논문 2010-2-3

소프트웨어 시험 기법을 적용한 완성도 감정

권기태*

Assessment of Completeness based on Testing Technique

Ki-Tae Kwon*

요 약

소프트웨어 시험이란 응용 프로그램 또는 시스템의 동작과 성능, 안전성이 요구하는 수준을 만족하는지 확인하기 위해 결함을 발견하는 메커니즘이다. 응용 프로그램 또는 시스템이 잘 작동하는지를 확인하는 것이 전통적인 시험 개념이었다면, 현재의 시험은 사용자의 기대 수준과 요구 사항에 맞게 구현되고 동작하는지를 확인하고 이를 통해 결함을 발견하고, 최종적으로는 결함 데이터를 근간으로 개발 프로젝트의 위험도에 대한 수치적인 판단 근거를 프로젝트 관리자에게 전달하는 것이다. 본 연구에서는 소프트웨어 완성도 감정에 시험 기법을 적용하기 위한 연구를 진행한다. 구체적으로는 소프트웨어 완성도 감정 시 단위 시험과 통합 시험의 적용 가능성 및 적용 기법 그리고 그 한계점을 극복하기 위한 대안을 제시한다.

Abstract

Software Testing is defined as a mechanism which can discover faults in order to check out whether application program's operation, performance and safety satisfy the requirements or not. Although traditional testing concept was to validate whether application program operate or not, current testing is to check that system is implemented and operated meeting the requirements. Also, it presents mathematical criteria about the risk of development project based on fault data to the project manager. This paper proposes completeness assessment method that applies the software testing technique. Specifically, it proposes the possibility of unit testing and integration testing being applied. Furthermore, it presents the methods to apply unit testing and integration testing, and solutions to overcome the limitations.

한글키워드 : 소프트웨어 시험, 완성도 감정

1. 서 론

시험이란 응용 프로그램 또는 시스템의 동작과 성능, 안전성이 요구하는 수준을 만족하는지 확인하기 위해 결함을 발견하는 메커니즘이다. 응용 프로그램 또는 시스템이 잘 작동하는지를 확인하는 것이 전통적인 시험 개념이었다면, 현재의 시험은 사용자의 기대 수준과 요구 사항에 맞게 구현되고 동작하는지를 확인하고 이를 통해 결함을 발견하고, 최종적으로는 결함 데이터를

* 강릉원주대학교 컴퓨터공학과 (교신저자)
(email: ktkwon@gwnu.ac.kr)
접수일자: 2010.8.11 수정완료: 2010.10.21

근간으로 개발 프로젝트의 위험도에 대한 수치적인 판단 근거를 프로젝트 관리자에게 전달하는 것이다[1].

이러한 시험 목적을 달성하기 위해서 모든 시험 프로세스는 무엇을 필요로 하는지에 대한 기획, 무엇을 시험해야 하는지에 대한 명세화 또는 설계, 그리고 이러한 시험 사례(test case)를 실행시키기 위한 활동을 포함한다. 또한 모든 결함을 발견하는 것이 불가능하다는 것과 우리가 원하는 어떠한 것을 모두 시험하기에는 충분한 시간 또는 인원이나 비용이 결코 없다는 것은 보편적인 사실이다. 우리에게 허용된 자원을 어떻게 현명하게 사용할 것인지에 대한 선택이 이루어져야 한다[2].

본 연구에서는 소프트웨어 완성도 감정에 시험 기법을 적용하기 위한 방안을 제시한다. 구체적으로는 소프트웨어 완성도 감정시 단위 시험과 통합 시험의 적용 가능성 및 적용 기법 그리고 그 한계점을 극복하기 위한 대안을 제안한다.

