The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 21, No. 2, pp.79-88, Apr. 30, 2021. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2021.21.2.79
JIIBC 2021-2-12

과학용 소프트웨어 구현을 위한 RE와 TDD기반 V&V 개발 프로세스 제안

Suggestion of RE and TDD-based V&V Development Process for Scientific Software Implementation

이재홍[*], 김덕수[**], 김승희[***]

Jae-Hong Lee[*], Duksu Kim[**], Seung-Hee Kim[***]

요 약 과학용 소프트웨어는 전문가 수준의 깊이있는 도메인 지식 요구 등과 같이 고유한 특성들로 인하여 전통적인 응용소프트웨어와는 다른 개발 프로세스가 요구된다. 본 연구에서 우리는 과학용 소프트웨어 개발을 위하여 RE(Reverse Engineering)과 TDD(Test-Driven Development)에 기반한 V&V 개발 프로세스를 제안하였다. 또한 실제 프로젝트에 가상 시나리오를 구성하고 적용하여 절차를 검증하고 정교화 하였다. 과학용 소프트웨어 개발을 목적으로 제안된 본 연구의 개발 프로세스는 고품질, 고 신뢰성을 제공할 수 있는 소프트웨어 개발 및 적용에 기여할 것이다. 또한 본 연구는 과학용 소프트웨어 개발 및 연구의 저변 확산의 계기가 될 것으로 기대된다.

Abstract Scientific software requires a development process different from conventional application software due to its unique characteristics, such as expert-level deep domain knowledge requirements. In this study, we proposed a V & V development process based on RE (Reverse Engineering) and TDD (Test-Driven Development) for software development for science. We also configured a virtual scenario for the actual project, applied it, checked the procedure, and refined it. The development process of this study, suggested for the purpose of developing scientific software, will contribute to the development and application of the software that can provide high quality and high reliability. And This study is expected to serve as an opportunity for the development of scientific software and the spread of research.

Key Words : Scientific Software, Reverse Engineering, TDD, V&V, software development process

Ⅰ. 서 론

과학용 소프트웨어(scientific software)는 대규모 컴퓨팅 구성요소가 포함된 소프트웨어로 의사결정지원을 위한 데이터를 제공하는 소프트웨어로 정의된다.[1,2,3]한편, Kelly et al. 는 과학용 소프트웨어의 유형을 기후 모델과 같이 과학적 목표를 달성하기 위해 작성된 최종 사용자 응용 소프트웨어와 MATLAB용 자동화 소프트웨어

[*]학생회원, 한국기술교육대학교 컴퓨터공학원
[**]정회원, 한국기술교육대학교 컴퓨터공학부
[***]정회원, 한국기술교육대학교 IT융합SW공학과, 교신저자
접수일자 2021년 3월 15일, 수정완료 2021년 3월 30일
게재확정일자 2021년 4월 9일

Received: 15 March, 2021 / Revised: 30 March, 2021 / Accepted: 9 April, 2021
[***]Corresponding Author: sh.kim@koreatech.ac.kr
Dept. of IT Convergence software Engineering, Korea university of Technology & Education, Korea

테스트 도구와 같이 과학적 모델 및 과학적 코드 실행을 표현하는 코드 작성을 지원하는 도구로 분류[1,4]하였다. 최근 기계, 항공, 우주 분야에서는 전산학에 기초한 전산역학 연구를 활발히 진행중이며, 최근 사회과학연구에서는 데이터마이닝, 머신러닝을 기반으로 하는 연구가 진행 중이며, 생명과학의 단백질, DNA/RNA의 구조 연구도 컴퓨터 시뮬레이션을 기반으로 연구되고 있다[5,6,7].

이와 같이 현대 과학, 공학 분야는 복잡하고 높은 연산량의 수식을 풀고 연구의 데이터 수집, 가공, 검증, 시뮬레이션 등을 위해 과학용 소프트웨어를 활용하며, 과학용 소프트웨어로부터 생성된 자료는 연간 간행물 등의 증거 자료로써 사용된다[8]. 또한 과학용 소프트웨어는 주로 사용하는 데이터가 응용소프트웨어에 비해 크고(GB~TB) 사용하는 컴퓨터의 성능 또한 일반 PC에 비해 뛰어나며, 프로그램에 관한 신뢰성, 응용성이 높은 경우가 다수로, 높은 정밀도와 연상량이 요구되어 클러스터, GPGPU[9], FPGA[10]와 같은 고성능 컴퓨팅 장치가 주로 사용되기 때문에 컴퓨터과학의 고성능컴퓨팅(HPC)분야와 밀접한 연관을 갖는다.

이러한 다양한 배경들은 일반 컴퓨팅과 과학용 컴퓨팅 사이에 넓은 틈을 만들고 있으며 다수의 연구진들이 이러한 차이를 지적[8,11,12,13,14]하고 있다.

그 결과 전사적 자원관리와 같은 전통적인 소프트웨어 개발자와 과학용 소프트웨어 별 도메인에 대한 지식과 특성을 알아야 하는 과학적 소프트웨어 개발자 사이에는 아이디어 교환을 단절[14]시키는 가능성을 높이고 있다. 또한 과학용 소프트웨어가 과학자 입장에서 사전에 요구사항을 정확히 정의하기 어렵거나 불가능[14]할 뿐만 아니라 개발자 입장에서 깊이있는 도메인 지식의 한계로인하여 과학자와 과학용 소프트웨어 엔지니어 간 협업을 방해하는 요인이 된다. 뿐만아니라 Jo Erskine Hannay et al.[14]에 따르면 과학 연구에 종사하는 과학 또는 공학자가 소프트웨어 테스트에 대한 이해가 낮다는 결과를 보이고 있다.

