논문 2023-4-3 http://dx.doi.org/10.29056/jsav.2023.12.03

SW 완성도 감정 시 문제점 해결 방법

우 균*†

Solving the Problems in the Appraisal on the Completeness of Software

Gyun Woo*†

요 약

소프트웨어 개발이 활발해짐에 따라 소프트웨어 완성도에 대한 분쟁 사건이 늘어나고 있다. 소프트웨어 완성도에 대한 감정 요구는 늘어나고 있지만, 감정 과정에서는 여러 문제점이 대두되고 있다. 먼저 감정 목적물 소프트웨어의 완성도가 현저히 낮을 수 있다. 두 번째로 소프트웨어 모듈의 구조가 매우 복잡할 수 있다. 끝으로 소프트웨어의 실행 환경이 구축되기 힘들 수 있다. 이 논문은 이러한 문제점에 대처하기 위한 몇 가지 가능한 방법을 제안한다. 사례 연구를 통해 이들 방법의 적용 가능성을 타진한다. 제안 방법은 소프트웨어 완성도 감정의 부담을 경감시킬 수 있다.

Abstract

With the active developments of software, the cases of disputes on the completeness of software are also increasing. Despite the need for the appraisal of the completeness of software, several problems are encountered during the appraisal process. First of all, the completeness of the target software can be severely low. Secondly, the structure of the software modules can be very complicated. Lastly, the execution environment of the software can be hardly reconstructed. The paper proposes a set of plausible methods to cope with these problems. The applicability of the methods is examined using case studies. The proposed methods can relieve the burden of the appraisal on the completeness of software.

한글키워드 : 소프트웨어 감정, 소프트웨어 완성도, 테스트, 소프트웨어 구조, 실행 환경

keywords : software appraisal, software completeness, test, software structure, execution environment

1. 서 론

인공지능과 더불어 소프트웨어의 발전에 힘입어 소프트웨어 감정 요구는 많아지고 있지만, 소프트웨어 감정 작업은 점점 어려워지는 실정이다. 이는 소프트웨어 구조 자체가 복잡해졌기 때문이기도 하지만, 다양한 공개 소스(open source) 및 프레임워크와 더불어 소프트웨어의 감정 범위가 넓어졌기 때문이다. 또한, 소프트웨어 감정은 대개 법적 분쟁과 연관되는데, 이로 인해서 분쟁 양측에서는 소프트웨어 자체에 초점을 맞추는 것에 벗

* 부산대학교 정보컴퓨터공학부
† 교신저자: 우 균(email: woogyun@pnu.edu)
접수일자: 2023.12.02. 심사완료: 2023.12.12.
게재확정: 2023.12.20.

어나 프로젝트 전체로 감정 범위를 확대하여 해석하기도 한다.

이러한 상황에서 감정인의 업무가 종래의 감정 업무보다 가중되는 경향이 있으며, 종국에는 법정에 증인으로 출석하거나 법적 분쟁에 연루되는 예도 있다. 특히 이러한 경향은 소프트웨어 완성도 감정의 경우에 더 심하게 나타나는데, 완성도를 바탕으로 한 개발비 분쟁의 경우에는 감정 결과가 분쟁 양측에 중요한 자료로 활용되기 때문이다.

본 연구에서는 소프트웨어 완성도 감정의 경우에 흔히 발생하는 문제점과 이에 대한 해결책을 모색하고자 한다. 특히 소프트웨어 감정 시 체험하게 되는 다양한 문제점을 열거하고 이에 대한 가능한 해결 방안을 제시한다. 2절에서는 소프트웨어 완성도 감정과 관련된 과거 연구를 살펴보고 3절에서는 소프트웨어 감정 관련된 몇 가지 용어를 정리하고 본 연구의 범위를 지정한다. 4절에서는 완성도 감정 시 접하게 되는 문제와 이에 대한 해결 방안을 하나씩 기술한다. 5절의 사례 연구와 6절의 토의 후 7절에서 결론을 맺는다.

