정보기술응용연구
제 3 권 제 4 호
2 0 0 1 년 12 월

통계적 품질관리에 의한 소프트웨어 제품의 품질평가

류문찬* 임성택* 정상철** 이상덕*** 신석규***

요 약

소프트웨어 제품의 품질을 높이는데는 소프트웨어 개발프로세스 중심과 제품 중심의 2가지 접근방법이 있다. CMM, ISO 9000 패밀리나 ISO/IEC 12207, SPICE 등이 프로세스의 인증을 통해서 소프트웨어 품질을 향상시키려는 시도라고 할 수 있다. 그렇지만 '좋은' 프로세스만으로는 '좋은' 제품의 품질을 보장하기가 어렵다. 최근 독립적인 제3자에 의한 소프트웨어 제품의 품질을 평가하기 위한 필요가 점점 증가하고 있다. 본 연구에서는 소프트웨어 제품의 품질을 평가하는데 SQC 기법을 응용하여 평가프로세스의 효율을 기하고 평가결과의 객관성을 확보하는 방안을 다룬다. 랜덤추출법에 의한 테스트 케이스를 선정하는 방법을 소개하고 소프트웨어 제품의 품질에 대한 적합성 기준을 선정하는 방안을 제시한다.

*) 고려대학교 경영정보학과
**) 충남대학교 경영학과
***) 한국정보통신기술협회(TTA)

1. 서론

소프트웨어의 품질을 향상시키는 것은 사용자의 만족도를 높이고 한편으로 국가 경쟁력을 높이기 위한 필수적인 대안이다. 소프트웨어의 품질보증 접근방법은 크게 개발 프로세스 중심과 제품 중심으로 나눌 수 있다. 질 좋은 소프트웨어의 선결 조건이 질 좋은 개발 프로세스임은 두말할 나위가 없다. 그래서 ISO 9000 패밀리나 ISO/IEC 12207, CMM, CMMI, Bootstrap 혹은 Trillium이나 SPICE 등 소프트웨어의 개발 프로세스를 평가하고 이를 통한 개선을 도모하여 궁극적으로 소프트웨어 제품의 품질을 향상시키려는 많은 시도가 있어 왔다. 그렇지만 좋은 프로세스만으로는 소프트웨어의 품질을 보장할 수는 없다.

방위시스템이나 항공관제시스템과 같이 품질에 결함이 있으면 인명 피해와 직결되는 소프트웨어의 경우 개발프로세스에 대한 관심 못지 않게 소프트웨어의 제품의 품질에 대한 평가에도 많은 노력을 기울여야 한다. 이러한 경우가 개발자나 사용자가 아닌 독립적인 제3자에 의한 평가의 주요 대상이 될 수 있는데, 공공기관 등이 경우에 따라서는 법에 의해 독립적인 평가를 요구하기도 한다. 홈뱅킹, 전자지불시스템이나 공공행정 시스템 혹은 기업의 생산 시스템, 회계정보 시스템 등도 높은 수준의 품질을 요구하게 된다. 조직의 정보시스템에 대한 의존도가 높아지고 전략적 중요성이 커짐에 따라 소프트웨어 제품의 품질을 보증할 수 있는 방안을 다각도로 모색하고 있는 것이다. 이와 같이 제품 자체의 품질이 매우 중요할 때는 프로세스의 품질에 대한 고려 외에도 제품 자체의 품질을 보장할 수 있는 방법을 수립해야 한다. 다시 말해서 프로세스 중심의 관점과 제품 중심의 관점이 서로 보완되어야 한다는 것이다.

이 외에도 소프트웨어의 개발의 계약조건의 일부로 사용자가 소프트웨어 개발 회사에게 독립적인 품질평가를 요구하기도 한다. 그렇게 함으로써 소프트웨어 완제품의 전달과정에서 생길 수 있는 분쟁의 소지를 줄일 수 있다. 또한 품질인증 마크나 씰(seal)을 부여하기 위한 목적으로도 소프트웨어의 품질평가가 쓰이기도 한다. 이는 품질이 우수한 소프트웨어 제품에 대하여 시장에서 마케팅적 이점을 주자는 의도에서 나온 것이다. 또한 유사한 종류의 소프트웨어를 비교하기 위한 취지로 제품의 품질을 평가하기도 한다. 소프트웨어 관련 잡지 등에서 소프트웨어 패키지를 벤치마킹하기 위하여 이러한 방법을 쓰기도 한다. 이와 같이 제반 사회적 요구에 따라 독립된 제3자에 의한 소프트웨어 품질인증의 필요성이 높아지고 있는데, 소프트웨어 제품의 품질인증이란 품질 인증기관이 일정한 기준에 따라 소프트웨어의 품질에 대한 시험을 통해 소프트웨어 제품에 대하여 인증을 해주는 제도를 말한다.