2. 단계별 소프트웨어 시험

2.1 소프트웨어 시험의 개념

소프트웨어 시험은 개발된 원시 코드의 품질을 단순히 평가하는 작업이 아니며 디버깅과도 상이한 개념이다. 디버깅이 이미 노출된 소프트웨어의 결함을 제거하는 작업이라면, 시험은 노출되지 않은 숨어 있는 결함을 찾기 위해 소프트웨어를 작동시키는 일련의 행위와 절차를 말한다. 소프트웨어 특성은 결함 없는 개발이 불가능하도록 만든다. 소프트웨어 시험은 결함이 없음을 혹은 얼마나 없는가를 증명하는 것이 아니라 결함이 존재함을 보여주는 작업이다. 이를 보여주는 방법은 소프트웨어로 하여금 많고 어려운 관문들을 통과하도록 환경을 만드는 것이다. 이 관문을 시험 사례라고 부르며, 소프트웨어가 하나의 시험 사례를 통과할 수 없음을 입증시킬 때마다 하나의 결함이 발견된 것으로 정의된다[3].

소프트웨어 시험의 유형은 다음과 같이 분류될 수 있다.

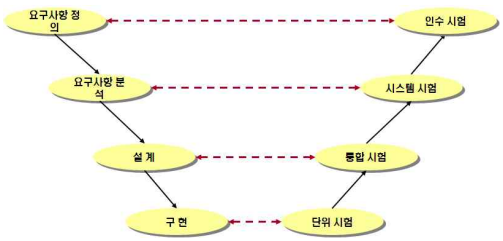
① 시험 단계에 의한 분류: 컴파일된 독립 모듈들의 완전성을 따져보는 단위 시험, 모듈들을 결합시켜 하나의 시스템으로 만드는 통합 시험, 통합된 시스템을 시험하는 시스템 시험, 사용자가 수용할만한 품질과 기능을 갖추었나 시험하는 인수 시험, 혹은 필요에 따라 현장에 시스템을 설치 가동시켜보면서 시험하는 설치 시험 등이다.

② 시험 목적에 의한 분류: 주어진 입력에 기대되는 출력을 과연 제공하는가를 시험하는 기능 시험, 응답시간, 처리량, 그리고 트래픽 등을 시험하는 성능 시험이 있다. 또한 과다한 트래픽이 부과될 때, 최저 조건에 미달되고 최고 조건을 초과할 때, 물리적인 충격이나 변화 시에 반응정도로 신뢰성을 시험하는 스트레스 시험도 있다. 위 세 가지 시험이 사용자를 위한 품질향상 목적의 시스템 시험의 유형이라면 유지보수성을 염두에 둔 구조 시험이 있다. 구조 시험은 단위 시험, 통합 시험, 시스템 시험에 광범위하게 적용되는 개념이다.

③ 시험 방법에 의한 분류: 소프트웨어를 내면은 알 수 없는 블랙박스로 규정하고 외부에서 그 기능 및 성능 등을 시험하는 블랙박스 시험과 소프트웨어 내면의 세계를 파헤치는 화이트박스 시험으로 분류된다.

2.2 완성도 감정을 위한 단계별 시험

소프트웨어 개발 단계와 시험은 [그림 1]과 같이 V-모델로 표현될 수 있다[1].



[그림 1] 소프트웨어 개발과 시험의 V-모델

사용자 환경으로부터 요구가 발생하면 개발하고자 하는 시스템이 정의되고 구체적인 분석 단계를 시스템 설계와 상세 설계로 진행된다. 이어서 상세 설계된 각 모듈들은 프로그램으로 구현되고 각 모듈들은 독립적인 단위 시험을 거쳐 통합 과정을 지나 하나의 시스템으로 묶여질 때까지 다양한 시험을 통과해야만 한다. 이 시스템은 사용자의 인수 시험을 거쳐 설치됨으로써 서비스 및 유지보수 단계로 접어드는 것이다. 각 시험 단계의 목적은 V-모델의 대칭 관계로도 쉽게 설명될 수 있을 것이다.