2. 관련 연구

소프트웨어 감정 관련해서는 과거 많은 연구가 진행되었으며, 이러한 관련 연구는 특히 소프트웨어 완성도 감정에 큰 방향을 제시하였다고 할 수 있다. 특히 완성도 감정의 일반적인 절차에 대해서[1], 또 기성과 관점에서 완성도를 어떻게 고려할 것인지에 관해서[2, 3] 많은 연구가 진행되었다.

대부분의 연구[2, 3]에서 소프트웨어 완성도는 기능점수 방법으로 산출되어야 함을 제시하고 있다. 다만, 소프트웨어 기능점수를 측정할 때 사용되는 용어에 대해서는 정리할 필요가 있다. 기능이 충분히 구현되었고 정상적으로 동작하는 경우에는 큰 문제가 없지만 오동작하는 경우에는 구현 상황 및 동작 상황에 대해 명확히 정의할 필요가 있다.

윤영선의 연구에 따르면 구현된 코드의 존재 여부에 따라 미구현, 부분 구현을 구별하였으며, 구현 코드와는 별개로 동작 형태에 따라 미동작, 부분 동작, 오동작을 구별하였다. 구현된 코드를 검토하는 상황은 소프트웨어 검사 분야의 화이트 박스 테스트(white box test)에 해당하며, 동작 형태를 검토하는 상황은 블랙박스 테스트(black box test)에 해당한다. 감정 요청사항에 적시된 기능 여부만 고려한다면 블랙박스 테스트만 고려해야 하겠지만 감정 요청서에 해당 코드의 검토가 적시되어 있다면 화이트 박스 테스트를 고려해야 할 것이다.

블랙박스 테스트 관점에서 부분 동작과 오동작은 가능한 경우 구별하여 감정하는 것이 필요하다. 부분 동작은 해당 기능의 일부분만 구현된 것을 의미하나, 오동작은 기능 전체가 구현된 것을 전제로 하였을 때 일부 입력에 대해 예상과 다른 동작을 보이는 경우를 의미한다. 부분 동작과 오동작은 모두 블랙박스 테스트 관점에서 소프트웨어를 관찰하였을 때 얻을 수 있는 것이다. 소프트웨어 분야에서 언급하는 버그(bug)는 블랙박스 테스트 관점의 오동작과 유사하지만, 사실 버그는 화이트 박스 테스트 관점의 소프트웨어 오류이다. 즉, 감정 당시에 오동작을 관찰할 수 없다고 하더라도 내부 논리적 설계에 흠결이 있었다면 버그라고 분류할 수 있다. 김도완의 연구[3]에서도 하자(오동작)와 버그를 분류해야 한다고 언급하고 있다.

요약하면 소프트웨어를 화이트 박스 관점에서 보느냐, 블랙박스 관점에서 보느냐에 따라 각 기능의 구현 여부를 판단하는 기준이 달라질 수 있다는 것을 고려해야 한다. 특히 소스 코드가 감정 대상에 포함되는 경우에는 외형적인 동작의 구현

여부 외에도 실제 해당 구현 코드의 존재 여부도 고려해야 하며, 구현 코드가 존재할 때에는 버그의 존재 여부도 함께 고려할 수 있을 것이다. Dijkstra가 지적한 것[4]처럼 테스트를 통해 버그의 존재 여부를 관찰할 수 없을지라도, 버그의 존재를 감정인이 확인한 경우에는 이를 감정서에 적시하는 것이 필요하다. 버그의 부재를 확인하는 것은 소프트웨어 감정의 범위에 벗어나며 이는 소프트웨어 증명에서 다루어야 할 것이다.

3. 소프트웨어 감정 개요 및 연구 범위

소프트웨어 감정은 크게 유사도 감정과 완성도 감정으로 나눌 수 있다. 유사도 감정은 이미 등록된 소프트웨어와 감정 목적물의 유사도를 감정하기도 하고, 감정 목적물에 포함된 두 소프트웨어의 유사도를 감정하기도 한다. 어느 경우든 원본 소프트웨어와 유사한 형태로 개발된 모듈이 존재하는지 판단하는 형태로 감정이 진행되는데, 같은 프로그래밍 언어로 구현되어 있는 경우에는 원활하게 감정이 진행되는 편이지만, 구현 언어가 바뀐 경우[1]에는 유사도 감정 자체가 난해한경우도 있다. 특히, 언어를 바꾸어 재구현한 경우에는 구현 노력이 포함된 것으로 간주할 수 있으므로 유사도 자체를 판단하는 기준을 고도화해야 한다. 소프트웨어의 소스 코드를 컴파일러나 난독화 도구 등 기계적 방법으로 처리한 경우는 이 범주에 포함시키지 않는다.

