈수록 부품업체들의 기술과 품질 수준은 증가되는 성격을 가진다. 또한 규격화를 통해 부품의 납품 전 품질 검증이 납품업체 자체적으로도 충분히 이루어질 수 있다. 산업 생태계적으로 바라보았을 때 상대적으로 성숙되고 안정적인 품질 체계가 운영되고 있다고 볼 수 있다.

2.3.2 소프트웨어 품질의 안정성

이에 반해, 소프트웨어는 부품화에 대한 연구 자체도 아직까지 완전히 성숙되어 있지 않은 단계이다. 말 그대로 소프트웨어기 때문에, 하드웨어와 같이 규격화 시키는 것 자체가 어쩌면 그 유전적 장점에 반하는 것일 수도 있다. 이러한 특성으로 인해 부품화, 표준화, 규격화가 하드웨어 비해 훨씬 불안정 하다.

이런 특성의 또 다른 원인은 하드웨어 대비 훨씬 급속한 기술 진화에 있다. 표준화가 되기 전에 새로운 기술이 등장하고 수개월 사이로 끊임없이 발전한다. 이러한 특성은 소프트웨어 아웃소싱 업체들의 지속적인 지식 축적과 품질 안정화에 부정적인 요소로 작용한다. 한마디로 소프트웨어는 하드웨어 대비 훨씬 불안정한 품질을 태생적으로 생태적으로 가지고 있다는 것을 인식하여야 한다.

2.4 변경 특성

제품을 개발하다 보면 개발 초기 단계 및 중간 단계, 심지어는 양산 단계에서도 많은 변경이 이루어진다. 첨단 제품, 신제품 일수록 더욱 그렇다. 제품개발에서 변경이 발생한다는 것은 피할 수 없는 요소이며, 소프트웨어 융합 제품에서는 최첨단 기술들과 기능들이 적용되기 때문에 더욱 그렇다.

기능에 대한 변경, 결함 제거를 위한 변경 등이 발생할 경우 시각적으로 인지되는 하드웨어 대비 소프트웨어의 intangible 한 특성때문에 많은 사람들이 소프트웨어는 변경하기 쉽다고 생각하게 된다.

2.4.1 하드웨어 변경 특성

융합 제품의 개발 시 하드웨어는 설계가 완료되고 부품이 확정된 후 제품의 조립에 들어가게 된다. 여기서 하드웨어란 마이컴뿐만이 아닌 기계적 하드웨어 (예, 조작버튼)도 포함한다. 하드웨어는 무게, 부피 공간, 부품 비용, 양산 체계 등이 다 연관되어 개발되는 특성을 가지기 때문에, 설계 및 부품 확정 후에 변경이 발생할 경우 반영하기가 쉽지 않다. 특히 양산에 진입하게 되면 거의 불가능하다고 볼 수 있다.

2.4.2 소프트웨어 변경 특성

하드웨어의 특성과는 다르게, 소프트웨어는 보이지 않는 Intangible한 특성과 언제든지 프로그램을 편집하여 재컴파일할 수 있다는 flexibility로 인하여 변경의 반영이 용이하다고 인식된다. 사실상 소프트웨어 적인 특성 때문에 변경을 가하는 것이 용이한 것은 사실이다. 그러나 안전한 변경, 품질을 해치지 않는 변경이 용이하다는 것은 보장되지 않는다. 변경을 가하기는 쉽지만, 변경을 잘하기는 어렵다는 성격을 가진다.

변경의 난이도는 소프트웨어의 사이즈와 복잡도가

증가함에 따라 기하급수적으로 증가하게 된다. 그러나 소프트웨어에 대한 선입관 때문에 융합 산업에서 많은 변경에 대한 반영이 소프트웨어를 통하여 이루어지고 있다. 양산 직전 단계, 혹은 양산 단계 중에서도 이러한 소프트웨어 변경이 빈번히 발생하지만, 소프트웨어에 대한 변경이 제대로 이루어지지 않아 결함 발생과 리콜로 이루어지는 경우도 많다.