2.2.1 단위 시험 (컴포넌트 시험)

단위 시험은 테스트 가능한 단위로 분리된 소프트웨어 내에서 결함을 찾고 그 기능을 검증하는 것이다. 단위 시험은 구조적 시험은 물론 기능 시험과 비기능 시험을 포함한다. 시험 사례는 컴포넌트 명세, 소프트웨어 상세 설계 또는 데이터 모델 명세와 같은 개발 중간산출물에서 도출된다.

일반적으로 단위 시험은 소스 코드를 시험 대상으로 하며 주된 시험 방법은 화이트 박스 시험으로 구조기반 시험 기법이 사용된다. 주로 해당 개발자에 의해서도 수행될 수 있으나 타개발자나 제3자에 의해서도 실시될 수 있다.

2.2.2 통합 시험

단위 시험을 통과한 모든 모듈들이 통합되었을 때 문제가 발생하리라는 의심의 배경은 모듈 간의 인터페이스이다. 모듈 사이에서 데이터 손실이 나타날 수 있고 한 모듈이 다른 모듈에 대한 충분한 역할을 수행하지 못하거나 역효과를 낼 수도 있다. 부수기능들의 조합은 주기능을 이루지 못하는 경우도 있으며 개별적으로는 수용가능한 부정확성이지만 통합될 경우 수용 불가능한 수준으로 확대될 수도 있다. 전역 자료구조에도 문제가 발생가능하다.

통합 시험은 인터페이스와 관련된 결함들을 발견하고 제고하면서 모듈들을 체계적으로 조합시키는 작업이다. 즉 시험을 통과한 개별 모듈들로부터, 소프트웨어 구조를 형성하는 것이다. 통합 시험은 모든 모듈들을 한꺼번에 조합하는 빅뱅 방안보다는 상향식과 하향식의 통합방안을 채택한다.

2.2.3 시스템 시험

통합 시험이 끝나면 모든 모듈들은 하나의 시스템으로 작동하게 된다. 그러나 정말 사용자의 모든 요구를 하나의 시스템으로서 완벽하게 수행하는지에 대한 다양한 시험들이 필요하다. 시스템 시험은 개발 프로젝트 차원에서 정의된 전체 시스템/제품의 동작과 관련이 있다.

그러나 시스템 시험에 대한 정의는 모호하며

전문가들 사이에도 의견이 상이하다. 예를 들어, R.S. Pressman은 시스템 시험을 소프트웨어가 하드웨어와 결합한 후 실시되는 시험으로 정의하는가 하면, R. Fairley는 통합시험과 인수시험을 총칭하여 시스템 시험이라고 한다. 또한 소프트웨어 시험의 대가인 G.J. Myers는 외부 기능시험을 시스템 시험에서 제외시키기도 한다. 검증 시험을 시스템 시험에 앞선 인수 시험으로 정의한 사례도 있다[4].

시스템 시험은 기능적/비기능적인 시스템의 요구 사항을 모두 포함해야 한다. 기능적 요구사항의 시스템 시험은 시험 대상의 특성에 가장 적절한 명세 기반(블랙박스) 기법을 사용한다. 비기능적인 요구사항에 대한 시험은 구조기반 기법(화이트박스)을 사용할 수 있다. 시스템 시험은 독립적인 시험 팀이 수행하는 경우가 대부분이다.

2.2.4 인수 시험

인수 시험은 사용자측 관점에서 소프트웨어가 요구를 충족시키는가를 평가한다. 개발자의 책임 영역이 아니라 고객이나 사용자의 영역으로서 계약서의 검수기준과 비교해 보는 것이다. 개발 조직 내의 품질보증 요원에 의해 사용자 집단을 대신하여 진행될 수도 있다. 시험 방법은 역시 시험 사례를 이용하는 것인데 개발 과정 중에 준비된 시험 사례들이 재사용될 수도 있다.