소프트웨어 완성도 감정의 경우에는 소프트웨어 개발 용역의 경우에 흔히 발생하는데, 개발 용역 발주사와 개발사 사이의 용역 대금 청구 및 반환 관련하여 감정 요청이 발생하는 경우가 흔하다. 소프트웨어 개발이 진행된 것은 맞지만, 원하는 형태로 개발되지 않은 경우, 흔히 양측의 분쟁이 시작된다. 개발사 입장에서는 그간 들인 노력의 대가를 받으려고 하지만, 발주사 입장에서는 입안했던 것과 상당히 다른 결과로 인해 대금 지급을 거절하는 경우가 흔하기 때문이다.

소프트웨어 완성도 감정이 필요한 분쟁은 정교한 계약을 통해 일부 방지할 수 있다. 소프트웨어 개발 대가 중도금 지급 시기를 중간 산출물 이후로 늦추는 형태로 계약하거나, 개발 과정 중에 발생하는 산출물을 발주사가 확인하는 형태로 계약한다면 개발 중에도 소프트웨어 개발 과정을 확인할 수 있으므로 큰 문제가 발생하지 않는다. 그러나 폭포수 모델을 따르는 경우 중간 과정에서 구체적인 소프트웨어 형태가 나타나지 않는 경우도 흔히 발생하므로 계약 고도화가 근본적인 대책이라고 하기는 어렵다.

소프트웨어 완성도 감정은 흔히 기능점수(number of functional points) 방식으로 산출되는데, 기능점수 방식의 완성도는 식 1에 따라 산출된다.

$$\text{완성도}(\%) = \frac{\text{정상 작동 기능점수}}{\text{감정대상 총 기능점수}} \times 100 \quad (1)$$

식 1의 기능점수는 각 기능 수를 세는 방식으로 진행될 수도 있으며, 기능의 중요도에 따라 가중치를 두는 방식으로 진행될 수도 있다. 가중치를 두는 방식을 택하는 경우, 계약서의 계약 내용에 따라, 또 감정인의 전문성에 따라 가중치를 둘 수 있는데, 이미 분쟁이 시작된 경우에는 가중치 자체에 양측의 의견이 달라질 수 있으므로 공정성을 높이기 위해 단순 개수로 판단하기도 한다.

식 1에서 정상 작동 기능점수는 2절에서 언급

1) 구현 언어를 바꾸는 경우는 대부분 유사한 패러다임의 언어에 한정된다. 예컨대 C나 C++ 코드를 Java 코드로 바꾼다든지, Python 코드를 JavaScript 코드로 바꾼다든지 하는 경우가 이에 해당한다. C와 C++의 경우처럼 한 언어가 다른 언어의 부분집합인 경우에는 이 범주에 포함하지 않는다.

한 대로 완전히 구현된 형태가 아니라 부분적으로 구현될 수도 있다. 이처럼 부분 구현된 기능은 기능의 구현 정도에 따라 가중치를 부여해야 할 것이다. 기능이 전혀 동작하지 않는 미동작의 경우에도, 소스 코드를 확인할 수 있는 경우에는 해당 구현의 정도에 따라 부분 구현 가중치를 둘 수 있을 것이다. 그러나 이 경우 소프트웨어 전체에 관하여 일관된 기준으로 진행되어야 한다.

이 연구에서는 소프트웨어 감정에서 완성도 감정만을 다룬다. 완성도 감정은 소프트웨어의 기능을 파악하는 것이므로 요구사항 분석 과정을 통해 소프트웨어의 요구사항이 기능별로 명시된 경우에 감정하는 것이 적절하다. 다만 소프트웨어 개발이 크게 진척되지 않은 경우나, 개발 업체가 영세하여 요구사항 분석 과정을 생략한 경우에는 계약서나 회의록, 기타 서면 자료를 통해 요구사항을 분석하는 과정이 필요할 수 있다.