따라서, 과학용 소프트웨어 개발 프로젝트의 성공적 완수를 위해서는 이러한 위험을 예방·회피·감소시킬 수 있는 소프트웨어 개발 프로세스가 요구된다. 즉, 과학 연구에 사용하는 알고리즘, 수식, 용어를 사용하기 때문에 기존 소프트웨어 프로젝트와는 다른 접근법이 요구되며 더욱 섬세한 소프트웨어 품질 및 위험관리가 필요하다. 그러나 과학용 소프트웨어를 개발과 관련한 대다수의 연구는 애자일 프로세스를 중심으로 하는 방법론 중의 하나로 다루어지거나, 사례 연구, 과학용 소프트웨어 개발 환경 및 활용 등을 주제로 하고 있어 과학용 소프트웨어 개발 측면의 특성을 반영한 프로세스 연구는 매우 미흡한 상황이다.

본 연구에서는 레거시 시스템이 존재하는 과학용 소프트웨어 개발 시 보다 효율적인 고품질의 소프트웨어 개발에 적용할 수 있는 RE(Reverse Engineering)와 TDD(Test driven development)를 활용한 소프트웨어 개발 프로세스를 제안한다.

이를 위해, 2장에서는 과학용 소프트웨어 관련 선행연구와 TDD에 대한 사전 연구를 수행하며, 3장에서는 본 연구에서 제안하는 개발 프로세스를 상세히 설명한다. 4장에서는 실제로 수행된 개발 프로젝트에 대하여 가상의 시나리오를 적용하는 실험을 통해 프로세스를 검증한다. 5장에서 결론과 향후 과제를 살펴본다.

II. 선행 연구

1. 이전 연구

과학용 소프트웨어에 관련하여 실제 사례를 분석한 연구에 대하여 살펴보았다. Jo Erskine Hannay et al.[14]는 실제 과학연구에 종사하는 과학자 및 공학자들에게 과학 소프트웨어 개발에 관해 설문조사를 하였다. 연구결과로 많은 과학자와 공학자들이 과거보다 현재 과학 소프트웨어의 연구에 더 많은 시간을 소비하고 있으며, 소프트웨어 테스트가 중요성을 느끼지만 소프트웨어 공학적인 테스트에 관한 이해가 부족하였고, 소프트웨어의 규모가 큰 경우 소프트웨어 전문가의 중요성을 높게 보았다. Adel Taweel.[15]는 국제 과학 연구 및 소프트웨어 엔지니어링 프로젝트에서 얻은 경험을 바탕으로 협업 프로세스와 소프트웨어 개발 진행을 방해할 수 있는 요인을 제시하며 그에 관한 방법과 메커니즘을 제시하고 있다. B. C. Jang and S. C. Kim 등[16]은 MATLAB GUI를 이용하여 U자형 금속 벨로우즈 설계 SW를 개발하였다. 이와 같이 통계와 실제 사례를 제시한 연구가 다수 존재하나, 직접적인 과학용 소프트웨어 개발에 관한 연구는 부족하다.

Rasmus H. Fogh et al.[17]은 과학적 데이터 모델링과 자동화 소프트웨어 개발에 관한 프레임워크를 제시하였고, Chris A. Mattmann et al.[18]은 과학 어플리케이션의 방대한 데이터를 분산하여 처리할 수 있도록 소프트웨어 프레임워크 OODT를 개발하였다. D. Glez-pena et al.[19]은 생물의학에서 생의학/중개의학

연구에 필요한 오픈소스 자바 어플리케이션 프레임워크 AIBench를 개발하였다. Markus Geimer et al.[20]는 클러스터를 이용한 과학 소프트웨어를 쉽게 빌드하고 설치할 수 있는 프레임 워크를 제시하였다. Anoton A. Goloborodko et al.[21]은 단백체학 데이터 분석을 수행하는 파이썬 프레임워크를 개발하였다. 이와 같이 과학 소프트웨어 개발에 도움이 되는 프레임워크, 라이브러리 등을 개발한 연구가 있으나, 이 연구들은 라이브러리의 활용방법을 서술한 것으로 개발 방법론과 연관성을 찾기 어렵다.

Ronald F. boisvert et al.[22]은 과학용 소프트웨어 관리 프레임워크를 제시하였으나 연구가 오래되어 현대 과학 소프트웨어와는 많은 차이점을 보이고 있다.

Chris Morris와 Judith Segal[23]는 분자생물학 분야 소프트웨어 개발에 관한 문제, 소프트웨어 개발자가 분자생물학과 일반적인 물리학 소프트웨어 개발의 차이점을 제시하고 있다.

과학용 소프트웨어 개발에 개발방법론을 적용한 연구로 M. T. Sletholt el al.[24]은 과학용 소프트웨어 개발에 애자일 방법론을 적용한 사례와 관련한 문헌 연구를 수행하였다. 애자일 방법론이 과학용 소프트웨어 개발에 좋은 영향을 줄 수 있고 특히, 개발 규모가 작은 경우에 영향이 크다는 연구 결과를 나타냈다. M. T. Sletholt et al.[25]은 과학용 소프트웨어에 애자일 방법이 과학용 소프트웨어에 어떤 영향을 주는지 연구하였다. 또한 S. K. Yoo and S. Y. Cho[26]는 하드웨어 제어를 포함한 시스템 소프트웨어의 품질평가에 활용할 수 있는 ISO25000 기반의 SW 품질평가 모델 평가 지표를 고안하였다.

이렇듯 과학용 소프트웨어를 애자일 방법론으로 적용한 사례가 있으며, 과학용 소프트웨어와 관련된 연구가 다수 존재하나, 과학용 소프트웨어에 특화된 개발 프로세스나 방법론에 대한 연구는 매우 미흡하다.

2. TDD 개요

TDD는 TFD(Test First Development)와 refactoring을 조합한 소프트웨어 개발 절차[27]로, 독립 실행형(Stand-alone) 환경으로써 Extreme Programming(XP) 방법론에서 널리 채택되고 있으며, 자주 “red-green-refactor cycle”[28]로도 불리어 진다. TDD는 소프트웨어 개발자가 테스트 코드를 작성하기 전에 자동으로 실행할 수 있는 테스트 케이스를 작성하는 반복적이고 점진적인 접근 방식[29] 으로 이를 통해 소프트웨어의 기능들이 구축되고 개선된다. 구체적으로는 Test-Production coding-refactor 주기를 통해 시스템과 세부 설계가 수행된다.