3. 완성도 감정을 위한 시험 설계 프로세스

시험 설계(test design) 기법은 시험 사례를 도출하고 수행하여 시험 대상이 어느 수준까지 시험되었는지 검증하기 위해 사용된다. 완성도 감정을 위해 다양한 종류의 시험 설계 기법으로 어떻게 시험 사례를 도출하고 수행하는지 살펴본다.

3.1 시험 설계 프로세스

시험 설계 과정에서 시험 설계 기법을 이용하여 시험 사례와 시험 데이터를 설계하고 명세화한다. 시험 사례는 입력값의 묶음, 실행 사전조건, 예상 결과와 실행 사후조건으로 구성되고, 특정 시험 조건을 커버하기 위해 개발된다. 시험 사례에는 그 외에도 <표 1>과 같이 ID, 추적성, 중요도, 합격/불합격 등 다양한 내용이 필요에 따라 포함될 수 있다[7].

<표 1> 시험 사례 양식

ID	구분	사전조건	수행순서(시험조건)	예상결과	P/F	추적성	중요도	비고
XX 1.1	전화번호검색(이름)	소프트웨어 탑재 - 시험조건(이름: 이효리, 학번: 20081208)	1. 전화번호부 페이지로 이동 2. 전화번호 찾기 창에서 방향키로 검색	- 해당 이름의 존재 여부가 확인됨 - 입력값에 따른 반응		문서 yy.7.3	중간	…
XX 1.2		소프트웨어 탑재 - 시험조건(이름: 김탁구, 학번: 20081137)	1. 메뉴 버튼을 누르고 오른쪽 방향키를 조정하여 "4. 전화번호부"로 이동한 후 확인 버튼 누름 2. 전화번호부 찾기 메뉴를 선택한 후 확인 버튼 누름 3, 전화번호부 찾기 창에서 방향키를 이름 순으로 차례로 누름	- "김탁구" 입력 시 "검색결과가 없음"이라는 팝업이 나타났다가 사라짐		문서 yy.7.3	높음	…
XX 1.3	…	…	…	…	…	…	..	…

시험 사례 명세의 일부로 생성되어야 하는 예상결과는 결과값, 데이터나 상태의 변화, 다른 시험 결과로 구성된다. 만약 예상결과가 정의되어 있지 않다면, 그럴듯하지만 실제로는 틀린 결과가 올바른 것으로 판정받을 수 있다. 그 이유로 시험 실행 전에 예상결과가 정의되어 있어야 한다.

시험 사례는 4.2절에서 기술하는 시험 기법으로 도출된다. 이렇게 만들어진 시험 사례로 감정 대상 소프트웨어를 시험해야만 결함을 발견할 수 있고, 이를 기반으로 완성도 감정을 실시할 수 있다.

결함은 4가지 경우가 존재할 수 있으나, 완성도 감정의 기준은 요구사항이 되어야 하므로, 기본적으로 완성도에 영향을 주는 결함은 다음 중 ①과 ②에 해당한다.

① 요구사항에 명시되어 있는데 구현되지 않은 경우: 요구사항을 기반으로 시험 사례를 만들어서 실행해야 한다.

② 요구사항대로 구현되었는데, 때에 따라 정상동작하지 않는 경우: 요구사항을 기반으로 시험 사례를 만들어서 실행하고 추가적인 시험 사례를 수행한다.

③ 요구사항에 명시되어 있지 않은데 구현되어 있는 경우: 요구사항 기반의 시험으로는 해당 결함을 발견하기 어려우므로 비공식적인 시험(탐색적 시험 등)을 추가적으로 실행해야 한다.

④ 요구사항에 명시되어 있지 않고 구현은 되어 있는 부분에서 정상동작하지 않는 경우: 요구사항 기반의 시험으로는 해당 결함을 발견하기 어려우므로 비공식적인 시험(탐색적 시험 등)을 추가적으로 실행해야 한다. ③의 경우보다 예상 결과를 알기 어려우므로 요구사항을 기반으로 비공식적인 시험을 보다 체계적이고 강도 높게 설계하고 실행해야 한다.