4. 완성도 감정의 문제점 및 해결 방안

4.1 수행 환경 문제

소프트웨어가 일부 혹은 전부가 개발된 경우에 흔히 발생하는 문제는 소프트웨어 수행 환경에 관한 문제이다. 통상 소프트웨어 개발이 실패한 경우, 감정 의뢰 및 감정 개시 전까지 상당한 시간이 흐르는 경우가 대부분이다. 이런 경우에 소프트웨어의 생명주기(life cycle)보다 기술의 진보 속도가 빠를 수 있는데, 이러한 경우에 소프트웨어 수행 환경 자체를 재현하기가 어려울 수 있다.

특히 웹 소프트웨어의 경우에는 프레임워크의 기술 개발 및 업데이트 속도가 매우 빠르게 진행되는 경향이 있는데, 이러한 경우에 이전 프레임워크 자체가 지원되지 않는 경우가 허다하다. 소프트웨어 개발 주기마다 이를 예측하여 가상화 환경으로 구축하는 것은 상당히 번거로운 작업이며 개발 시한에 쫓기는 개발사에게 이러한 것을 기대하기란 어려운 것이 사실이다. 게다가 이렇게 구축해 두었다고 하더라도 프레임워크 구성 요소 사이의 의존성으로 인해 개발 당시의 환경을 그대로 복원하는 것이 불가능할 수도 있다.

개발사가 개발 당시의 테스트 서버를 개발 당시 수준으로 보존하고 있는 경우나, 발주사가 납품된 환경을 그대로 유지하는 경우에는 이를 활용할 수 있다. 그러나 이 경우에도 감정 목적물과 서버의 결과물이 일치하는 것인지 확인 과정이 필요하다. 납품된 소프트웨어의 소스 코드(A)가 존재하고 감정 목적물에도 해당 소스 코드(B)가 적시된 경우, A와 B의 유사도 비교를 통해 이를 확인할 수 있다. 그러나 A와 B가 일치한다고 하더라도 실행 환경의 코드가 이진 코드라면 이 코드가 A로부터 산출된 것인지 확인하는 작업이 추가로 필요하다.

4.2 소프트웨어 부재 혹은 빈약한 문제

소프트웨어 완성도 감정 시 흔히 발생하는 문제점은 감정 방법에 대한 논쟁이다. 앞서 언급한 대로 소프트웨어 감정은 기능점수 방식을 택하고 있는데, 중도금 지급 전에 분쟁이 발생하는 경우나 개발 일정이 지연된 경우 개발된 소프트웨어 자체가 전혀 없거나 개발 분량이 극히 미미할 수 있기 때문이다.

개발된 소프트웨어 자체가 전혀 없는 경우 기능점수 방식으로 완성도를 감정하게 되면 0%로 산정될 수밖에 없다. 이 상황에서 감정 방법 자체를 재고하지 않으면 개발사 측에 상당히 불리한 감정 결과가 나올 수밖에 없다.

이런 경우에는 기성고(旣成高)를 고려하여 감정을 진행할 수 있는데, 간단히 말해서 소프트웨어 구현 전에 진행했던 작업을 고려하는 방식이다. 사실 '기성고'란 건축 분야에서 흔히 사용되는 용어로서, 건축물 일부만 건설한 상황에서 총공사비

중, 이미 건축된 부분에 대한 건설 비용을 뜻하는 용어다. 영세 업체가 건설을 담당하는 경우 이런 일이 흔히 발생하는데, 소프트웨어의 경우에 소프트웨어의 범위를 프로젝트 전체로 확대하여 산정할 때 기성고 개념을 도입할 수 있다.

그러나 기성고를 고려할 때도 중간 산출물을 확인해야 한다. 한국소프트웨어산업협회에서 공표한 “SW사업 대가산정 가이드”[5] 및 윤영선의 연구[2]에 따르면 소프트웨어 개발 단계(분석 단계, 설계 단계, 구현 단계, 시험 단계)마다 수행해야 하는 활동 및 작업에 대해 대체적인 합의가 이루어져 있으며 표준 산출물도 제시되어 있다. “SW사업 대가산정 가이드”에 따른 개발 단계별 비중을 보면 표 1과 같다.