이론 및 통계적으로 동일 변경내역에 대한 하드웨어적인 변경 대비 소프트웨어 적인 변경을 그 비용 측면에서 통계 분석하여 비교한 데이터는 아직 존재하지 않기 때문에, 어느 쪽이 더 비용적인 측면에서 유리하다고 판단할 수는 없다. 그러나 많은 변경이 하드웨어 보다는 소프트웨어를 통하여 이루어지고 있고, 소프트웨어 융합제품에서 변경은 피할 수 없는 사실이다.

2.5 구성 요소간 의존성

소프트웨어가 하드웨어 보다 변경하는 것이 결코 쉽지 않다는 특성을 나타내는 것 중의 하나가 바로 구성 요소간의 의존성이다. 하드웨어에 비하여 소프트웨어는 구성요소 간에 월등히 복잡한 의존 관계를 가진다. 레거시 코드일 수록 더욱 심하며, 모듈화가 안된 코드일수록 더욱 극심하다.

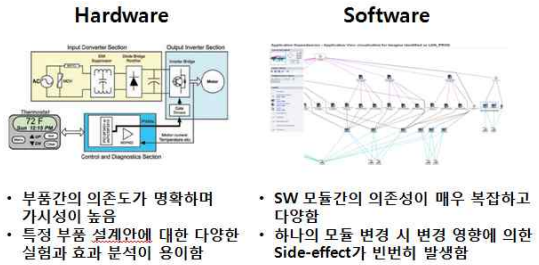

그림 4. HW와 SW의 구성요소 간 의존성
Fig 4. Dependency of HW and SW components

위의 그림 4에서 표현된 것과 같이 하드웨어는 회로적 특성, 기계적 특성상 구성 요소(부품)간의 상호 연관성과 의존성이 매우 명확하다. 또한 이러한 의존성을 파악하기 위한 가시성이 높다. 이러한 특성은 설계시 특정 부품에 대한 다양한 대안적 실험과 효과분석이 용이하다는 부가적 장점도 가지게 된다.

이에 반하여 소프트웨어 구성요소(소프트웨어 모듈)간의 연관성 및 의존성은 매우 복잡하다. 상기 그림에서와 같이 모듈간의 호출 관계, 데이터 의존도, 호출의 방식 등 소프트웨어 공학에서 분류하는 Coupling의 종류에 따라 분석할 경우 매우 복잡한 관계를 가지는 것이 일반적이다. 소위 역공학 (소스코드로 부터 설계정보를 역으로 분석하는 기법)을 통하여 수십만 라인 정도의 소스코드를 분석하여 보면 거미줄보다도 수백배는 복잡한 모듈간 관계도가 생성되는 것을 볼 수 있다.

이러한 모듈간의 복잡한 의존성은 특정 모듈에 대한 수정, 추가, 삭제와 같은 변경이 가해질 경우 예상치 못한 Side-effect가 빈번히 발생하게 한다. 더욱 위험한 상황은 수십만 라인 이상의 소스코드의 경우, 소스코드간의 관계를 모두 알고 있는 엔지니어는 거의 없다는 것이다. 각자 자기가 맡은 부분 중 본인이 파악하고 개발한 부분에 대해서만 알고 있으며, 더구나 현재 버전의 소스코드에 상응하는 설계도가 존재하는 경우는 거의 없다.

2.7 양산 개념

융합 제품에는 양산이라는 개념이 존재한다. 설계 개발된 제품에 대한 대량 생산이다. 이러한 양산의 개념은 하드웨어적 측면과 소프트웨어적인 측면이 매우 다르다. 특히 품질의 개선 시점에서 차이가 있다.

2.7.1 하드웨어 양산

하드웨어에서의 양산은 양산 계획과 프로세스에 따른 부품들의 물리적 조립을 의미한다. 이러한 특성은 조립하는 방법에 따라 품질이 차이가 발생할 수 있다는 것이다.