제품 품질의 평가에 대한 필요성은 여러 가지 국제적인 표준화 활동에 반영되고 있으며 각국에서 다양한 형태로 품질평가 제도를 시행하고 있다. 전반적으로 유럽을 중심으로 성과를 높이고 있으며 남미나 아시아에서도 유사한 움직임

이 포착되고 있다. 독일의 GGS(독일 소프트웨어 산업협회)는 소프트웨어 제품에 대한 품질 씰을 처음 도입한 기관중의 하나이며, ESPRIT 프로젝트 SCOPE는 처음으로 소프트웨어 제품평가의 인증제도를 수립하기 위한 국제적인 시도라고 할 수 있다. 덴마크의 DELTA 소프트웨어공학에서는 소프트웨어 평가를 위하여 MicroScope의 접근방법을 채택하고 있는데 이것은 SCOPE의 결과를 기본으로 하여 ISO/IEC 9126과 ISO/IEC 14598-5를 따르고 있다. 또한 독일의 TÜV Nord는 안전성과 관련된 실시간 시스템의 프로세스 제어 프로그램을 평가하기 위한 방법을 개발하였다. 한편 브라질에서는 CTI가 소프트웨어 품질평가 서비스를 제공하는 주된 역할을 수행하고 있는데, 이들은 MicroScope와 유사한 체크리스트 위주의 평가방법을 채택하고 있다. 이 외에도 프랑스, 이태리, 아일랜드, 네덜란드 및 스웨덴에서도 유사한 활동을 전개하고 있으며 미국에서는 VeriTest나 NSTL등 주로 민간 인증기관에서 소프트웨어 제품의 품질인증 서비스를 수행하고 있다[4].

한편 한국의 경우 소프트웨어 산업진흥법에 의한 소프트웨어 품질인증제도가 한국정보통신기술협회(TTA)의 주도하에 시행되고 있으며, 품질인증은 ISO/IEC 14598[14, 15, 16, 17, 18, 19]과 ISO/IEC 12119[13]을 근간으로 시행하고 있다[2]. 대부분의 소프트웨어 개발업체들은 자체 인지도가 낮고 대외적인 신뢰도가 떨어지기 때문에 패키지 소프트웨어의 판로를 개척하는데 많은 어려움을 겪고 있다. 이러한 회사 제품에 대하여 공신력있는 독립적인 기관의 객관적인 인증을 통하여 제품에 대한 이미지를 높이고 신뢰성을 확보하여 이들이 겪고 있는 애로사항을 해결하자는 취지로 품질인증제도가 도입된 것이다. 물론 이를 통한 자발적인 품질개선 노력을 유도하여 소프트웨어의 품질을 향상시킴으로써 기업체의 기술력을 확보하여 수출경쟁력을 높이는 부수효과도 기대하고 있다.

본 연구에서는 소프트웨어 제품의 품질평가 과정에 통계적 품질관리기법을 활용하여 인증프로세스의 계량화를 통한 인증의 객관성과 신뢰성을 확보할 수 있는 근거를 제공할 수 있게 하고자 한다. 소프트웨어 개발과정에서의 통계적 기법의 활용은 제조 분야에 비해 많이 뒤떨어져 있으나, 최근에 와서 많은 연구가 나오고 있다. 류문찬[1]은 소프트웨어의 품질 향상을 위하여 TQC 이론을 활용할 수 있는 방안을 모색한 바 있다. Deming 싸이클은 Plan, Do, Check 및 Act의 순환적 싸이클을 반복함으로써 품질을 지속적으로 향상시키자는 개념인데 이를 기반으로 하여 소프트웨어 품질을 보증할 수 있는 방안이 제시되기도 하였다[3]. 세계적 수준의 기업은 이와 같은 통계적 품질관리이론을 소프트웨어 품질보증에 적용하기 위한 개념의 정립단계를 벗어나 개발과정에서 활발히 활용하는 수준이라고 할 수 있다[7, 26]. 개발과정에서는 SPC를 중심으로 활용하고 있으며, 통계적 기법을 활용한 소프트웨어의 신뢰도에 관한 연구도 활발히 진행되고 있으나 소프트웨어 제품의 품질평가에 대한 통계적 기법의 활용은 상대적으로 저조한 편이다[8, 20, 21, 22, 25]. 본 연구에서는 샘플링검사의 원리를 활용하여 소프트웨

어 제품의 품질 평가 결과에 대한 객관적인 근거를 제시하고자 한다. 2장에서는 테스트 데이터 그룹을 구성하는 방법을 제시하며, 테스트데이터의 시험 결과로부터 소프트웨어 제품의 품질에 대한 적합도 여부를 판정하는 방안은 3장에서 다룬다.

2. 테스트데이터 그룹

소프트웨어는 사용자의 요구사항에 따라 다양한 형태의 입력에 대한 처리 결과를 출력으로 내보내게 된다. 제조물을 생산하는 공장의 경우 원자재가 공장에 투입되어 일련의 가공과정을 거쳐 완제품으로 나오게 된다. 원자재를 완제품으로 변환하는 과정을 프로세스라고 하는데, 프로세스의 품질이 우수하면 그로부터 생산되는 완제품의 품질 또한 우수할 것이다. 프로세스 중심의 품질개선 철학은 이와 같은 생각을 바탕으로 하고 있다.