표 1. 개발 단계별 비중(SW사업 대가산정 가이드)
Table 1. The weights of the development stages

개발 단계	비중(%)
분석	19
설계	24
구현	32
시험	25
계	100

그러므로 기성고를 고려하여 완성도 감정을 진행할 때에는 이러한 산출물을 확인해야 하며 산출물의 완성도도 아울러 확인해야 한다.

특히, 감정대상 소프트웨어 자체가 전무하거나 미비한 경우에는 기성고 관점의 완성도와 기능점수 방식의 완성도를 병행으로 제시하는 것도 한 가지 방법이다. “SW사업 대가산정 가이드”에 따른 기준은 말 그대로 프로젝트 시작 전에 적정한 개발 비용을 산정하기 위한 것이며 소프트웨어 완성도 감정의 기준이 되기에는 여러 가지로 미비한 점이 있다. 이는 여러 연구에서도 지적된 바가 있으며, 기성고 관점에서 고려해야 하는 투입공수, 즉 맨먼스(man-month)에 관해서는 소프트웨어 완성도와 항상 정비례 관계에 있다고 볼 수 없기 때문이다[7].

따라서 산출물을 통해 투입공수를 산출하는 것은 개발사의 노력을 일부 반영한다는 측면에서 제시할 수 있을 뿐이며 투입공수가 실제 최종 결과물인 소프트웨어 완성도에 얼마나 기여하는지는 정확히 판단할 수 없다. 따라서 기능점수 방식의 완성도만 고려하기 힘든 경우에 한해 기성고 관점의 완성도를 함께 제시하는 것이 적절할 것이다.

4.3 복잡한 구조의 소프트웨어

개발사에게 요구된 소프트웨어 개발 프로젝트가 여러 모듈로 구성된 경우, 각 모듈의 완성도가 매우 다를 수 있다. 이런 경우에 모듈마다 완성도가 다를 수 있으며 모듈에 따라서는 완성되지 기능이 전혀 완성되지 않을 수 있다. 이 경우 모듈의 완성도를 간단히 0%로 산출할 수도 있겠지만, 설계 과정에서 다른 모듈과의 연관성을 고려하여 완성도를 일부 반영할 수 있다. 이는 4.2절에서 논의한 기성고 관점에서 파악할 수 있다.

이처럼 다른 모듈과의 연관성을 고려할 때, 특히 중요한 점은 공통 데이터 모델의 구축 여부이다. 여러 모듈로 구성될 경우 데이터 구조를 공유하는 경우가 있을 수 있으며, 이러한 공통 데이터 모델을 바탕으로 기능을 구현하는 경우가 대부분이다. 그러나 기능점수 모델을 바탕으로 하였을 때는 데이터 모델의 구축은 부분적으로 산출될 수 밖에 없다. 즉, 해당 데이터 모델을 활용하는 기능이 구현되었을 때만 산출될 수 있다.

공통 데이터 모델이 구현된 경우에는, 해당 데이터 모델을 활용하는 기능점수에 일부 구현 정도를 반영할 수 있을 것으로 생각된다. 이는 일찍이 Wirth가 주장했던 것처럼[6] 프로그램은 알고리즘과 데이터 구조로 구성되기 때문이다. 따라서 이러한 공통 데이터 구조의 구현은 이와 연관된 기

능점수 구현에 일부 반영할 수 있을 것이며, 이때 기능점수에 반영되는 정도(가중치)는 앞서 논의한 것처럼 해당 분야 전문 감정인의 판단에 맡겨야 할 것이다.

5. 완성도 감정 사례 연구

5.1 장례식장 업무관리 시스템

장례식장 업무관리 시스템은 종합병원 장례식장에서 장례식장별 안내 시스템과 판매 시스템을 결합한 시스템을 개발하는 시스템이었다. 이 경우 기존 병원 시스템과의 연동이 중요한 요구사항이었다. 결재 내역 등은 병원 시스템과 연동되어 관리되어야 하기 때문이다.