일반적인 개발 절차와 비교하면, 개발자는 코드를 먼저 작성한 후 개발 마지막 단계에 주로 테스트를 수행한다. 그러나 TDD 모델은 테스트를 생성하고 그 이후 테스트를 위한 코드를 개발하는 절차로 수행된다. 이는 실제 코드를 작성하기 전에 코드 작성자가 전체적인 설계나 요구 사항에 대해 생각하는 시간을 주는 효과가 있다.

가. Test Driven Development cycle

Step 1. 테스트 설계 및 작성: TDD의 시작으로 새로운 기능에 대한 테스트 케이스를 작성한다. 이를 위해 개발자는 먼저 유저 스토리나 사례들을 통해 기능명세(specification)와 요구 사항을 상세히 이해해야 한다. 또한 구현된 기능의 우선순위도 사전에 정의해야 하는데 이를 위해 CV (Cumulative Voting) 방법을 사용하여 중요도, 복잡도, 종속성과 같은 기준을 적용할 수 있다.[30,31]

Step 2. 테스트 실행 및 새 테스트 실패 확인(red) : 이 단계에서 개발자는 테스트케이스를 작성하고 실행함으로써 테스트를 수행한다. 아직 구현한 코드가 없으므로 테스트는 당연히 실패하는데, 이 실패는 자동화된 테스트 프레인워크가 정상적으로 작동되고 있음을 반증하며, 새로운 테스트케이스에 대하여 오류가 있을 경우 이를 정상적으로 발견할 것이라는 확신을 주는 것과 같다. Step 3. 새 테스트를 만족하는 충분한 구현 코드 작성: 개발자는 테스트를 통해 기능 설계에 대한 작동 원리를 이해하게 되어 새 테스트를 만족시킬 수 있을 만큼 충분한 수준의 구현 코드를 작성 및 추가한다. 완벽하지 않은 테스트 케이스가 통과를 하더라도 본 단계의 목적상 크게 상관없다.

Step 4. 모든 테스트를 실행하고, 필요 시 모든 테스트가 통과할 때 까지 3회 정도 반복 (green) :

기능이 의도한 설계대로 수행되는 것이 확인되면, 개발자는 새로운 코드를 구현하고, 이 때 마다 테스트 CASE 툴을 사용하여 테스트를 다시 실행할 수 있다.

Step 5. 코드 구조 개선을 위한 리팩토링 : TDD에서는 늘어나는 코드 베이스를 정리하는 작업이 때때로 또는 주기적으로 수행된다. 즉, 이전 단계들이 코드 구현에 집중하였다면, 본 단계에서는 코드 효율성을 높이기 위한 작업으로 소스 코드의 중복을 최소화하면서 새로운 기능들을 추가하는 프로그램 내부 구조 개선에 집중한다.

Step 6. 반복 테스트 및 통과 확인: 리팩토링 후 테스트를 반복 실행하여 모든 테스트를 통과하는지를 확인한

다. 모든 기능을 커버할 수 있는 수준으로 TDD 주기를 반복한다.

TDD와 다른 소프트웨어 개발 모델과의 상세한 비교는 Kamini Bajaj et al.[30]의 연구를 참조하기 바란다.

Ⅲ. RE와 TDD를 적용한 V&V 소프트웨어 개발 프로세스

1. 제안 프로세스 개요

기존 레거시 시스템의 활용을 극대화 하고, 고품질을 소프트웨어 구현을 달성할 수 있도록 그림 1과 같이 RE(Reverse Engineering)과 TDD를 활용한 V&V 개발 프로세스를 제안한다.

과학용 소프트웨어 개발에서 가장 중요한 부분은 품질로, 본 절차는 기존의 V&V 절차의 수행방법으로 TDD를 접목함으로써 품질 제고를 더욱 높일 수 있도록 설계하였다. 즉, 1차 작업은 기존 레거시 시스템으로부터 역공학을 통해 UML과 기초 소스 코드를 생성하고, 생성된 UML 보정을 거쳐 상세 설계를 수행한다. 최종 설계 내용은 고객의 확인을 득함으로써 다음 단계인 TDD 구현 단계로 진행된다.

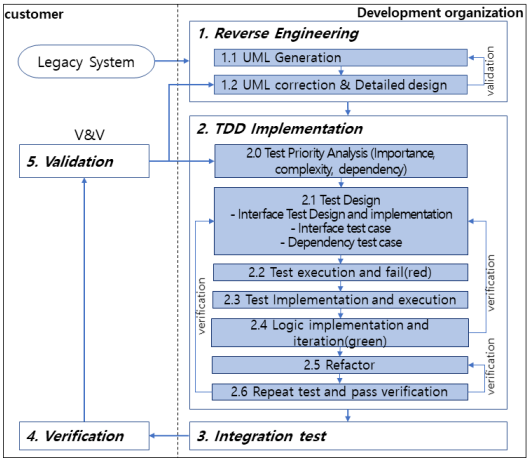


그림 1. RE와 TDD를 적용한 소프트웨어 개발절차
Fig. 1. Software development procedure applying RE and TDD

TDD 구현 단계는 설계 단계에서의 분석 결과를 반영하여 우선순위에 따라 수행한다. 절차는 Test 설계 및 작성, Test 실행 및 실패, 구현 코드 작성, 테스트 반복 및 pass, refactor, 반복 테스트 및 통과 확인의 절차를 반복하며 완성된다. 모든 테스트 및 구현이 통과되면 통합 테스트를 통해 최종 검증하고 최종 개발된 서비스는 고객의 확인을 거침으로써 완료된다.