소프트웨어를 공장으로 비유한다면 투입되는 입력은 원자재로, 출력은 완제품에 비유된다. 소프트웨어의 품질이 우수하면 그로부터 산출되는 출력 역시 에러가 없는 소위 양품이 나오게 될 것이다. 따라서 소프트웨어의 품질은 소프트웨어가 생성해 내는 출력들의 품질과 동등한 차원에서 파악할 수 있다[5, 6]. 양산체제의 공장에서 나오는 모든 제품의 품질을 검사하는 것은 불가능하거나 설사 가능하다고 하여도 시간적으로나 비용면에서 경제적이지 않는 경우가 많다. 이런 경우 모집단의 일부를 표본으로 선택하여 여기에 들어있는 정보를 토대로 모집단의 품질에 대한 판정을 하게 된다. 이와 마찬가지로 소프트웨어를 통해 나올 수 있는 출력들은 무수히 많으며 이것들 모두가 출력의 모집단을 형성하게 된다. 다시 말해서 소프트웨어를 하나의 공장으로 보았을 때 여기서 생산되는 제품의 수는 무수히 많으며 따라서 이들로 구성되는 모집단은 무한모집단으로 볼 수 있다. 이 모든 제품(출력)을 검사할 수는 없기 때문에 일부인 표본만을 검사하여 그 표본에 담겨있는 정보를 토대로 모집단 전체의 품질을 판정하게 되며, 그 결론은 바로 소프트웨어의 품질에 대응되는 것이다.

그런데 소프트웨어의 출력은 입력과 일대일 대응관계에 있기 때문에 출력을 샘플링한다는 것은 바로 입력을 샘플링하는 것과 같게 된다. 완제품 모집단에서 표본을 취하여 모집단의 품질을 파악한다는 것은, 입력 데이터 집합에서 표본을 취하여 이를 테스트데이터 그룹으로 구성하고 이들에 대한 시험 결과로 소프트웨어의 품질을 파악한다는 것과 마찬가지이다. 테스트 데이터 그룹인 표본을 구성하기 위해서는 우선 샘플링의 대상인 모집단이 명확히 정의되어야 한다. 이를 입력 도메인이라고 하는데, 이는 모든 가능한 입력 데이터와 소프트웨어가 처리할 각각의 입력에 적용되는 규칙의 집합이라고 할 수 있다. 입력 도메인에는 입력변수의 유형, 입력변수의 유형에 대한 특성, 입력변수의 사용규칙 및 입력변수의 사용에 대한 제약조건 등이 망라된다. 입력 도메인이 명확히 정의되면 그로부

터 테스트 데이터 그룹을 쉽게 구성할 수가 있다. Cho[5]는 SIAD(Symbolic Input Attribute Decomposition)트리를 이용하여 소프트웨어의 입력 도메인을 나타낸 바 있다. 이를 이용하면 소프트웨어의 품질을 검사하기 위하여 소프트웨어에 투입될 테스트 데이터 그룹을 선정하기가 용이해진다. SIAD 트리란 소프트웨어의 입력 도메인을 트리의 형태로 표현한 것을 말한다.

이제 테스트 데이터 그룹을 구성하는 방법, 다시 말해서 모집단으로부터 표본을 취하는 방법을 다룬다. 표본에 담겨있는 품질정보를 토대로 모집단의 품질, 즉 소프트웨어 제품의 품질을 평가하는 방법은 3장에서 다룬다. 입력변수는 수치적인 것, 비수치적인 것 등 여러 가지 형태가 있을 수가 있는데, 품질평가와 관련된 의사결정의 신뢰도를 높이기 위해서는 입력 도메인으로부터 통계적으로 랜덤한 형태로 샘플링을 하여 편의(bias)가 없도록 해야 한다.

소프트웨어의 품질을 평가하기 위한 샘플링 방법 중에서 단순무작위 샘플링 방법(simple random sampling)은 입력도메인의 구성요소들 모두에게 뽑힐 확률을 동등하게 하고 뽑는 방법을 말한다. 한편, 가중 샘플링(weighted sampling)은 입력도메인의 구성요소에 대하여 각기 다른 가중치를 부여하여 그 가중치에 대응하는 비율로 입력변수를 뽑는다. 입력변수에 따라 사용빈도나 중요도가 다를 수 있는데 이를 고려하여 샘플링을 하는 방법인 것이다. 단순무작위 샘플링은 가중치를 모두 같게 하여 뽑는 방법이라고 할 수 있다.

경계샘플링(boundary sampling)은 주로 수치적 입력변수에 적용하는 것으로 입력변수가 가질 수 있는 값의 범위가 존재할 때, 즉 입력변수의 하한값이나 상한값이 주어질 때 샘플링하는 방법이다. 만일 입력변수가 s개 있으면 샘플링이 가능한 개수는 2^s개이다. 예를 들어 입력변수의 수가 20이라면 가능한 조합은 2^{20}으로 무려 1,000,000 이 넘는다. 이들을 모두 검사한다는 것은 설사 불가능하지는 않다고 하여도 시간이나 비용면에서 경제적일 수는 없다. 따라서 단순무작위 샘플링이나 가중 샘플링을 통해서 합리적인 수만큼의 입력변수를 선택하게 된다.

한편 무효 샘플링(invalid sampling)이란 입력변수 일부에 대하여 일부러 유효하지 않은 값을 가정하는 것이다. 즉 유효한 값의 하한이나 상한을 벗어난 값을 부여하거나 비수치적 변수에 엉뚱한 문자를 부여하여 이에 대하여 어떻게 소프트웨어가 처리하는지를 보는 것이다. 물론 여기서도 그러한 무효한 입력변수의 조합을 구성한 후 이들 중 표본에 포함시킬 것을 뽑는데는 단순무작위 샘플링이나 가중 샘플링을 이용하게 된다.