그러나 분쟁이 일어나 감정을 실시하는 경우에는 요구사항이 분명하지 않거나, 요구사항이 명세화되어 있지 않은 상태에서 구현된 소프트웨어만 존재하는 경우가 있다. 이 경우에는 상기 항목들이 아니라 구현된 소프트웨어를 대상으로 각 기능들을 시험해야 한다.

3.2 완성도 감정을 위한 시험 기법

소프트웨어 시험을 준비하는 궁극적인 목표는 좋은 시험 사례들을 작성하는 것이다. 따라서 이를 위한 방법이 필요하다. 아마도 가장 나쁜 방법은 임의로 떠오르는 값들을 입력시키는 방법인 무작위 입력 시험일 것이다.

시험 사례의 작성은 일종의 최적화 문제다. 모든 가능한 시험 사례 중에서 결함 발견 확률을 가장 높일 수 있는 시험 사례들이 무엇인가 찾는 문제로서 시간, 비용, 컴퓨터 시간 등의 제한이 따른다. 이 최적화를 위한 방법으로서 고안된 것이 시험 기법들이다.

3.2.1 블랙박스 시험 기법 – 명세 기반 기법

블랙박스 시험은 소스 코드는 보지 않은 채 목적 코드를 수행시켜 가면서 결함을 발견할 수 있는 시험 사례를 준비하며 시험에 임하는 방식이다. 데이터 위주 혹은 입출력 위주 시험이라고도 한다. 블랙박스 시험은 부정확하거나 빠진 결함, 인터페이스 결함, 자료구조상의 결함, 성능 결함, 시작이나 종결 상의 결함 등을 찾게 된다.

① 동등분할 기법: 소프트웨어나 시스템의 입력값은 입력의 결과로 나타날 결과값이 동일한 경우 하나의 그룹으로 간주될 수 있다. 그리고 이러한 그룹 내의 입력값은 내부적으로 같은 방식으로 처리됨을 가정한다. 동등분할은 이러한 원리를 이용하여 다양한 입력조건들을 갖춘 시험

사례의 유형들로 분할한 뒤에 각 동등분할의 대푯값 하나를 선택하여 시험 사례를 선정하는 것이다. 동등분할은 모든 시험 단계에서 사용이 가능하다. 단위 시험, 통합 시험, 시스템 시험, 인수 시험 각각에서 동등분할 기법을 사용할 수 있다.

② 경계값 분석 기법: 시험 사례 유형의 아무 값이나 골라 보거나 입력조건에만 치중하는 동등분할 기법과는 다르게 경계치에 치중하며 출력유형도 고려하는 특성이 있다. 동등분할의 경계 부분에 해당되는 입력값에서 결함이 발견될 확률이 경험적으로 높기 때문이다. 동등분할과 마찬가지로 모든 시험 단계에서 적용될 수 있다. 결함 발견율이 높고 적용하기 쉬운 장점이 있어 가장 많이 사용되는 시험 기법이다.

③ 원인-결과 그래프 기법: 동등분할이나 경계값 분석 기법의 단점은 입력 환경의 복합성을 완전하게 고려할 수 없다는 점이다. 입력 데이터 간의 관계가 출력에 영향을 미치는 상황을 체계적으로 분석하여 효용성 높은 시험 사례를 발견코자 원인-결과 그래프 기법이 제안되었다. 이 기법은 입력간 혹은 입출력간의 상관 관계를 사전에 정의된 표기법으로 그리도록 추천한다.

④ 오류 예측 기법: 동등분할, 경계값 분석, 원인-결과 그래프 등 블랙박스 시험 기법들이 흔히 놓칠 수 있을 만한 오류들을 직관과 경험으로 찾아보는 것이다. 흔히 입력 데이터의 오류를 시험하기 때문에 데이터 검증 기법으로 불리기도 한다.