2. 제안 프로세스 상세

레거시 시스템은 고객사에서 기존에 사용하는 과학용 소프트웨어의 소스 코드를 포함하고 있는 응용서버를 의미한다. 고객은 개발사에게 요구사항과 함께 기존 소프트웨어의 소스코드 일부 혹은 전체를 전달한다. 일반적으로 최종 프로그램 코드가 체계적으로 잘 관리되지 못한 경우에는 전사적 보안 정책의 범위 내에서 전체 메뉴에 대한 읽기 권한을 갖는 계정을 신규로 생성하여 제공할 수 있다. 개발이 진행되는 동안 레거시 시스템은 역공학을 통한 정보와 자료, 설계 문서를 생성할 수 있는 매개체로서의 역할을 하며 기능 구현 시 결과 검증에 이용된다.

가. Reverse Engineering

Step 1. UML 생성 : 개발사는 레거시 시스템에서 고객사가 요구하는 과학용 소프트웨어의 UML 다이어그램을 도출한다. 이 단계에서 생성된 UML 다이어그램은 요구사항에 관한 도메인 지식이 포함되어 있기 때문에, 개발사가 도메인 지식이 부족하더라도 소프트웨어를 복제 및 변환된 저작물을 통해 설계하고 개발할 수 있게 한다. 또한 일반적인 산출물에 존재하지 않는 알고리즘이나 현행화 되지 않은 설계 내용까지도 모두 추론할 수 있도록 해 준다. 그러나, 동일한 이유로, 사용하지 않는 로직에 대한 정보도 포함하기 때문에 UML보정 및 상세 설계를 완료하면 고객을 통한 확인이 반드시 요구된다. 또한, 암호화 기술이나 해킹 기술 등과 관련해서는 사전에 고객과의 충분한 검토를 통한 설계가 이루어지도록 한다.

Step 2. UML 보정 / 상세 설계 : 생성된 UML 다이어그램, 이전 회차에서 수행된 설계 및 고객사의 요구사항에 맞추어 소프트웨어를 설계한다.

나. TDD 구현

TDD 구현을 위해 가장 먼저 고려되어야 할 것은 테스트 우선순위를 정하는 것이다. 이를 위해 요구사항정의서 등 요구사항의 우선순위를 파악할 수 있는 문서를 확인해야 한다. 그 외 중요도, 복잡도, 종속성 등을 파악하여 테스트 우선순위를 결정하며, 이는 반드시 고객의 확인이 요구된다.

Step 1. Test 설계 및 작성 : 정해진 우선순위에 따라 Test를 설계하고, 그 설계를 위한 테스트 케이스를 구현하는 단계이다. 테스트 케이스는 각 프로그램 단위별 레거시 시스템의 입출력과 요구사항을 반영한다. Interface test를 우선적으로 권고하며, 이와 관련된 인터페이스 테스트 케이스와 종속성 테스트 케이스를 작성할 것을 제안한다. 그 외 기능에 대해서는 분석된 우선순위에 다라 테스트를 설계한다. 이는 본 연구의 테스트 케이스를 통해 검증한 결과를 반영한 것으로 TDD에서 테스트 케이스를 구현하고 내부 동작을 구현하고 테스트를 하기 위해서는 인터페이스가 미리 구현이 되어야 하기 때문이다.

Step 2. Test 실행 및 실패

테스트에 알맞은 테스트 케이스를 개발하여 실행하고 실패를 도출함으로써 테스트 프레임워크의 정상 작동을 확인하며, 새로운 테스트 케이스에 대한 테스트 환경을 검증한다. 인터페이스와 종속성에 따라 테스트 실행 순서를 결정하여 순서대로 실행한다.

Step 3. Test 구현 및 실행 : 테스트 케이스에서 설계된 단위 테스트를 통과할 수 있는 테스트용 소스코드를 구현 한다.

Step 4. 로직 구현 및 반복 : 로직이 구현된 모든 단위 모듈은 Step 3에서 생성된 모든 테스트 케이스가 모두 통과할 때까지 반복하며 구현되고 테스트가 실시된다. 즉, 앞서 만들어진 테스트 케이스를 구현된 모듈이 모두 통과해야한다. 만약 문제가 발견되면 테스트 설계부터 재 수행할 수 있도록 Step 1 단계로 이동한다.

Step 5. Refactor : 구현 테스트를 통과한 기능은 효율성 측면에서 재구조화 된다. 특히 과학용 소프트웨어의 개발 시에는 중복성 배제, 실행 성능과 정밀도 등을 모두 고려하여 refactoring 지표를 설정함으로써 고품질, 신뢰도와 최적화를 위한 활동을 수행한다.

Step 6. 반복 테스트 및 통과 확인 : 기능에서 요구되는 모든 테스트 케이스에 대해 모두 통과를 하도록 TDD 주기를 반복수행 한다. 이때 fail이 발생하는 경우 refactor 단계로 이동하여 구현 코드의 개선 작업을 수행한다.

다. 통합테스트

TDD를 통해 각 단위 모듈을 모아 하나의 소프트웨어로 만들고 통합된 기능에 문제가 없는지 테스트한다.

라. 검증(verification)

구현된 알고리즘이 제대로 되었는지 고객과 함께 확인한다. 일반적인 verification은 개발 조직 차원에서 검증 작업이 진행되나 과학용 소프트웨어라는 점을 감안하여 공동 진행한다.

마. 확인(validation)

프로그램이 제대로 동작하고 문제가 없는지 확인한 후 요구사항을 전달한다. 이후 개발사는 다시 고객사의 요구사항을 기반으로 우선순위에 따라 Test 설계 및 작성을 반복한다. 만약 고객 확인 결과 업무 분석 상에 문제가 발견되었다면 역공학 단계로 되돌아가서 설계를 재검토하거나 최악의 경우 UML 생성을 통해 분석 내용을 재확인한다. 역공학 작업이 완료되면 역시 고객의 확인을 통해 TDD 구현 활동을 진행한다.