3. 적합도 판정

하드웨어의 경우 판정 대상의 롯트가 구성되고 여기서 그 일부인 표본을 취한 후 각각의 제품에 대하여 양, 불량을 가린 후 표본에 포함된 불량품의 수에 따라 롯트의 합격, 불합격을 판정하게 된다. 소프트웨어에 있어서는 2절에서 언

급한 여러 가지 샘플링방법에 의해 테스트데이터 그룹을 구성하고, 테스트 케이스 각각에 대하여 시험을 통해 하드웨어 완제품에 해당되는 출력에 대한 결함 여부를 확인하게 되는데, 결함이란 하드웨어의 불량에 해당된다.

이제 테스트 결과로부터 소프트웨어 제품 품질의 적합도를 판정하는 방법을 모색한다. 적합도를 판정하기 위한 검사는 보통 다음과 같이 시행된다: 모집단(입력도메인)으로부터 크기 n인 표본(테스트케이스 그룹)을 취하여 각각의 테스트케이스에 대하여 시험을 통해 결함 여부를 가린다. n개의 테스트케이스 중에서 결함의 수 x가 합격판정개수 c 이하이면 그 소프트웨어는 적합(합격)으로 판정하고 c보다 크면 부적합(불합격)으로 판정한다. 따라서 적합도를 판정하는 규칙은 표본의 크기 n과 합격판정개수 c에 의해 결정된다.

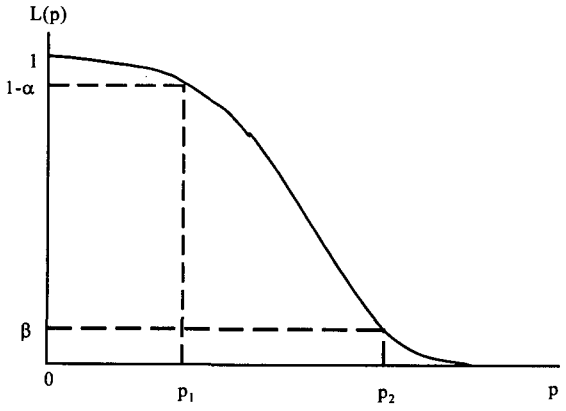
소프트웨어 제품의 적합 판정의 기준이 결함률로 정의될 수가 있다. 합격판정의 기준 결함률이 p_A이라면, 이보다 낮은 결함률을 갖는 소프트웨어는 모두 합격되어야 하고 이보다 높은 결함률을 갖는 소프트웨어는 모두 불합격되어야 마땅하다. 그렇지만 100%검사가 아닌 샘플링검사를 통한 의사결정에서는 판정 오류가 발생할 수가 있다. 즉 합격 판정을 받아야 할 소프트웨어가 불합격 판정을 받는다든가 불합격 판정을 받아야 할 소프트웨어가 합격 판정을 받게 되는 경우가 있게 되는 것이다. 2장에서 언급한 바와 같이 소프트웨어의 적합도를 판정하는 검사는 100% 검사가 이루어질 수 없기 때문에 이와 같은 판정 오류를 피하기가 어렵다.

소프트웨어의 결함률이 p라면 그 소프트웨어가 샘플링에 의한 검사를 통하여 합격될 확률은

$$L(p) = \sum_{x=0}^{c} \binom{n}{x} p^x (1-p)^{n-x} \qquad (1)$$

이다. [그림 1]은 OC곡선이라고 하는데, p의 값의 변화에 따라 L(p) 값이 어떻게 달라지는지를 보여준다. [그림 1]에서 알 수 있듯이 p가 커지면 합격될 확률은 작아지게 된다.

소프트웨어 제품의 인증과정에서는 공급자(개발자)와 소비자(사용자)가 상반된 관점을 갖게 된다. 개발자는 인증을 받기 위해 신청한 소프트웨어가 합격되기를 바랄 것이고, 사용자는 결함률이 높은 소프트웨어라면 불합격되기를 바랄 것이다. 이 경우 소프트웨어 개발자의 관점과 사용자 관점 양쪽을 모두 고려하거나, 그것이 어렵다면 어느 한 쪽의 관점을 우선적으로 고려하여 판정규칙을 정해야 할 것이다.



[그림-1] OC곡선

먼저 양자의 관점을 모두 고려하는 판정규칙을 다루기로 한다. 이 때는 개발자 관점에서 합격되기를 바라는 결함률의 최대값 p_1과 사용자 관점에서 불합격되길 바라는 결함률의 최소값 p_2 ($p_1<p_2$)가 미리 정해져야 한다. 개발자와 사용자 양자의 관점을 반영하는 판정규칙을 구체적으로 표현하면 다음과 같다: 결함률이 p_1 이하인 소프트웨어가 불합격 판정을 받게 될 확률이 α이하가 되게 하고 결함률이 p_2 이상인 소프트웨어가 합격 판정을 받게 될 확률이 β이하가 되게 한다([그림 1] 참조). 여기서 α는 개발자 위험(developer's risk), β는 사용자 위험(user's risk)이다. 결함률이 p_1이하이면 모두 합격되어야 하나 α만큼 불합격될 가능성이 있으며 이는 개발자가 입게 되는 손실이고, 결함률이 p_2 이상이면 사용자 입장에서 모두 불합격되어야 하나 β만큼 합격될 수 있으며 이는 사용자가 입게 되는 손실이다. 100% 검사를 하게 되면 α와 β가 0이 될 수 있으나 소프트웨어 테스팅에서 100% 검사는 사실상 불가능하기 때문에 α와 β가 0이 되기를 기대할 수는 없다. 다만 표본의 크기를 크게 하면, 다시 말해서 테스트 케이스를 늘리면 α와 β를 줄일 수는 있다.