⑤ 결정표 시험 기법: 결정표(decision table)는 논리적인 조건이나 상황을 구현하는 시스템 요구사항을 도출하거나 내부 시스템 설계를 문서화하는 매우 유용한 도구이다. 이것은 시스템이 구현해야 하는 복잡한 비즈니스 규칙을 문서화하는데 사용된다.

⑥ 상태 전이 시험 기법: 시스템은 현재 상황이나 이전의 이력을 반영하는 상태 및 그 변화에 따라 다르게 동작할 수 있다. 시스템의 이러한 측면을 상태 전이 다이어그램으로 표현할 수 있다. 상태 전이 시험은 일반적으로 임베디드 소프트웨어나 기술적으로 자동화가 필요한 부분에서 사용된다. 뿐만 아니라 특정한 상태를 가지는 비즈니스 객체 모델링이나 인터넷 어플리케이션 또는 비즈니스 시나리오와 같이 화면-대화창 흐름을 시험할 때에도 적절하게 사용할 수 있다. 실제로 상태로 표현되는 부분은 거의 모든 경우 상태 전이 시험이 가능하다.

⑦ 유스 케이스 시험 기법: 유스 케이스나 비즈니스 시나리오를 기반으로 시험 사례를 설계하고 수행할 수 있다. 유스 케이스는 시스템이 실제 사용되는 방식에 기반하여 “프로세스 흐름”을 기술하고 있다. 따라서 유스 케이스에 기반하여 생성된 시험 사례는 시스템이 실제 사용되는 프로세스 흐름에서 결함을 발견하는데 유용하다. 특히 인수 시험에서 유용하며, 통합 시험 단계에서 서로 다른 컴포넌트 사이의 상호작용과 간섭으로 발생하는 통합 결함을 찾는데 도움이 된다. 유스 케이스 시험에서는 유스 케이스 각각을 시험하는 방법과 유스 케이스 간의 상호작용과 활동을 시험하는 방법을 생각할 수 있다.

3.2.2 화이트박스 시험 기법 – 구조 기반 시험

화이트박스 시험은 시험 사례들을 만들기 위해 소프트웨어 형상의 구조를 이용하는 것이다. 즉 프로그램상에 이용되는 모든 논리적 경로를 파악하거나 경로들의 복잡도를 계산하여 시험 사

레들을 만드는 기본토대로 삼자는 것이다.

① 시험 영역(test coverage)에 대한 이해: 시험 영역이란 시험하고자 하는 원시 코드의 범위를 말하며 문장영역, 물리적 경로 영역, 논리적 경로 영역으로 분류될 수 있다.

② 구조 시험: 프로그램의 논리적 복잡도를 측정한 후 이 척도에 따라 수행시킬 기본 경로들의 집합을 정의한다.

③ 루프 시험 기법: 프로그램의 루프 구조에 국한해서 실시하는 기법이다. 따라서 구조 시험과 병행해서 사용이 가능하다. 루프 시험은 초기화 결함, 인덱싱 및 증가의 결함, 루프의 경계치에서 나타나는 경계 결함과 같은 루프 구조의 오류를 발견한다.

4. 소프트웨어 완성도 감정

완성도 감정은 3.1절에서 기술한 시험 사례를 3.2절의 시험 기법을 이용하여 도출한 후 수행된다. 시험 사례를 수행한 시험 결과는 의도대로 동작한 경우를 나타내는 합격(Pass)과 의도대로 동작하지 않은 불합격(Fail)으로 구분된다.

4.1 가중치를 고려하지 않은 완성도 감정

완성도 감정의 목적은 감정대상 시스템이 고객의 요구사항을 충분히 반영하고 있는가를 확인하고 향후 시스템 운영의 완전성을 검증하며, 의도한 시스템 품질을 확보하기 위한 것이다.