이 감정 사례의 문제점은 감정 목적물이 특정되지 않았다. 그도 그럴 수 있는 것이 감정 대상 소프트웨어가 키오스크와 안내 서버, 병원 서버 등 여러 시스템과 연관되어 존재하기 때문이다. 따라서 감정을 요청한 측에서도 소프트웨어를 특정하지 못하고 감정만 의뢰한 상태였다.

이 경우 감정 목적물을 특정하기 위해서 현장 방문이 필요했다. 그리고 안내 서버로 특정된 컴퓨터의 하드디스크를 복제하여 감정 목적물을 확보할 수 있었다. 해당 하드디스크에는 감정 목적물로 추정되는 프로그램이 여럿 있었으나, 감정 요청서에 명기된 날짜 부근의 파일을 검색하여 감정 목적물을 특정할 수 있었다.

5.2 약국 관리 시스템

약국 관리 시스템은 세 개의 계약으로 구성된 세 가지 시스템을 개발하는 대형 시스템이다. 각 시스템은 인공지능 헬스케어, 블루채널, 몰인몰 시스템인데, 세 시스템 모두 웹 시스템이 주요 시스템으로 계약되어 있으며 블루채널 시스템에는 앱 개발도 포함되어 있었다.

인공지능 헬스케어 프로젝트는 챗봇 개발 프로젝트였는데, ChatGPT[8]와 유사한 형태로 맞춤법 오류에 강건하며 추론 가능한 형태의 챗봇을 구현하는 것이었다. 이 챗봇은 요구사항과 달리 맞춤법 오류에 강건하지 못하며 추론 기능 또한 결여된 프로그램이다. 챗봇 프로그램 프레임워크에 데이터베이스만 탑재된 형태라고 할 수 있다.

이 경우에는 감정 요청서에 명시된 요구사항이 맞춤법 오류 처리 기능, 추론 기능, 질문 분류 기능, 고객의 특성 관리 기능 등이었다. 당시 기술 수준을 볼 때, 이는 상당히 구현하기 어려운 기능으로 판단되나 개발사에서는 이를 충분히 구현할 수 있을 것으로 판단했던 것 같다.

두 번째 프로젝트는 쇼핑몰 프로그램이었는데, 상품 등록뿐 아니라 판매 POS 관리를 수행해야 한다. 두 번째 프로젝트는 퍼블리싱(publishing) 소스 형태로만 존재했다. 퍼블리싱 소스를 실행하면 판매 데이터가 그래프로 분석되는 것처럼 보이나, 이는 실제 데이터베이스에서 가져오는 결과가 아니라 임의 데이터를 보여준 형태였다.

세 번째 프로젝트의 감정 목적물에서는 화면 설계만 제시되었을 뿐 코드를 찾을 수 없었다.

5.3 사례별 분석 결과

앞서 기술한 두 건의 감정 사례에 관하여 분석된 결과를 요약하면 표 2와 같다. 두 건의 감정 사례 중에서 약국 관리 시스템의 경우, 실질적으로 세 개의 감정 대상이 포함되어 있으므로 이를 구별하여 정리하였다. 표 2에서는 감정 대상별 문제점과 이에 대한 해결 방안을 정리하고 있다.

표 2에서 볼 수 있는 것처럼 약국 관리 시스템은 세 개의 시스템으로 구성된 매우 복잡한 구조의 소프트웨어였다. 따라서 기성고 관점의 완성도 범위를 산정하였으며 이 과정에서 일부 설계 문서와 퍼블리싱 소스의 동작을 고려하였다. 또한, 데이터베이스에 설계된 모델도 설계의 일부로 고려하였다.