실제로 하드웨어는 양산 프로세스의 성숙도에 따라 제품의 품질에 많은 차이가 발생한다. ISO 9001이나 이를 기반으로 한 다양한 도메인으로의 제품 품질 표준들(예, TS 16949)에서는 양산에 대한 품질 요건을 엄격히 요구한다. 또한 품질 혁신 방법으로 세간의 큰 유행을 가지고 왔던 6시그마, Lean Process, TPS 등은 모두 양산 프로세스의 혁신에 많은 비중을 둔 방법론 들이다.

즉, 하드웨어는 부품의 품질이 제품 품질에 영향을 미치기도 하지만, 양산 프로세스와 양산 환경이 제품의 품질에 많은 영향을 미친다.

2.7.2 소프트웨어 양산

반면에 소프트웨어에서는 양산의 개념이 없다. 물론 양산기간동안 아무일도 하지 않는 것은 아니다. 앞에서

언급한 것 같이, 양산 단계에서도 변경 요구는 계속되는 경우가 많다. 물론 버그를 고치는 작업도 포함된다. 그러나, 하드웨어와 같이 제조라인에서 부품을 조립하여 완제품을 만드는 양산의 개념은 없다.

이미 양산단계 이전에 마이크로 컨트롤러로 소프트웨어가 크로스 컴파일되어 들어가게 된다. 엄격이 말해 양산이란 소프트웨어 측면에서는 복제활동에 불가하다. 따라서, 양산 프로세스의 성숙도에 따라 소프트웨어 품질은 별로 영향을 받지 않게 된다. 즉, 양산 전의 개발 단계에서 품질이 좌우된다.

또한, 동일 제품에 대한 하드웨어 양산의 경우에는 프로세스를 튜닝하면서 발생된 결함에 대한 재발 방지 활동을 반복적으로 수행 가능하지만, 소프트웨어는 그러한 활동이 불가능하다. 소프트웨어 개발 프로세스를 수십만 번 반복되는 양산 프로세스와 같이 반복할 수 없기 때문이다. 따라서, 양산 프로세스 개선을 위한 품질 혁신 방법론을 소프트웨어 품질 혁신에 적용 시 많은 부분이 논리적으로 부적합성을 가지게 된다.

3. 융합 제품에서의 소프트웨어 품질 개선 전략

2장에서는 융합 제품에서의 소프트웨어 특성을 살펴보았다. 3장에서는 앞에서 고찰된 소프트웨어 특성들을 기반으로 융합 제품 산업에서 소프트웨어의 품질을 어떻게 개선하여야 할 것인가에 대해 전략적인 측면에서 논하여 보고자 한다.

3.1 결함 특성을 고려한 필요사항

먼저 결함의 근원 요소가 하드웨어 대비 절대적으로 많다는 특성이 있다. 즉, 소프트웨어는 더 이상 단일 부품이 아니다. 또한, 소프트웨어의 intangible한 특성상 많은 변경이 요구되고 이를 대응 할 수 밖에 없는 것이 현실이다. 더구나, 소프트웨어 부품(모듈은) 간의 관계는 하드웨어 대비 매우 복잡하며, 변경 발생 시 이로 인한 예기치 못한 side-effect의 위험성이 크다. 이러한 특성을 고려하여 볼 때 다음과 같은 기술적 접근이 필요하다.

▷ **소프트웨어 결함의 잠재 근원에 대한 광범위한 접근**

▷ **소프트웨어 변경 시 모듈간 의존성에 대한 체계적인 분석**

3.2 품질 개선 특성을 고려한 필요사항

소프트웨어는 품질의 성숙도가 하드웨어에 비하여 생태계적, 유전적으로 훨씬 취약하며, 품질이 양산 단계 이전에 대부분 결정된다는 특성이 있다. 이러한 특성을 고려하여 볼 때 다음과 같은 프로세스적 접근이 필요하다.

▷ **개발 초기 단계부터 결함 발생에 대한 방지적, 예방적 접근**

▷ **개발 프로세스와 상호 긴밀한 조화**

3.3 전통적인 소프트웨어 품질 개선의 한계점

이러한 필요사항의 관점에서 일반 산업체에서 이루어지고 있는 대표적인 전통적인 소프트웨어 품질 개선 방법들이 한계점을 살펴보기로 한다.