Ⅳ. 가상 프로젝트를 통한 검증

과학용 소프트웨어 구축 사례를 본 개발방법론에 대입하고 가상의 시나리오를 통해 상세 절차를 검증 및 개선하였다. 본 사례는 총 3회차의 프로세스 반복 시나리오로 구성되어있다.

1. 프로젝트 배경 및 개요

고객사에서는 MATLAB 언어를 이용하여 컴퓨터 생성 홀로그래피(Computer Generated Holography, 이하 CGH) 소프트웨어를 개발하였다. 당초, 파동광학 연구의 한 분야인 CGH 기술을 이용하여 빛의 파동을 시뮬레이션하기 위한 목적으로 과학용 소프트웨어를 개발하였으나 GB~TB급의 복소수 행렬을 다루기 때문에 필수적으로 요구되는 높은 연산속도 문제로 인하여, 프로그램의 컴퓨팅 자원의 사용률이 기대에 미치지 못하여 연산 시간이 너무 오래 걸리는 문제가 발생하였다. 이를 개선하기 위해 GPU 가속 시스템을 활용할 수 있는 컴퓨터 생성 홀로그램 소프트웨어를 다시 개발하게 되었다. 그러나 개발을 담당할 개발사는 홀로그램에 관한 도메인 지식이 미흡하여 CGH 알고리즘을 개발하는데 난관에 부딪치게 되었다. 이러한 문제를 해결하고 과학용 소프트웨어라는 특수성을 반영하여 RE & TDD 개발 프로세스를 적용하여 3 회전의 개발 반복을 통해 개발을 완료하였다. 개발 언어로는 CUDA C/C++ 을 사용하였다.

2. Cycle 1 : CGH 알고리즘 구현

가. Reverse Engineering

역공학을 이용하여 MATLAB기반으로 개발된 레거시 시스템으로부터 UML을 도출하였다. 그러나 레거시 시스템이 클래스 기반으로 개발되지 않았기 때문에 개발자가 직접 각 함수의 호출순서를 파악하여 UML다이어그램을 보정하였다. MATLAB과 C/C++ 간의 차이를 보정하기 위하여 MATLAB에 구현되어있는 함수를 C/C++에서 외부 라이브러리를 사용하도록 설계하였다.

사용자 인터페이스는 개발사 스스로 설계와 구현이 가능하므로 레거시 시스템에 맞추지 않고 새로 구현을 진행하는 것으로 결정하고 설계를 진행하였다.

나. TDD 구현

상세 설계 결과를 바탕으로 테스트 케이스를 설계한다. 우선순위에 따라 1회차 프로세스에서는 CGH 알고리즘을 구현하는 것을 그 범위로 하였다.

테스트 케이스는 레거시 시스템의 입출력을 그대로 이용 가능하므로, 개발 프로그램과 레거시 시스템에 동일한 입력을 넣었을 때 동일한 출력이 나오는지를 검사하는 것으로 테스트를 설계하고 테스트를 실행하였다.

테스트 케이스로부터 테스트가 자동으로 수행되는 것을 확인한 후 해당 입출력을 위한 프로그램을 구현하였다. 구현 코드 작성 시 테스트 케이스 루틴을 통과하는 최소한의 코드 정도로 프로그램을 구현하였으며, 테스트를 반복하여 각 모두 테스트 케이스가 모두 성공적으로 완료되었다. 성능을 보다 높이고, 기존 코드와의 중복성을 배제하기 위한 Refactor 작업을 반복하여 각 모듈 별 TDD 구현을 완료하였다.

다. 통합 테스트, 검증 및 확인

TDD 방식의 구현이 최종 완료된 각 모듈을 통합하여 통합테스트를 진행하였다. 고객사에서는 CGH 알고리즘이 올바르게 구현되었는지 개발사와 함께 코드 검증을 수행하였다. 검증이 완료된 후 cycle 1에서 개발된 소프트웨어를 직접 사용해보면서 레거시 시스템과 비교를 통해 CGH가 제대로 생성되는지 확인하는 절차를 수행하였다.

3. Cycle 2 : GPU 알고리즘 구현

1회차에서 개발된 CGH 알고리즘을 활용하여 고객의 두 번째 요구사항인 GPU 알고리즘 개발 요구사항 구현을 위하여 RE & TDD 개발 절차를 수행하였다.

가. Reverse Engineering

1회차 개발 프로세스에서 고객사가 CGH 생성이 정상적으로 작동하는 것을 확인한 후, GPU 하드웨어를 사용하도록 UML 보정 및 상세 설계를 수행하였다.

나. TDD 구현

GPU 하드웨어를 이용하도록 하기 위한 테스트를 설계하고 이에 맞는 테스트 케이스를 새로 설계하였다. 설계에 따라 1회차와 동일한 알고리즘으로 구현 코드를 작성하여 정상적인 실패를 유도하였다. 정상적인 테스트 케이스를 적용하여 구현 코드를 재작성하고 실행을 통과한 알고리즘에 대해 refactor를 실시하였다. refactor의 주요 목적은 GPU 하드웨어를 사용함으로써 연산 성능의 개선에 집중하도록 다양한 구현 코드를 반복 실험하는 것이었다.

그러나 검증단계에서 GPU 하드웨어를 사용하기 위한 외부 라이브러리에서 연산 오차가 달라 반복 테스트에서 통과 기준을 만족하지 못하여 테스트 케이스는 fail이라는 결과를 낳았다. 이후 반복되는 refactor작업을 통해 허용 오차 범위를 개선하여 테스트 기준에 적합한 TDD 기반 프로토타입이 완성되었다.

다. 통합 테스트, 검증 및 확인

프로토타입에 대하여 고객사에서 검증한 결과 문제 가 없는 것으로 통과되었으나 확인과정에서 문제가 발생하였다. 즉, 홀로그램 생성 과정에서 낮은 연산 정밀도로 인해 고해상도 홀로그램 생성이 불가능한 문제가 발견되었다. 그러나 이는 현재 고객사 시스템 자체의 문제로 확인되어 고객은 GPU 알고리즘의 pass로 승인하였다.