개발자 위험과 사용자 위험을 각각 α와 β로 보호하기 위해서는 다음의 두 식이 만족되어야 한다.

$$L(p_1)=\sum_{x=0}^{c}\binom{n}{x}p_1^{\,x}(1-p_1)^{n-x}\geq 1-\alpha \qquad (2)$$

$$L(p_2)=\sum_{x=0}^{c}\binom{n}{x}p_2^{\,x}(1-p_2)^{n-x}\leq \beta \qquad (3)$$

우리가 다루게 되는 소프트웨어의 결함률은 높은 값을 갖지 않으며 또한 n도 적

당히 큰 값을 가지므로 식 (2)와 (3)은 다음과 같이 포아송분포로 근사시킬 수 있다.

$$\sum_{x=0}^{c}\frac{e^{-np_1}(np_1)^x}{x!}\geq 1-\alpha \tag{4}$$

$$\sum_{x=0}^{c}\frac{e^{-np_2}(np_2)^x}{x!}\leq \beta \tag{5}$$

그런데

$$\sum_{x=0}^{c}\frac{e^{-np}(np)^x}{x!}=P\{\chi^2(2c+2)\geq 2np\} \tag{6}$$

이 성립한다[23]. 여기서 $\chi^2(2c+2)$는 자유도가 (2c+2)인 x^2분포를 따르는 확률변수이다. 따라서 식 (6)을 이용하면 식 (4)와 (5)는 다음과 같이 나타낼 수 있다.

$$P\{\chi^2(2c+2)\geq 2np_1\}\geq 1-\alpha$$

$$P\{\chi^2(2c+2)\geq 2np_2\}\leq \beta$$

그러므로

$$\chi^2_{1-\alpha}(2c+2)\geq 2np_1 \tag{7}$$

$$\chi^2_{\beta}(2c+2)\leq 2np_2 \tag{8}$$

이 얻어진다. 여기서 $\chi^2_{\beta}(2c+2)$은 자유도가 2c+2인 x^2분포함수에서 꼬리확률이 β인 확률변수 값이다.

$$r(c)=\frac{\chi^2_{\beta}(2c+2)}{\chi^2_{1-\alpha}(2c+2)} \tag{9}$$

이라고 하면 r(c)는 c에 대한 감소함수임을 보일 수 있다. 따라서 식 (7)과 (8)로부터 c는

$$r(c)\leq\frac{p_2}{p_1}\leq r(c-1) \tag{10}$$

을 만족하는 가장 작은 정수값이 되며, n은

$$\frac{\chi^2_{\beta}(2c+2)}{2p_2}\leq n\leq\frac{\chi^2_{1-\alpha}(2c+2)}{2p_1} \tag{11}$$

에서 구할 수 있다. [표-1]에는 α=β= 0.01, 0.05, 0.10일 때 c의 변화에 따른 r(c)와 식(9)의 분모, 분자 값이 나와 있다. α나 β의 다른 값에 대해서도 x^2분포표를 이용하여 구할 수 있다.

[표-1]을 이용하여 판정규칙 (n, c)을 구하는 방법을 소개한다. 예를 들어 결함률 p_1=2%에서의 개발자 위험을 α=5%로, 결함률 p_2=8%에서의 사용자 위험을 β=5%로 보호한다면 이에 합당한 판정규칙은 다음과 같이 구한다.

[표-1] c의 변화에 따른 r(c) 값

C	α=β=0.01			α=β=0.05			α=β=0.10		
	r(c)의 분모	r(c)의 분자	r(c)	r(c)의 분모	r(c)의 분자	r(c)	r(c)의 분모	r(c)의 분자	r(c)
0	0.02	9.21	458.22	0.10	5.99	58.40	0.21	4.61	21.85
1	0.30	13.28	44.69	0.71	9.49	13.35	1.06	7.78	7.31
2	0.87	16.81	19.28	1.64	12.59	7.70	2.20	10.64	4.83
3	1.65	20.09	12.20	2.73	15.51	5.67	3.49	13.36	3.83
4	2.56	23.21	9.07	3.94	18.31	4.65	4.87	15.99	3.29
5	3.57	26.22	7.34	5.23	21.03	4.02	6.30	18.55	2.94
6	4.66	29.14	6.25	6.57	23.68	3.60	7.79	21.06	2.70
7	5.81	32.00	5.51	7.96	26.30	3.30	9.31	23.54	2.53
8	7.01	34.81	4.96	9.39	28.87	3.07	10.86	25.99	2.39
9	8.26	37.57	4.55	10.85	31.41	2.89	12.44	28.41	2.28
10	9.54	40.29	4.22	12.34	33.92	2.75	14.04	30.81	2.19
11	10.86	42.98	3.96	13.85	36.42	2.63	15.66	33.20	2.12
12	12.20	45.64	3.74	15.38	38.89	2.53	17.29	35.56	2.06
13	13.56	48.28	3.56	16.93	41.34	2.44	18.94	37.92	2.00
14	14.95	50.89	3.40	18.49	43.77	2.37	20.60	40.26	1.95
15	16.36	53.49	3.27	20.07	46.19	2.30	22.27	42.58	1.91
16	17.79	56.06	3.15	21.66	48.60	2.24	23.95	44.90	1.87
17	19.23	58.62	3.05	23.27	51.00	2.19	25.64	47.21	1.84
18	20.69	61.16	2.96	24.88	53.38	2.15	27.34	49.51	1.81
19	22.16	63.69	2.87	26.51	55.76	2.10	29.05	51.81	1.78
20	23.65	66.21	2.80	28.14	58.12	2.07	30.77	54.09	1.76