테스팅 중심의 접근 : 많은 기업체에서 소프트웨어 품질 이슈가 떠오를 때 마다 가장 먼저 취하는 행동이 테스팅 활동 강화이다. 그러나, 테스팅은 사후 처리적 성격이 강하고 개발 팀과의 역할 분배 측면에서 복잡한 이슈들을 가지다 보니 아래와 같은 한계가 항시 발생한다.

- [비용] 테스팅 단계의 결함 발견 및 제거 비용은 개발과정내의 결함제거 비용 대비 수백 배에 달하는 비효율성을 가진다.
- [책임] 품질은 테스팅 전담팀의 책임이라는 문제에 대한 책임전가 현상이 일어난다.
- [시기] 문제의 발생 방지가 아닌 발생 후 발견 활동으로서 시기적으로 한계가 있다.
- [안정성: Stability] 테스터의 개별 능력에 따라 결함 발견률의 편차가 심하다.
- [문제 해결] 원인 분석을 통한 문제 해결을 위한 현상 재현이 쉽지 않다.

소프트웨어 프로세스 개선의 접근 : 2000년도 초반부터 국내에서는 CMMI, SPICE 등 많은 소프트웨어 프로세스 개선 활동을 해오고 있다. 여러 측면의 직간접적인 효과가 일어나고 있지만 아래와 같은 방법론 상의 피할 수 없는 한계가 존재한다.

- [기간] 프로세스 개선은 장기적인(수년이상) 품질 개선 활동이다. 또한 개선 효과에 대한 검증을 위해서는 장기적으로 많은 양의 데이터 축적이 필요하다.
- [추진방식] 프로젝트 내 문제 중심의 개선이라기보다, 조직 단위의 개선 접근이다. 따라서 개인별 혹은 팀 단위의 혁신 활동을 추진하기 어렵다.
- [중점 대상] 프로세스 수준의 개선 활동으로 주요 개선 대상이 프로세스이다. 프로세스 보다 하위의 기술적 접근은 공식적으로 범위 외에 있다. 그렇기 때문에 CMMI 같은 경우 SCAMPI같은 심사 방법론 상에서 산출물의 내용적 품질에 대해서는 평가하지 못하게 하고 있다.
- [추진 단계] 전형적으로 모델 대비 Gap analysis를 통한 프로세스 정립/보완 활동이 우선 수행된다. 따라서 실제 프로젝트 적용을 위해서는 최소 수개월 이상이 소요된다.

3.4 대안적인 접근 전략

위와 같은 전통적인 품질 개선 접근 전략의 한계점과 3.1절과 3.2절의 필요사항을 고려하여 볼 때 현실적으로 융합 제품에 적합한 접근 전략이 필요하다. 융합 제품의 소프트웨어적 특성과 전통적인 접근의 한계성 등을 고려하여 다음 그림 5과 같이 보다 현실적이고 문제해결/결함예방 중심의 접근이 필요하다.

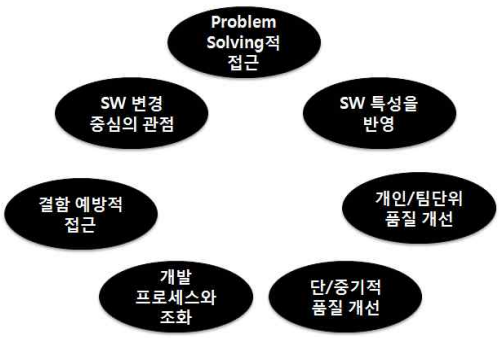

그림 5. 융합 제품의 SW 품질 개선 전략
Fig 5. Strategy for SW quality improvement of IT convergence product

4. 융합 제품의 소프트웨어 품질 개선 체계

4.1 소프트웨어 품질 개선활동 체계

본 논문에서는 앞선 융합 제품의 소프트웨어 품질 개선 전략에 따라 그림 6과 같이 소프트웨어 개발 과정에서 개인관점에서 수행할 수 있는 소프트웨어 품질 개선활동 체계를 제안한다. 융합제품의 주요 품질 특성들 중, 기존 재품의 파생 모델을 기반으로 한 개발 특성, 이로 인한 과거 결함 정보의 활용, 제품 Feature 중심의 요구사항 관리, 변경 중심의 개발 및 변경 영향 중심의 결함 관리 등이 주요 초점으로 고려되었다.