일반적인 완성도 감정은 소프트웨어 개발이 종료되어 제품이 인도된 후에 혹은 개발 도중에 특정한 사유에 의해 개발이 종료된 후 해당 시점까지 개발된 제품의 완성도를 감정하는 것이다. 따라서 이러한 경우에는 2.2절의 시험 단계에서 시스템 시험이나 인수 시험 기법이 적용될 것이다. 또한 4.1절에서 기술한 시험 사례를 4.2절의 시험 기법을 이용하여 도출한 후 수행된 결과는 합격과 불합격으로 시험 사례들이 구분될 것이다. 만약 최종적인 제품이 개발되고 나서 매뉴얼이 작성된 후에 감정이 실시된다면 매뉴얼을 이용한 시험 사례 도출은 매우 용이할 수 있다. 다만 매뉴얼이 사용자 요구사항을 정확하게 반영하였는가에 대한 검증이 사전에 실시되어야 한다.

첫 번째 완성도 감정 방안은 모든 시험 사례를 규모, 결함의 심각도(severity)에 관계없이 동등하게 취급하는 것이다. 이 경우에 완성도는 다음과 같이 식에 의해서 계산된다.

$$\text{완성도} = \frac{\text{합격한 시험 사례의 수}}{\text{생성된 시험 사례의 수}} \times 100$$

4.2 가중치를 고려한 완성도 감정

시스템의 특성에 따라 가중치를 다음과 같이 고려할 수도 있다.

① 단위 기능별 규모 가중치: 감정대상 시스템을 기능점수 관점의 트랜잭션 기능별로 감정을 실시할 경우에는 각 단위 기능의 규모가 상이하다는 것은 명백한 사실이다. 이때 각 기능의 규모에 따른 가중치를 반영해야 한다. 이 경우 시스템의 완성도는 앞의 경우처럼 시험 사례의 수가 아니라 각 트랜잭션 기능을 대상으로 설계된 시험 사례를 기준으로 다음과 같이 완성도를 감정한다. 이 방식은 시험과 함께 기능점수 산정이라는 부수적인 작업이 필요하다.

$$\text{완성도} = \frac{\text{합격한 시험 사례의 기능점수}}{\text{생성된 시험 사례의 기능점수}} \times 100$$

② 결함의 심각도 가중치: 감정 대상 시스템이 사용자 요구사항을 달성하지 못했다고 하더라고, 결함의 심각도 수준(severity level)에 따라 완성도를 다르게 판단해야 할 수도 있다. 결함 심각도를 완성도 감정에 반영하기로 결정하면, 수발주자 간에 결함 심각도 간의 구분을 명확한 기준을 가지고 정의해야 한다. 다음은 결함 심각도 구분 기준과 각 기준별 스케일의 예이다.

- 치명적 결함(critical defects) - 10: 소프트웨어 장애, 시스템 중단, 시스템 잠김, 데이터 손실 또는 변조
- 주요 결함(major defects) - 8: 기능 누락, 잘못된 기능, 주요 기능 오작동
- 일반 결함(average defects) - 6: 불완전한 기능, 사소한 기능 오작동, 잘못된 인터페이스
- 사소한 결함(minor defects) - 4: 타이핑 에러, 사용자 불편, 스크린 표준의 위반, 좋지 않은 인터페이스
- 개선사항(enhancement) - 2: 에러는 아니지만 개선이 필요한 사항

이때 결함도 수준 선정의 변별력을 높이고 결함도가 높고 낮은 것을 확실하게 구분하기 위해 적절한 가중치 스케일을 사용한다. 물론 해당 스케일은 프로젝트의 성격과 조직의 특성 등을 반영하여 변형시켜 사용할 수 있다.

예를 들어, 10개의 시험 사례가 있고, 각 시험 사례에 대한 내역은 <표 2>와 같다고 가정하자.