4. Cycle 3 : 레거시 시스템의 버그 clear

Cycle 2에서 발견된 홀로그램의 정밀도와 해상도 문제를 비롯하여 기존 레거시 시스템이 가지고 있는 버그 문제를 제거하기 위한 활동을 수행하였다.

가. Reverse Engineering

신규 알고리즘 운영에 영향이 있을 것으로 예상되는 1회 및 2회차에서 발견된 기존 레거시 시스템의 버그 부분들에 한하여 UML을 재생성하는 한편, UML을 보정 및 상세 설계를 재실시하였다. 효율적 작업 수행을 위해 고객사에서 선별하여 제공한 문제가 있는 모듈을 중심으로 높은 정밀도로 연산하도록 설계를 보정하였다.

나. TDD 구현

정밀도 연산테스트 케이스 설계를 통해 구현 코드를 작성한 후 해상도와 정밀도와의 상관성을 고려하며 refactor를 집중적으로 반복하였다.

표 1. refactor 반복을 통해 완성된 홀로그램 이미지
Table 1. Holographic image completed through refactor repetition

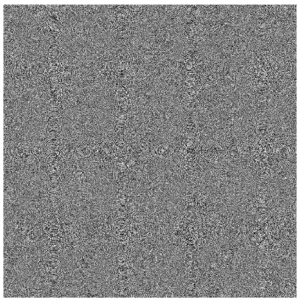
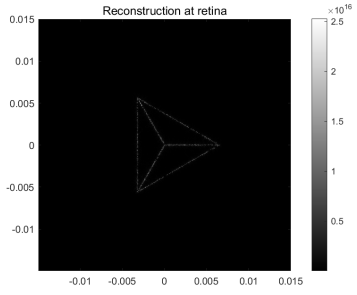
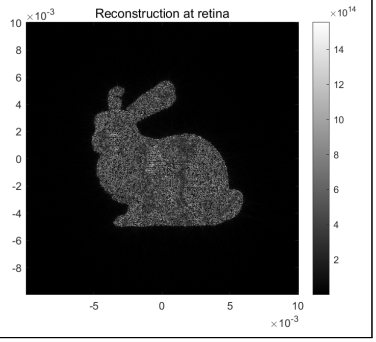
Refactor	Generated image
(a) amplitude hologram	
(b) hologram reconstruction of tetra	
(c) hologram reconstruction of bunny	

표 1의 (a)는 홀로그램의 한 픽셀을 그레이 스케일 값(0~255)으로 생성한 것으로, 평균 제곱 오차(MSE;mean square error) 값이 개선되면서 반복 횟수에 따라 그림(b), 그림(c) 과정과 같이 정밀도와 해상도가 향상되어 정상적인 홀로그램 관측 이미지가 도출됨을 알 수 있다. 반복 회차에 따른 실제 보정 과정의 MSE 변화 추이도 그림 2와 같이 개선되는 것을 알 수 있다.

다. 통합 테스트, 검증 및 확인

정밀도와 해상도 간 발생된 홀로그램 버그 문제를 비롯하여 기존 레거시 시스템이 가지고 있는 문제가 모두 제거되었다. 고객은 신규로 개발된 알고리즘을 활용하여 정밀도와 해상도를 모두 만족하는 홀로그램(a)과 홀로그램 관측 이미지(b), (c)가 도출됨을 확인한다.

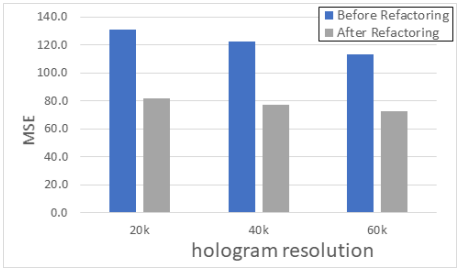


그림 2. refactoring 전·후 홀로그램의 Mean-Square-Error 비교
Fig. 2. Mean-Square-Error comparison of the hologram before and after refactoring

4. 제안 개발 프로세스의 시사점

가. 기존 소프트웨어 개발절차와의 차별성

본 연구에서 제안하는 TDD 기법 적용에 있어서의 특징은 우선 Test 설계 시 중요도, 복잡도, 종속성 등을 반영하는 절차를 명시적으로 절차화 하였으며, 그와 별도로 테스트를 위하여 Interface 테스트 케이스를 가장 먼저 설계하고 테스트 하도록 제시하였다는 점이다.

둘째, 기존 TDD와 비교하면 2.3 Test Implementation and execution, 2.4 Logic implementation and iteration(green)으로 단계를 명확히 구분하여 절차화 하였다는 것이다. 이는 테스트가 프로그램 단위별 이루어지기 때문에 각 단위별로 테스트 케이스와 로직 구현이 가능하다. 따라서 프로그램 전체의 테스트 케이스를 구현하는 것이 아닌, 프로그램 단위 하나를 테스트하고 구현하는 것이 가능하다. 이러한 점을 반영하여, 2.3 Test Implementation and execution, 2.4 Logic implementation and iteration(green)으로 단계를 구분하였다.

셋째, 과학용 소프트웨어 개발프로세스에서 고객사의 V&V를 제안하였다. 본래 개발 프로세스에서 검증은 개발사의 프로세스이지만 과학의 도메인 지식이 없는 개발사는 검증에 실패할 가능성이 있다. 따라서, 도메인 지식을 갖춘 고객사에서 검증을 하도록 하는 개발 프로세스를 제안하였다.