그림 6은 본 프로세스의 최상위 체계이다. 지적 보안상 세부 기술전략에 대한 설명은 한계가 있음을 밝혀둔다.

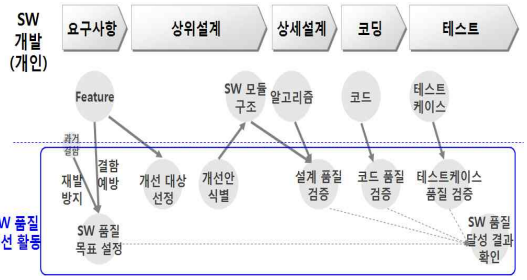

그림 6. 소프트웨어 품질 개선 활동 체계
Fig 6. Activities for software quality improvement

품질 개선활동 체계는 소프트웨어 품질 목표 설정, 개선 대상 선정, 개선안 식별, 설계/코드/테스트케이스 품질 검증, 소프트웨어 품질 달성 결과 확인의 7개 활동들이 소프트웨어 개발 활동과 연계되도록 구성되어 있다. 품질 개선활동 체계의 주요 특징은 다음과 같다.

- 변경 중심의 결함 예방 관점에서의 품질 활동

신규 융합 제품들이 등장할 때, 완전히 새로운 제품이 개발되는 경우는 희박하다. 대부분 기존 모델(베이스모델)에서 일부 기능이 변경(추가/수정/삭제)되어 새로운 제품으로 개발된다. 이러한 기능의 변경은 결함을 발생시킬 가능성이 높기 때문에, 신규 제품 개발 시 결함이 발생할 수 있는 영역을 식별하여 이러한 영역에 대한 집중관리를 하도록 해야 한다.

- 개인 단위의 단기적인 품질 활동 강화

품질 개선 대상 범위를 개인 단위로 한정하고, 개인 단위에서 수행 가능한 품질 개선 활동들로 구성함으로써 단기적인 품질 개선 및 개선성과 확인이 가능하도록 한다.

- 개발 프로세스 단계와의 연계

개발 과정에서 작성된 산출물들을 대상으로 다양한 품질 검증 활동을 수행하도록 함으로써 개발자들이 품질 활동을 부가적인 활동으로 인식하는 것을 최소화하도록 한다.

4.2 소프트웨어 품질 개선활동 소개

본 절에서는 4.1절에서 소개한 소프트웨어 품질 개선 활동 체계의 각 활동들에 대해 소개한다.

4.2.1 소프트웨어 품질 목표 설정

소프트웨어 개발 요구사항(Feature)이 식별되고 나면, 베이스모델과 비교하여 변경되는 요구사항에 따라 발생 가능한 결함을 예방하는 관점에서 달성하고자 하는 품질 목표를 설정한다. 품질 목표는 개인 단위로 설정한다. 품질 목표를 설정하기 위해서 품질 지표를 선정하고, 해당 품질 지표에 대해 달성하고자 하는 목표치를 정의한다. 품질 달성 가능 여부와 적정 수준의 품질 목표치를 선정하기 위해서 Benchmarking 등을 적용한다.

4.2.2 개선 대상 선정

품질 목표가 결정되면 중점적으로 개선해야 할 대상을 선정한다. 변경되는 요구사항에 따라 영향을 받는 소프트웨어 모듈들이 존재한다. 이러한 소프트웨어 모듈은 모두 변경에 따른 영향으로 인해 결함이 발생할 가능성이 있기 때문에, 품질 개선 대상이 될 수 있다. 품질 개선 대상에 포함될 소프트웨어 모듈은 변경되는 요구사항과 직/간접적으로 관련된 모든 소프트웨어 모듈을 포함한다. 이러한 소프트웨어 모듈을 모두 식별하기 위해서는 소프트웨어 모듈 간 의존성 분석이 이루어져야 한다. 이를 위해 Requirement Traceability Matrix, Design Structure Matrix와 같은 방법을 적용한다.