식 (10)으로부터

$$r(c) \le \frac{p_2}{p_1} = \frac{0.08}{0.02} = 4 \le r(c-1)$$

이므로 [표-1]에 의하면 c=6이다. 식 (11)로부터 n은

$$\frac{\chi^2_{.05}(14)}{(2)(0.08)} \le n \le \frac{\chi^2_{.95}(14)}{(2)(0.02)}$$

을 만족해야 한다. 즉,

$164.35 \le n \le 199.4$

이어야 하므로 n은 165에서 199 사이의 값을 선택하면 된다. 만일 n=170으로 결정했다면 다음과 같이 검사를 수행하면 된다: 테스트 케이스 170개를 추출하여 각각에 대하여 시험을 한 후 여기에서 결함이 6개 이하가 나왔다면 그 소프트웨어를 적합으로 판정하고 7개 이상이 나왔다면 부적합으로 판정한다.

그런데 이와 같이 개발자와 사용자 양자의 위험을 보호해주는 판정규칙은 기준 결함률 p_A와 p_1, p_2의 관계를 어떻게 정립할 것인지가 다소 막연하다는 문제가 있다. 이에 대한 명확한 입장이 세워져 있지 않다면 양자를 보호하는 위와 같은 규칙은 채택하기가 곤란하다. 이럴 경우에는 기준 결함률 p_A를 중심으로 개발자와 사용자 양자 중에서 어느 한쪽을 우선적으로 고려하여 판정규칙을 결정할 수 밖에 없을 것이다. 소프트웨어 제품에 대한 품질인증의 원래의 취지로 볼 때 개발자 보다는 사용자의 보호가 우선되어야 한다고 여겨진다. 그렇다면 p_A보다 높은 결함률을 갖는 소프트웨어가 합격될 확률을 일정한 값(β) 이하로 제한할 수 있는 규칙을 모색해야 할 것이다. 즉,

$$L(p_A) = \sum_{x=0}^{c} \binom{n}{x} p_A{}^x (1-p_A)^{n-x} \le \beta$$

를 만족하는 (n, c)를 구하면 되므로

$$\chi^2_{\beta}\,(2c+2) \le 2np_A \qquad (12)$$

의 결과를 얻는다.

[표-2]는 β와 c의 변화에 따른 식 (12)의 좌변 값이 나와 있다. 이를 통해 n과 c 중에서 어느 하나의 값이 정해져 있을 때 나머지 값을 구할 수 있다. 예를 들어, 합격기준 결함률 p_A=5% 에서의 판정오류 확률 β를 5%로 제한하고자 한다. 표본의 크기 n이 200으로 미리 정해진 경우라면 2_np_A= 20 이므로 [표-2]에서 β=5%에서 20보다 큰 값 중에서 제일 작은 c 값을 찾으면 되므로 c=5 가 된다. 따라서 테스트케이스 200개 중에서 나온 결함의 수가 5개 이하이면 그 소프트웨어를 적합으로 판정하고 그렇지 않으면 부적합으로 판정한다.

또한 c가 특정한 값으로 미리 정해진 경우도 식 (12)에 의해 n을 구할 수 있다. 예를 들어, c가 0으로 미리 정해진 경우라면 [표-2]에서 c=0일 때 식 (12)의 좌변 값이 5.991 이다. 식 (12)에 의하면

$$n \ge \frac{5.991}{(2)(0.05)} = 59.91$$

이므로 n=60이 된다.

[표-2] 식 (12)의 좌변 값

c	β				
	0.001	0.005	0.01	0.05	0.1
0	13.815	10.597	9.210	5.991	4.605
1	18.466	14.860	13.277	9.488	7.779
2	22.457	18.548	16.812	12.592	10.645
3	26.124	21.955	20.090	15.507	13.362
4	29.588	25.188	23.209	18.307	15.987
5	32.909	28.300	26.217	21.026	18.549
6	36.124	31.319	29.141	23.685	21.064
7	39.252	34.267	32.000	26.296	23.542
8	42.312	37.156	34.805	28.869	25.989
9	45.314	39.997	37.566	31.410	28.412
10	48.268	42.796	40.289	33.924	30.813
11	51.179	45.558	42.980	36.415	33.196
12	54.051	48.290	45.642	38.885	35.563
13	56.892	50.994	48.278	41.337	37.916
14	59.702	53.672	50.892	43.773	40.256
15	62.487	56.328	53.486	46.194	42.585
16	65.247	58.964	56.061	48.602	44.903
17	67.985	61.581	58.619	50.998	47.212
18	70.704	64.181	61.162	53.384	49.513
19	73.403	66.766	63.691	55.758	51.805
20	76.084	69.336	66.206	58.124	54.090

결함은 보통 치명결함(critical defect), 중결함(major defect), 경결함(minor defect)으로 구분할 수가 있다. 치명결함이 하나만 있어도 해당 소프트웨어를 적합하다고 판정하기는 곤란하며, 중결함과 경결함을 같은 비중으로 취급하여 판정해서도 역시 곤란할 것이다. 따라서 결함의 종류별로 판정규칙을 달리 하여 적용하는 것이 바람직할 수 있다. 즉, 치명결함, 중결함 및 경결함 대하여 각각 다른 판정규칙 (n_1,c_1), (n_2,c_2) 및 (n_3,c_3)을 적용하는 것이다. 물론 $c_1=0$이 될 것이며, $n_2=n_3$라면 $c_2<c_3$가 되겠다. 치명결함에 대하여 우선 판정을 하고 여기에서 합격되면 그 다음에는 중결함에 대하여, 그리고 마지막에 경결함에 대하여 판정하면 될 것이다.