완성도는 다음과 같이 계산된다.

$$\frac{\text{시험결과 시험사례의 가중치합}}{\text{전체 시험 사례의 가중치합}} \times 100 = \frac{70}{100} \times 100 = 70\%$$

만약 심각도를 고려하지 않고, 4.1절의 방식에 의해 완성도를 계산하면, 다음과 같다.

$$\frac{\text{합격한 시험 사례의 수}}{\text{생성된 시험 사례의 수}} \times 100 = \frac{5}{10} \times 100 = 50\%$$

<표 2> 시험 사례 내역

시험사례	시험 결과	심각도	스케일	가중치
1	실패	치명적 결함	10	0
2	실패	주요 결함	8	2
3	실패	일반 결함	6	4
4	실패	사소한 결함	4	6
5	실패	개선사항	2	8
6	합격	-	0	10
7	합격	-	0	10
8	합격	-	0	10
9	합격	-	0	10
10	합격	-	0	10
합계			**30**	**70**

이 방식의 경우에는 최종적인 완성도에 큰 영향을 미칠 수 있는 결함 심각도를 구분하기 위한 기준 및 스케일에 대한 합의가 필요하다. 또한 각 시험 사례마다 결함 심각도를 위한 판단하기 위한 부수적인 작업이 필요하고, 매우 많은 개수의 시험 사례가 유도되는 대규모 시스템의 경우에는 이 작업이 큰 부담이 될 수 있다.

5. 결론

소프트웨어 시스템은 비즈니스 어플리케이션부터 소비자 제품에 이르기까지 생활의 많은 부분에서 사용되고 있으며 그 비중은 계속해서 증가하고 있다. 그러나 소프트웨어가 요구사항대로 동작하지 않는 경우 다양한 문제가 발생한다. 소프트웨어 개발 과정 중에 각각의 개발 단계에 대

응하는 시험이 실시된다. 이와 같은 시험은 개발 과정에서 소프트웨어의 품질을 높이고 고객이 발견할 수 있는 결함을 최소화할 수 있다. 그러나 상당수의 시스템은 아직 발견되지 않은 결함들이 존재하고 이러한 결함을 가진 시스템을 대상으로 완성도 감정이 이루어진다. 본 연구에서는 이와 같은 완성도 감정을 위한 다양한 종류의 시험 설계 기법으로 어떻게 시험 사례를 도출하고 수행하는지 살펴보았다.

참 고 문 헌

[1] 권기태, 남영광 공역, “소프트웨어공학”, 제8판, 피어슨에듀케이션 코리아, 2008.
[2] 윤희병 외 4인 공역, “임베디드 소프트웨어 테스팅”, 홍릉과학출판사, 2007.
[3] 이주헌, “실용 소프트웨어공학론”, 법영사, 2004.
[4] Glenford J. Myers, "The Art of Software Testing", 2nd Ed. John Wiley & Sons, Inc., 2004.
[5] 권기태, “기능점수”, HP Education Center, 2010.
[6] 권기태 외 2인 공역, “기능점수와 소프트웨어 측정”, 도서출판 그린, 2003.
[7] 권원일 외, “개발자도 알아야할 소프트웨어 테스팅 실무”, 제2판, STA, 2008.
[8] FP표준기술연구회, “기능점수 측정 사례집”, 산업자원부 기술표준원, 2006.
[9] 최종천, “패키지기반 ERP 시스템 감정에 관한 고찰”, 한국저작권위원회 2010년 저작권 감정인 워크숍, 2010.
[10] 권용래, “소프트웨어 테스팅”, 생능출판사, 2010.

저 자 소 개

권 기 태

1986년 서울대학교 계산통계학과 졸업
1988년 서울대학교 계산통계학과 석사 졸업
1993년 서울대학교 계산통계학과 박사 졸업
1996년 미국 Univ. of Southern California, 전산학과 Post-Doc.
현재 강릉원주대학교 컴퓨터공학과 교수

<주관심분야 : 소프트웨어공학, 데이터마이닝, 지능시스템>