표 2. 감정 대상별 문제점 유형 및 가능한 해결 방안
Table 2. The kinds of problems of the software in the case study and the possible solutions

감정 대상		문제점 유형	해결 방안
장례식장 업무관리 시스템		수행 환경 재현 문제 감정 목적물 특정 문제	설치된 환경 활용 저장소 복제 후 수정 일자 정렬
약국 관리 시스템	인공지능 헬스케어	수행 환경 재현 문제	서버의 환경 활용 완성도와 더불어 기성고 관점의 범위 산정
	약국 POS	감정 목적물 부실	퍼블리싱 소스의 동작 고려 완성도와 더불어 기성고 관점의 범위 산정
	폐쇄적 쇼핑몰	감정 목적물 부재	완성도와 더불어 기성고 관점의 범위 산정

6. 토 의

앞서 대표적인 완성도 감정 시 문제점을 열거하였지만, 세부적으로는 여러 복잡한 문제에 봉착할 수 있다. 일부 문제는 발주사 및 개발사의 오해에서 비롯된다. 특히 소프트웨어 개발 프로젝트에 대한 발주사의 이해가 낮고, 또 개발사의 개발 숙련도가 다소 낮을 때 이런 일이 흔히 발생하는데, 이 절에서는 그중 대표적인 것 몇 가지를 논의하고자 한다.

일반적으로 웹 개발 업체에서 웹 프로젝트를 개발할 때에는 IA(information architecture) 혹은 WBS(work breakdown system)를 설계 문서로 간주하기도 한다. 그러나 IA는 메뉴 구조도에 해당하며 이는 설계 문서의 극히 피상적인 일부에 해당할 뿐이다. WBS는 작업을 나누어 배정한 것으로서, 아무리 크게 고려해도 작업 계획서 정도에 불과하다. 이는 개발사가 설계 문서 작성에 익숙하지 않은 상황에서 발생하는 착오이며, 필요에 따라 감정서를 통해 명확히 밝혀주는 것이 좋다.

윤영선의 연구[2]에 따르면 설계 단계의 산출물로는 다음과 같은 것이 포함된다. DB 설계 관련 내용은 생략하였다.

(1) 시스템 구조 설계
(2) SW 구조 설계
(3) 클래스 설계서
(4) 기능 설계서
(5) 화면 설계서
(6) 사용자 인터페이스 설계서
(7) 통합 테스트 시나리오
(8) 사용자 테스트 시나리오

이 중에서 IA는 기능 설계서(4)의 일부라고 볼 수 있을 것이다.

웹 개발사에 따라서는 퍼블리싱 소스를 개발 소스의 일부로 보는 경우도 있다. 그런데 사실 퍼블리싱 소스는 화면 형태만 그린 것으로서 화면 설계서(5)의 일부라고 간주할 수 있다. 왜냐하면 퍼블리싱 소스는 설계 단계에서 이러한 화면이 맞는지 사용자에게 확인하기 위한 코드이기 때문이다. 코드로 구현된 것이 아니라면 이는 페이퍼 프로토타이핑(paper prototyping) 단계[9]를 통해 수행될 수도 있다. 잘 알려진 것처럼 프로토타입은 실제 사용하기 위한 소프트웨어가 아니라 구현 가능성을 타진하기 위한 소프트웨어다.

그러나 이에 대한 분쟁이 첨예한 경우에는 HTML이나 CSS 부분으로 작성된 부분 외의 코드 부분을 구현 일부로 간주할 수 있을 것이다.

계속 개발이 진행된다면 퍼블리싱 소스의 코드 부분이 실제 구현 코드로 확장되어 사용될 수 있기 때문이다.

웹 시스템의 특정 기능이 구현된 코드를 판단하기 위해서는 프로그램 슬라이싱 기법[10]을 도입할 수 있다. 슬라이싱이란 프로그램 출력에 의존적인 프로그램 코드를 추출하는 기법으로서 프로그램 분석 및 최적화에 활용할 용도로 제안된 것이다. 소프트웨어 감정의 경우, 원하는 결과 산출에 기여하는 소스 코드를 추출하기 위해 슬라이싱 기법을 활용할 수 있다. 슬라이싱 기법을 도입할 때에는 기능별로 중복되어 산출되는 소스 코드를 구분하는 것이 필요하다.