ISO/IEC 9126에서는 품질특성을 기능성, 신뢰성, 사용성, 효율성, 유지보수성 및 이식성 등 6가지 범주로 정의하고 있다[9, 10, 11, 12]. 이러한 품질특성들은 모든 소프트웨어에 관계없이 같은 중요도를 갖지는 않는다. 따라서 각 품질특성별로 우선순위를 두고, 각기 다른 판정규칙에 의해 순차적으로, 적용하는 방안도 모색해 볼 수 있다. 만일 모든 품질특성을 한꺼번에 고려하여 종합적으로 판정하

기를 원한다면 품질특성별 가중치를 두어 가중 합계값에 의해 판정하면 될 것이다. 품질특성별 가중치는 AHP[24]를 이용하여 구할 수 있을 것이다.

4. 결론

사회의 제반 요구에 따라 독립된 제3자에 의한 소프트웨어의 품질인증의 필요성이 점차 높아지고 있는데, 본 연구에서는 인증기관에 의한 소프트웨어 제품의 품질인증과 관련된 문제를 통계적 품질관리이론을 통하여 다루었다. 임의성이 배제된 랜덤샘플링에 의하여 테스트 케이스를 선정하는 방법을 다루고 테스트 케이스에 대한 시험 결과에 의해 소프트웨어의 적합성, 즉 인증 여부를 결정하는 합리적인 기준도 제시하였다.

본 연구에서 제시한 통계적인 판정규칙은 인증기관에서 제품의 품질인증에 활용되어 인증결과의 객관성 및 신뢰성을 확보하는데 도움이 될 것이다. 또한 이 방법은 국내 IT업체의 소프트웨어 개발과정 및 제품의 품질보증에도 적용됨으로써 제품의 품질이 향상되어 소비자의 만족도를 제고시킬 수 있을 것이다. 또한 여기서 나온 분석결과가 개발 프로세스의 품질향상을 위한 기초자료를 제공하게 될 것이다. 통계적 방법의 활용 효과란 결국 관리의 정량화 및 경제성에 있다. 이렇게 하여 개발프로세스의 정량적, 합리적인 관리가 가능해지면 개발프로세스의 품질이 향상되고 궁극적으로는 소프트웨어 제품의 납기나 생산성 문제도 획기적으로 해결될 수 있을 것으로 기대된다. 그 결과 CMM의 단계 3이상의 수준으로 소프트웨어 개발업체의 개발능력을 끌어올리는데 일조를 하게 되며, 또한 국내 소프트웨어 개발업체의 국제경쟁력을 확보함으로써 해외진출에 기여할 수 있을 것이다.

인증기관에서 결정해야 할 정책변수라고 할 수 있는 p_A나 p_1, p_2의 값을 합리적으로 결정하는 문제는 차후의 연구과제로 남겨둔다.

참고문헌

[1] 류문찬(1994), 소프트웨어 품질향상을 위한 관리기술에 관한 연구, *산업개발연구*, 고려대학교.

[2] 함동한, 장우현, 유지원, 이상덕(2001), 소프트웨어 품질인증제의 개요 및 인지공학의 역할, 2001 *대한산업공학회 추계학술대회 발표논문집*, 79-82.

[3] Arthur, L.J.(1993), *Improving Software Quality: An Insider' s Guide to TQM*, John Wiley & Sons.

[4] Boegh, J.(2002), Quality Evaluation for Software Products, *Fundamental Concepts for the Software Quality Engineer*(ed., T. Daughtrey), 217-231.

[5] Cho, C.K.(1988), *Quality Programming: Developing and Testing Software with Statistical Quality Control*, John Wiley & Sons.

[6] Cho, C.K. (1996), Statistical Methods Applied to Software Quality Control, *Handbook of Software Quality Assurance*(ed. Schulmeyer,G.G. & McManus, J.I.), International Thomson Computer Press, 2nd ed.17, 427-464.

[7] Cobb, R.H. & Mills, H.D.(1990), Engineering Software under Statistical Quality Control, *IEEE Software*, 7(6), 45-54.

[8] Everett, W., Keene, S. & Nikora, A.(1998), Applying Software Reliability Engineering in the 1990s, *IEEE Transactions on Reliability*, 47(3), 372-378.