나. 개발 방법론의 한계

레거시 시스템으로부터 소스코드와 library를 통해 설계가 구현되거나 코드가 복제되는 본 절차의 특성상 버그가 개발 프로세서에 그대로 반영된다는 것이다. 이는 테스트 설계를 작성하고 테스트 구현 과정을 반복하더라도 레거시 시스템을 기반으로 테스트케이스를 구성할 경우에 버그를 찾을 수 없게 만드는 근본적인 문제가 발생한다. 이는, 고객사가 레거시 시스템의 버그를 인지하지 못하는 경우 고객사의 검증과 확인에 의존할 수밖에 없어 개발사가 새로운 시스템 구현 시 버그의 trouble shooting에 큰 부담으로 작용할 수 있다.

따라서 본 절차를 적용하는 과학용 소프트웨어를 포함한 고품질 소프트웨어 개발 사업 시 요구분석 단계에서 반드시 레거시 시스템이 갖고 있는 근본적으로 갖고 있는 버그(Fundamental bug)를 사전에 조사하는 절차를 진행할 것을 권고한다.

V. 결 론

본 연구는 요구사항 분석이 힘들고 매우 높은 수준의 프로그램 정확도가 요구되는 과학용 소프트웨어 개발을 위하여 역공학을 이용한 설계와 TDD를 활용한 개발 프로세스를 제안하였다. 또한 실제 과학용 소프트웨어 개발 프로젝트 사례에 가상의 시나리오를 적용함으로써 프로세스의 검증을 통해 문제점들을 개선하여 세부 절차를 정의하였다.

본 연구는 첫째, 과학용 소프트웨어 구현을 위하여 역공학과 TDD를 적용한 소프트웨어 개발 프로세스를 제안한 첫 번째 연구일 뿐만 아니라 V&V 프로세스 내에서 테스트와 요구사항 설계에 대한 해결책을 함께 제시함으로써 고품질 고신뢰성을 요구하는 과학용 소프트웨어 개발에 보다 최적화된 방법을 제안하였다는데 그 의의가 있다. 또한, 기존에 수행한 프로젝트 주제를 활용하여 가상의 시나리오를 구성하여 이를 기반으로 검증을 수행함으로써 실무적 활용성을 높였다.

현재는 공학 및 과학 연구자가 다루기 힘든 과학용 소프트웨어의 특성으로 인하여 소프트웨어 개발 시 연구 주체가 소프트웨어 전공자가 아닌 과학 종사자가 대다수이지만 향후 과학용 소프트웨어의 개발 요구가 가속될 것임을 고려할 때 본 연구는 소프트웨어 개발자와 과학자, 고객 간 원활한 소프트웨어 개발 협업을 촉진하는데 기여할 뿐만 아니라 미흡한 상태에 있는 과학용 소프트웨어 연구 및 개발을 위한 저변을 확산시킬 수 있는 계기가 될 것으로 사료된다.

References

[1] U. Kanewala and J. M. Bieman, "Testing scientific software: A systematic literature review", Information and software technology, Vol. 56, No. 10, pp. 1219-1232, 2018. DOI:https://doi.org/10.1016/j.infsof.2014.05.006

[2] K. Kreyman, D. L. Parnas, S. Qiao, "Inspection procedures for critical programs that model physical phenomena", CRL Report No. 368, McMaster University, 1999.

[3] R. Sanders, D. Kelly, "The challenge of testing scientific software", in: Proceedings Conference for the Association for Software Testing (CAST), Toronto, pp. 30-36. Jul. 2008.

[4] D. Kelly, S. Smith, N. Meng, "Software engineering for scientists", Comput. Sci. Eng., Vol. 14, No. 5, pp. 7-11, 2011.

[5] M. Suh, J. Yoo, S. Han, "Development framework of application software for chemical process plants", The Korea Contents Society conf., Vol. 5, No. 1, pp. 490-494, 2007.

[6] J. H. Moon, S. S. Shin, S. K. Choi, S. J. Cho and E. J. Rho, "Development of UAV Flight Control Software using Model-Based Development (MBD) Technology", Journal of the Korean Society for Aeronautical & Space Sciences, Vol. 38, No. 12, pp. 1217-1222, 2010.

[7] P. Librado, and J. Rozas, "DnaSP v5: a software for comprehensive analysis of DNA polymorphism data", Bioinformatics, Vol. 25, No. 11, pp. 1451-1452, 2009.

[8] R. Sanders, D. Kelly, "Dealing with risk in scientific software development", IEEE Softw. Vol. 25, No. 4, pp. 21-28, 2008.

[9] T. Baba, T. Ohkawa, S. Watanabe, K. Ootsu, B. J. Jackin, T. Yokota, Y. Hayasaki, & T. Yatagai, "Overcoming the difficulty of large-scale CGH generation on multi-GPU cluster", 11th Workshop on General Purpose Processing Using GPUs, GPGPU 2018 - Proceedings, pp. 13-21, Feb. 2018. DOI: https://doi.org/10.1145/3180270.3180273

[10] H. Kim, Y. Kim, H. Ji, H. Park, J. An, H. Song, Y. T. Kim, H. S. Lee & K. Kim, "A single-chip FPGA holographic video processor", IEEE Transactions on Industrial Electronics, Vol. 66, No. 3, pp. 2066-2073, 2019. (in Korean) DOI: https://doi.org/10.1109/TIE.2018.2835424

[11] V. R. Basili, D. Cruzes, J. C. Carver, L. M. Hochstein, J. K. Hollingsworth, and M. V. Zelkowitz, "Understanding the high-performance computing

community: A software engineer's perspective", IEEE Software, Vol. 25, No. 4, pp. 29–36, July/August 2008.

[12] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post, "Software development environments for scientific and engineering software: A series of case studies", In Proc. International Conference on Software Engineering, pp. 550–559, 2007.

[13] J. Segal, "When software engineers met research scientists. Empirical Software Engineering, Vol. 10, No. 4, pp. 517–536, 2005.

[14] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do scientists develop and use scientific software?", Proc. 2009 ICSE Work. Softw. Eng. Comput. Sci. Eng. SECSE 2009, pp. 1–8, 2009,
DOI: https://doi.org/10.1109/SECSE.2009.5069155.