7. 결 론

이 연구에서는 소프트웨어 완성도 감정에서 흔히 발생하는 문제점을 세 가지로 분류하고 각 경우에 관하여 해결 방법을 모색하였다. 완성도 감정의 경우, 감정 목적물의 형태가 매우 다양하게 제시될 수 있으므로 수행 환경을 구축하는 것이 어려울 수 있으며, 때론 감정 목적물에 소스 코드가 부족하거나 부재할 수도 있다. 또한, 감정 대상 소프트웨어의 구조가 매우 복잡한 형태일 수 있다. 이 논문에서는 이러한 경우에 대한 해법으로 유사도 기법 활용, 기성고 감정 병행, 데이터 모델 감정 등의 방법을 모색하였다.

소프트웨어 감정의 경우에 발주자와 개발사의 언어가 다른 경우가 많다. 이로 인해서 기획 단계를 설계 단계로 착각하거나 일부 설계 산출물을 구현 산출물로 오인하는 예도 있다. 소프트웨어 전문성을 체득하고 있는 감정인은 이를 명확히 적시해야 할 것이다. 이는 분쟁 양측 사이의 오해를 불식시키는 데에도 도움이 될 뿐 아니라, 소프트웨어 개발 과정을 명확히 함으로써 개발사에 가이드라인을 제시하는 효과를 줄 수 있을 것이라고 생각한다.

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (RS-2023-00242528)

참 고 문 헌

[1] K.-T. Kwon, “Overview of the Appraisal of Completeness for Software”, Journal of Software Assessment and Valuation, Vol. 11, No. 2, pp. 1-8, 2015.
http://www.i3.or.kr/html/paper/2015-2/(1)2015-2.pdf

[2] Y.-S. Yun, “Meaning and Computation of Completeness and Payment in SW Appraisal”, Journal of Software Assessment and Valuation, Vol. 15, No. 2, pp. 35-42, 2019.
http://dx.doi.org/10.29056/jsav.2019.12.05

[3] D. Kim, “A Study on the Need for Separation of Software Completeness Appraisal and Software Ready-made Appraisal”, Journal of Software Assessment and Valuation, Vol. 17, No. 2, pp. 11-17, 2021.
http://dx.doi.org/10.29056/jsav.2021.17.02

[4] E. W. Dijkstra, “On the Reliability of Programs”, Technical note (Transcribed by David J. Brantley),
https://www.cs.utexas.edu/users/EWD/transcriptions/EWD03xx/EWD303.html

[5] Korea Software Industry Association, The Software Cost Estimation in Software Projects (Revised 2022), Korea Software Industry Association, 2022,
https://www.sw.or.kr/site/sw/ex/board/View.do?cbIdx=276&bcIdx=53628

[6] N. Wirth, Algorithms + Data Structures = Programs, Prentice Hall, 1976. (ISBN: 0130224189)
[7] F. P. Brooks Jr., The Mythical Man-Month, Addison-Wesley, 1975. (ISBN: 9780201835953)
[8] A. Haleem, M. Javaid, and R. P. Singh, “An era of ChatGPT as a significant futuristic support tool: A study on features, abilities, and challenges”, BenchCouncil Transactions on Benchmarks, Standards and Evaluations 2 (2022) 100089, 2023. https://doi.org/10.1016/j.tbench.2023.100089
[9] R. Sefelin, M. Tscheligi, and V. Giller, “Paper Prototyping — What is it good for? A Comparison of Paper- and Computer-based Low-fidelity Prototyping”, In Proceedings on CHI ’03 Extended Abstracts on Human Factors in Computing Systems, pp. 778 - 779, 2003. https://doi.org/10.1145/765891.765986
[10] M. Weiser, “Program Slicing”, IEEE Transactions on Software Engineering, Vol 4. pp.352-357, IEEE, 1984. https://doi.org/10.1109/TSE.1984.5010248

저 자 소 개

우 균(Gyun Woo)

1991.2 KAIST 전산학과 학사
1993.2 KAIST 전산학과 석사
2000.2 KAIST 전자전산학과 박사
2000.3-2004.2 동아대학교 교수
2004.3-현재 : 부산대학교 교수
<주관심분야> 프로그래밍 언어, 컴파일러, 함수형 언어, 프로그램 분석, 프로그램 시각화, 프로그래밍 교육