[9] ISO/IEC JTC1/SC7 (ISO/IEC 9126-1.2) (1998), *Information Technology - Software Product Quality - part 1: Quality Model.*

[10] ISO/IEC JTC1/SC7 (ISO/IEC 9126-2) (2001), *Software Engineering - Product Quality - part 2: External Metrics.*

[11] ISO/IEC JTC1/SC7 (ISO/IEC 9126-3) (2001), *Software Engineering - Product Quality - part 3: Internal Metrics.*

[12] ISO/IEC JTC1/SC7 (ISO/IEC 9126-4) (2001), *Software Engineering - Software Product Quality - part 4: Quality In Use Metrics.*

[13] ISO/IEC JTC1/SC7 (ISO/IEC 12119) (1994), *Information Technology - Software Packages - Quality Requirements and Testing.*

[14] ISO/IEC JTC1/SC7 (ISO/IEC 14598-1) (1999), *Information Technology - Software Product Evaluation - part 1: General Overview.*

[15] ISO/IEC JTC1/SC7 (ISO/IEC 14598-2) (2000), *Software engineering - Product Evaluation - part 2: Planning and Management.*

[16] ISO/IEC JTC1/SC7 (ISO/IEC 14598-3) (2000), *Software engineering - Product Evaluation - part 3: Process for Developers.*

[17] ISO/IEC JTC1/SC7 (ISO/IEC 14598-4) (2000), *Software engineering - Product Evaluation - part 4: Process for Acquirers.*

[18] ISO/IEC JTC1/SC7 (ISO/IEC 14598-5) (1998), *Information Technology - Software Product Evaluation - part 5: Process for Evaluators.*

[19] ISO/IEC JTC1/SC7 (ISO/IEC 14598-6) (1999), *Software engineering*

- *Product Evaluation - part 6: Documentation of Evaluation Modules.*

[20] Hong, G.Y., Xie, M. & Shanmugan, P. (1999), A Statistical Method for Controlling Software Defect Detection Process, *Computers & Industrial Engineering*, 37, 137-140.

[21] Kelly, D.P. & Oshana, R.S. (2000), Improving Software Quality Using Statistical Testing Techniques, *Information and Software Technology*, 42, 801-807.

[22] Lewis, N.D.C. (1999), Assessing the Evidence from the Use of SPC in Monitoring, Predicting & Improving Software Quality, *Computers & Industrial Engineering*, 37, 157-160.

[23] Rohatgi, V.K. (1976), *An Introduction to Probability Theory and Mathematical Statistics*, John Wiley & Sons.

[24] Saaty, T.L. (1980), *The Analytic Hierarchy Process*, McGraw-Hill.

[25] Smidts, C., Stutzke, M. & Stoddard, R. (1998), Software Reliability Modeling: An Approach to Early Reliability Prediction, *IEEE Transactions on Reliability*, 47(3), 268-278.

[26] Weller, E.F. (2000), Practical Applications of Statistical Process Control, *IEEE Software*, 17(3), 48-55.

An Evaluation of Software Product Quality Using Statistical Quality Control

Moon-Charn Riew · Seong-Taek Rim · Sang-Chul Chung · Sang-Duk Lee · Suk Kyu Shin

Abstract

Improving software product quality is a key to increasing user satisfaction and to achieving competitive edge. There are two approaches to assure high software product quality; development process-oriented and product-oriented. There have been many efforts for improving software quality through process certification, for example, CMM, ISO 9000 family, ISO/IEC 12207, SPICE and Bootstrap. However, a good process alone cannot guarantee good product quality. A need for the evaluation of software product quality by an independent third party is growing rapidly for several reasons. We are concerned with an application of Statistical Quality Control (SQC) to the evaluation of software product quality to obtain the efficiency of evaluation processes and the objectivity of evaluation results. Methods for selecting test cases using a random sampling approach have been discussed and methods for selecting acceptance criteria with respect to software product quality have also been suggested.

◆ 저자소개 ◆

류문찬(Riew, Moon-Charn)
서울대학교 산업공학과를 졸업하고 KAIST에서 석사 및 박사학위를 취득하였다. 현재 고려대학교 경영정보학과에 재직 중이며, 주요 관심분야는 품질경영, 소프트웨어 품질, 서비스 경영 등이다.
E-mail : quality@korea.ac.kr
Tel : (041) 860-1561, Fax : (041) 860-1507

임성택(Rim, Seongtaek)
현재 고려대학교 경영정보학과 교수로 재직중이며, 미국 Georgia State University에서 경영정보학 석사 및 박사 학위를 취득하였다. 주요관심분야는 정보화 전략, 정보시스템 평가, e-business, 데이터 통신 등이다.
E-mail : misrim@korea.ac.kr
Tel : (041) 860-1564, Fax : (041) 860-1507

정상철(Chung, Sang-Chul)
현재 충남대학교 경영학부 교수로 재직중이며, 서울대학교에서 경영학 석사 및 박사 학위를 취득하였다. 주요관심분야는 경영정보, 품질경영, e-business 등이다.
E-mail : scjung@cnu.ac.kr
Tel : (042) 821-5586, Fax : (042) 823-5359

이상덕(Lee, Sang-Duck)
부산대에서 경영학 석사를 취득하였으며, 충남대 박사과정에 재학중이다. 시스템 공학연구소, 한국전자통신연구원을 거쳐 현재 한국정보통신기술협회 IT시험연구소 소프트웨어 시험센터장으로 재직중이다. 주요 관심분야는 소프트웨어 품질평가, 컴포넌트 기반 소프트웨어 개발 등이다.
E-mail : sdlee@tta.or.kr

충남대에서 컴퓨터과학 석사를 취득하였으며, 시스템 공학연구소, 한국전자통신연구원을 거쳐 현재 한국정보통신기술협회 IT시험연구소 S/W시험운영팀장으로 재직중이다. 주요 관심분야는 소프트웨어 품질평가, 컴포넌트 기반 소프트웨어 개발 등이다.
E-mail : skshin@tta.or.kr