[15] A. Taweel, B. Delaney, T. N. Arvanitis, and L. Zhao, "Communication, knowledge and co-ordination management in globally distributed software development: Informed by a scientific software engineering case study", Proc. - 2009 4th IEEE Int. Conf. Glob. Softw. Eng. ICGSE 2009, pp. 370–375, 2009, DOI: https://doi.org/10.1109/ICGSE.2009.58.

[16] B. C. Jang, and S. C. Kim, "Development of U-shaped Metal Bellows design software based on MATLAB", Journal of the Korea Academia-Industrial, Vol. 16, No. 4, pp. 2379-2384, 2015,
DOI: http://dx.doi.org/10.5762/KAIS.2015.16.4.2379.

[17] R. H. Fogh, W. Boucher, W.F. Vranken, A. Pajon, T. J. Stevens, T. N. Bhat, J. Westbrook, J. M. Ionides, and E.D.Laue, "A framework for scientific data modeling and automated software development," Bioinformatics, Vol. 21, No. 8, pp. 1678–1684, 2005, DOI: https://doi.org/10.1093/bioinformatics/bti234.

[18] C. A. Mattmann, D. J. Crichton, N. Medvidovic, and S. Hughes, "A software architecture-based framework for highly distributed and data intensive scientific applications," Proc. - Int. Conf. Softw. Eng., Vol. 2006, pp. 721–730, 2006,
DOI: https://doi.org/10.1145/1134285.1134400.

[19] Y. H. Wang and I. C. Wu, "Achieving high and consistent rendering performance of java AWT/Swing on multiple platforms," Softw. - Pract. Exp., Vol. 39, No. 7, pp. 701–736, 2009,
DOI: https://doi.org/10.1002/spe.

[20] M. Geimer, K. Hoste, and R. McLay, "Modern scientific software management using easybuild and lmod," Proc. HUST 2014 1st Int. Work. HPC User Support Tools - Held Conjunction with SC 2014 Int. Conf. High Perform. Comput. Networking, Storage Anal., pp. 41–51, 2014,
DOI: https://doi.org/10.1109/HUST.2014.8.

[21] A. A. Goloborodko, L. I. Levitsky, M. V. Ivanov, and M. V. Gorshkov, "Pyteomics - A python framework for exploratory data analysis and rapid software prototyping in proteomics," J. Am. Soc. Mass Spectrom., Vol. 24, No. 2, pp. 301–304, 2013, DOI: https://doi.org/10.1007/s13361-012-0516-6.

[22] R. F. Boisvert, S. E. Howe, and D. K. Kahaner, "GAMS: A Framework for the Management of Scientific Software," ACM Trans. Math. Softw., Vol. 11, No. 4, pp. 313–355, 1985
DOI: https://doi.org/10.1145/6187.6188.

[23] C. Morris, J. Segal, and W. Hall, "Some Challenges Facing Scientific Software Developers : the Case of Molecular Biology," 2009,
DOI: https://doi.org/10.1109/e-Science.2009.38.

[24] M. T. Sletholt, J. Hannay, H. C. Benestad, and H. P. Langtangen, "A Literature Review of Agile Practices and Their Effects in Scientific Software Development", pp. 1–9, 2011.

[25] M. T. Sletholt, J. E. Hannay, D. Pfahl, and H. P. Langtangen, "What Do We Know about Scientific Software Development's Agile Practices?", Comput. Sci. Eng., Vol. 14, No. 2, pp. 24–37, Mar. 2012,
DOI: https://doi.org/10.1109/MCSE.2011.113.

[26] S. K. Yoo and S. Y. Cho, "Development of Quality Evaluation Metrics for System Software", In Proceedings of The International Workshop on Future Technology, pp. 73-74, Cheonan, Korea, 2017. May

[27] P. Sfetsos, L. Angelis, and I. Stamelos, "Prioritized test-driven reverse engineering process: A case study", In 2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA) IEEE., pp. 1-6, Jul. 2015.

[28] J. Andrea, "Test-Driven Development: Driving New Standards of Beauty," Beautiful Testing: Leading Professionals Reveal How They Improve Software, pp. 181-194, O'Reilly Media, 2009

[29] K. Beck. Test-driven development: By example. Boston: AddisonWesley, 2003.

[30] K. Bajaj, H. Patel and J. Patel, " Evolutionary software development using test driven approach", In 2015 International Conference and Workshop on Computing and Communication (IEMCON), IEEE., pp. 1-6, Oct. 2015.

[31] D. Leffingwell and D. Widrig, " Managing Software Requirements - A Use Case Approach", 2nd ed. Addison-Wesley, Boston, 2003.

저 자 소 개

이 재 홍(학생회원)

- 2020년 2월 : 한국기술교육대학교 컴퓨터공학과(공학사)
- 2020년 3월~현재 : 한국기술교육대학교 컴퓨터공학과(석사과정)
- 주관심분야 : 병렬처리, 고성능컴퓨팅, 컴퓨터 홀로그래피

김 덕 수(정회원)

- 2008년 2월 : 성균관대학교 정보통신공학부(공학사)
- 2014년 8월 : KAIST 전산학과(공학박사)
- 2014년 7월 ~ 2018년 2월 : 한국과학기술정보연구원 선임연구원
- 2018년 3월 ~ 현재 : 한국기술교육대학교 조교수
- 주관심분야 : 고성능컴퓨팅, 그래픽스/가시화, 인공지능 등

김 승 희(정회원)

- 2003년 : 동국대학교 컴퓨터멀티미디어 공학과 (공학사)
- 2005년 8월 : 연세대학교 산업정보경영 (공학석사)
- 2014년 2월 : 서울과학기술대학교 산업정보시스템 (공학박사)
- 2016년 2월 ~ 현재 : 한국기술교육대학교 IT융합SW공학과 조교수
- 주관심분야 : SW 품질 공학, IT 서비스 최적화, 정보 시스템, IT프로젝트 관리, 블록체인 등