4.2.3 개선안 식별

개선 대상인 소프트웨어 모듈들이 식별되고 나면, 이러한 소프트웨어 모듈들에서 결함이 발생할 것으로 예상되는 부분을 개선하기 위한 개선안을 도출한다. 개선안은 소프트웨어 모듈을 어떻게 설계할 것인지에 대한 방향성을 제시하는 것으로, 구체적인 설계를 포함하는 것은 아니다. 개선안을 도출하는 부분은 단일 소프트웨어 모듈에 대해서도 가능하고, 소프트웨어 모듈 간의 관계에 대해서도 가능하다. 일반적으로 순환 참조가 발생하거나, 중복 또는 사용되지 않는 소프트웨어 모듈들이 해당될 수 있다. 개선안은 Refactoring이나 Design pattern과 같은 방법을 적용하여 도출하고, 이렇게 도출된 개선안을 이용하여 소프트웨어 개발 단계에서 구체적인 설계를 수행하도록 한다.

4.2.4 설계 품질 검증

소프트웨어 개발 단계에서 설계 활동이 완료되면 설계 내용에 대한 검증을 수행하도록 한다. 설계에 대한 검증은 동료 검토를 수행하도록 하며, 우선적으로 Inspection을 수행하도록 하되, 상황에 따라 Walkthrough와 같은 방법을 적용할 수도 있다. 설계 품질 검증을 필수적으로 수행하도록 함으로써 이후 코드 작성 시에 불필요한 결함이 발생하는 것을 최소화할 수 있다.

4.2.5 코드 품질 검증

소프트웨어 개발 단계에서 코드 작성이 완료되고 나면, 코드에 대한 품질을 검증한다. 코드 품질 검증은 정적 분석, 동료 검토의 두 과정을 모두 거친다. 대부분의 결함들이 실제 코드에 포함되기 때문에 코드에 대해서는 보다 엄격한 검증이 필요하다.

정적 분석은 일반적으로 자동화 도구를 이용해서 수행이 가능하며, 프로그램 실행 없이 대규모 코드에 대한 오류를 빠르게 발견할 수 있고, 사용자가 테스트하기 어려운 경로에서의 오류나 결함을 찾는 것이 가능하다는 장점이 있다.

하지만, 정적 분석을 수행하기 위해서는 적용 규칙 등이 필요하고 이러한 규칙의 수준에 따라 결과가 달라지므로 이를 보완해야 할 필요가 있다. 이를 위해서 코드에 대한 동료 검토를 수행함으로써 자동화 된 방법으로 인해 놓칠 수 있는 결함들을 추가적으로 발견할 수 있게 된다.

4.2.6 테스트 케이스 품질 검증

코딩이 완료된 후 테스트를 진행하기에 앞서 테스트 케이스가 수립된 품질 목표 달성 여부를 확인할 수 있는

정도로 준비가 되었는지에 대해 검증한다. 테스트 케이스가 변경되는 요구사항이나 관련된 코드를 충분히 커버하지 못할 경우 결함을 발견하지 못할 수 있기 때문이다. 작성한 테스트 케이스가 요구사항 (또는 개선안)에서 요구하는 기능을 모두 커버하지 못할 경우, Black-Box 기법을 적용하여 요구사항 (또는 개선안)을 기반으로 테스트 케이스를 추가로 확보하도록 한다.

또한 구현된 코드에 대한 충분한 테스트 케이스가 확보되었는지에 대한 검증은 Unit Test 수행 시 코드 커버리지를 분석하여 확인하도록 한다.

4.2.7 소프트웨어 품질 달성 결과 확인

품질 검증이 완료된 테스트 케이스를 이용하여 테스트를 수행한 후, 결함 검출 결과를 이용하여 목표 달성 여부를 확인하도록 한다. 개선 성과가 품질 목표를 달성하지 못한 경우 가설 검정과 같은 방법을 이용해서 개선 성과가 있다고 볼 수 있는지 판단한다.

5. 결론

융합 소프트웨어 제품은 몇 달 사이로 빠르게 진화한다. 아마도 소프트웨어의 진화 패턴과 싸이클이 과거 10여년전과 매우 다른 것 같이 보인다. 최근 들어 굴지의 글로벌 융합 소프트웨어 제품회사였던 노키아, 시스코, 모토롤라, 에릭슨 등의 사업 축소를 보면서 융합 소프트웨어의 생태계 변화를 직시하여 본다.

본 논문에서 언급한 것 같이 융합 소프트웨어 제품의 경쟁력은 내장된 소프트웨어의 기능과 품질에 좌우된다. 숙명적으로 반드시 해결하여야 할 문제이다. 생태계, 기술의 변화와 같이 소프트웨어 품질의 개선 전략 또한 진화가 필요한 시점이다. 본 논문에서는 기존의 전통적인 접근의 비효용성을 논하는 것이 아니라 그와 병렬적으로 현재의 환경에서 필요로 하는 소프트웨어 품질 개선 전략을 제시하였다. 본 논문에서 제시한 전략에 대한 실제 산업의 실 제품적용에 대한 결과에 따라 전략적 검증과 수정이 향후 주요 과제라고 판단된다.

참 고 문 헌

[1] Stefan Ferber, 'Architecture Reviews at Bosch' SEI 소프트웨어 Architecture Technology(SAT) User Network Conference, SEI, April, 2005

[2] *CHAOS Report*, Standish Group, 2004

[3] *CHAOS Report*, Standish Group, *2009*

[4] Thomas Pyzdek and Paul Keller, *The Six Sigma Handbook:3rd Edition,* McGraw-Hill, 2009

[5] Michael Kennedy, *Product Development for the Lean Enterprise: Why Toyota's System is Four Times More Productive and How You Can Implement It* , The Oaklea Press, 2010

[6] James M. Morgan and Jeffrey K. Liker, *The Toyota Product Development System: Integrating People, Process And Technology,* Productivity Press, Mar 2006

저 자 소 개

민 상 윤(Sang-Yoon Min) [정회원]

- 1993년 5월 : San Francisco State University, Computer Science(학사)
- 1997년 2월 : KAIST, 정보통신공학과, 소프트웨어공학 전공 (석사)
- 2003년 2월 : KAIST, 전산학과 소프트웨어공학 전공 (박사)
- 2004년 3월 ~ 현재 : KAIST 소프트웨어대학원 겸직교수
- 2008년 3월 ~ 2010.02 : 충북대학교 컴퓨터공학과 겸직교수
- 2000년 1월 ~ 현재 : (주)솔루션링크 대표이사

<관심분야> : 소프트웨어 공학, 융합 소프트웨어, 소프트 웨어 Globalization, IT Management

박 승 훈(Seung-Hoon Park)

- 2003년 2월 : 부산대학교 정보컴퓨터공학부(학사)
- 2005년 2월 : KAIST, 전산학과 소프트웨어공학 전공 (석사)
- 2011년 8월 : KAIST, 전산학과 소프트웨어공학 전공 (박사)
- 2010년 1월 ~ 현재 : (주)솔루션링크 책임 컨설턴트

<관심분야> : 소프트웨어 Quality, 융합 소프트웨어, Project Management

이 남 희(Nam-Hee Lee)

- 1991년 2월 : KAIST, 전산학과(학사)
- 1998년 2월 : KAIST, 전산학과(석사)
- 2003년 8월 : KAIST, 전산학과 소프트웨어공학 전공 (박사)
- 1994년 1월 ~ 1995년 7월 : LG전자 미디어통신연구소
- 2003년 9월 ~ 2005년 1월 : 삼성SDS 첨단소프트웨어공학센터
- 2005년 1월 ~ 2006년 4월 : 삼성전자 CS경영센터
- 20006년 6월 ~ 현재 : (주)솔루션링크 수석 컨설턴트

<관심분야> : 소프트웨어 Quality, 융합 소프트웨어, Project Management