KIPS Trans. Softw. and Data Eng.
Vol.11, No.7 pp.273~282
ISSN: 2287-5905 (Print), ISSN: 2734-0503 (Online)

https://doi.org/10.3745/KTSDE.2022.11.7.273

Code Coverage Measurement in Configurable Software Product Line Testing

Soobin Han† · Jihyun Lee†† · Seoyeon Go†

ABSTRACT

Testing approaches for configurable software product lines differs significantly from a single software testing, as it requires consideration of common parts used by all member products of a product line and variable parts shared by some or a single product. Test coverage is a measure of the adequacy of testing performed. Test coverage measurements are important to evaluate the adequacy of testing at the software product line level, as there can be hundreds of member products produced from configurable software product lines. This paper proposes a method for measuring code coverage at the product line level in configurable software product lines. The proposed method tests the member products of a product line after hierarchizing member products based on the inclusion relationship of the selected features, and quantifies SPL(Software Product Line) test coverage by synthesizing the test coverage of each product. As a result of applying the proposed method to 11 configurable software product line cases, we confirmed that the proposed method could quantitatively visualize how thoroughly the SPL testing was performed to help verify the adequacy of the SPL testing. In addition, we could check whether the newly performed testing for a member product covers the newly added code parts of a feature.

Keywords : Configurable Software Product Line, Software Product Line Testing, Test Coverage Measurement, Code Coverage

구성가능한 소프트웨어 제품라인 시험에서 코드 커버리지 측정

한 수 빈† · 이 지 현†† · 고 서 연†

요 약

구성가능한 소프트웨어 제품라인 시험은 모든 멤버제품들에 포함되는 공통 부분과 일부 또는 단일 제품에 포함되는 가변 부분을 고려해야 하기 때문에 단일 제품 시험 방법들을 그대로 적용하기 어렵다. 소프트웨어 시스템 시험에서 시험 커버리지는 수행된 시험의 적절성을 측정하는 척도이다. 구성가능한 소프트웨어 제품라인으로부터 생성될 수 있는 멤버제품들은 수백 개에 이를 수 있기 때문에 시험 커버리지 측정은 소프트웨어 제품라인 수준에서 시험의 적절성을 평가하기 위해 중요하다. 이 논문은 구성가능한 소프트웨어 제품라인 시험에서 제품라인 수준의 코드 커버리지 측정 방법을 제안한다. 제안한 방법은 피처 집합의 포함 관계를 기준으로 제품들을 계층화한 후 제품라인의 멤버제품들을 시험하고, 시험으로부터 얻어진 각 제품의 시험 커버리지들을 종합하여 SPL 시험 커버리지를 정량화한다. 제안한 방법을 11개의 구성가능한 소프트웨어 제품라인 사례에 적용하여 검증한 결과, 제안한 방법은 SPL 시험이 얼마나 철저하게 수행되었는지를 정량적으로 기술하여 SPL 시험의 적정성을 확인할 수 있도록 도왔다. 또한, 새로 추가된 SPL 멤버제품의 시험이 이전 멤버제품 시험 대비 커버리지를 증가시키는 방향인지 확인할 수 있어 새로운 시험케이스들이 멤버제품들 간의 차이를 커버하는 방향으로 시험이 이루어지고 있는지 확인할 수 있었다.

키워드 : 구성가능한 소프트웨어 제품라인, 소프트웨어 제품라인 시험, 시험 커버리지 측정, 코드 커버리지

1. 서 론

기업들은 단일 제품보다는 유사한 제품들을 일련의 제품군으로 생산하고 관리함으로써 고객의 다양한 니즈를 충족시켜 시장에서 경쟁력을 확보하고자 한다. SPL(Software Product Line)은 기업이 유사한 제품들을 제품군(product family)으로 계획하여 개발하고 이 제품군을 유지하고 지속적으로 진화시키는데 필요한 체계적인 절차와 기술들을 지원하는 개발 방법론이다[1,2]. SPL 방법론을 적용하여 개발한 제품군에 대한 시험은 제품군을 구성하는 멤버제품 모두에 포함되는 공통 부분과 일부 제품 또는 단일 제품에만 포함되는 가변 부분을 고려해야 하고 제품군과 멤버제품을 개발하는 두 개의 개발 라이프사이클이 존재하기 때문에 단일 제품 시험 방법들을 그대로 적용하기 어렵다[3]. 시험 커버리지 기준과 측정 방법 또한 마찬가지이다. 현재 SPL 시험 커버리지는 주로 멤버제품별로 측정하는 것으로 간주되고

※ 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.NRF-2020R1F1A1071650).
※ 이 논문은 2021년 한국정보처리학회 ACK 2021의 우수논문으로 "구성가능한 소프트웨어 시스템의 시험 커버리지 측정 연구"의 제목으로 발표된 논문을 확장한 것임.
† 비 회 원 : 전북대학교 소프트웨어공학과 학사과정
†† 정 회 원 : 전북대학교 소프트웨어공학과 교수
Manuscript Received : December 28, 2021
First Revision : February 4, 2022
Second Revision : February 25, 2022
Accepted : February 28, 2022
* Corresponding Author : Jihyun Lee(jihyun30@jbnu.ac.kr)

있다. 많은 연구들이 실제 SPL 시험 커버리지를 직접 측정하기 보다는 시험 대상인 샘플링한 멤버제품들이 얼마나 많은 피처, 피처 조합, 또는 코드를 커버하고 있는지로 측정하고 있다[4]. 그러나, 이러한 방식의 SPL 시험 커버리지 측정은 재사용과 중복된 개발 관련 활동의 최소화를 지향하는 SPL의 원칙에 부합하지 않는다. SPL 시험 또한 SPL 기본 원칙에 맞게 중복된 시험을 최소화할 수 있어야 하고[5], 시험 커버리지 역시 이를 고려하여 측정되어야 한다.

따라서 개별 제품이 아닌 SPL 수준에서 시험 커버리지 기준을 측정하고 시험의 적절성을 평가하기 위한 방법이 필요하다. 이 논문에서는 SPL 개발 방법이 잘 동작하는 분야인 구성가능한 소프트웨어 제품라인(configurable software product line)에서 시험을 수행하고 시험 커버리지를 측정하는 방법을 제안하고 실험을 수행한 결과를 기술한다.

본 논문의 기여는 다음과 같다: (1) 시험 커버리지 기준의 정의에 맞게 SPL 수준에서 시험케이스가 실행한 시험을 기준으로 시험 커버리지를 정량적으로 측정한다; (2) SPL 멤버제품의 시험케이스가 잘 설계되었는지를 확인하도록 해준다. SPL 시험이 얼마나 철저하게 수행되었는지를 정량적으로 기술하고, 새로 추가된 SPL 멤버제품의 시험이 이전 멤버제품 시험 대비 커버리지를 증가시키는 방향인지 확인할 수 있어 새로운 시험케이스들이 멤버제품들 간의 차이를 커버하는 방향으로 시험이 이루어지고 있는지 확인할 수 있다.

논문의 구성은 다음과 같다: 먼저 2장에서는 기존 연구들이 채택하고 있는 시험 커버리지 기준들과 측정 방법을 설명한다. 3장에서는 본 연구가 제안하는 커버리지 측정 방법을 기술하고, 4장에서는 제안한 방법을 적용한 실험 결과를 5장에서는 실험 결과에 대한 토의를 기술한다. 마지막으로 6장에서는 결론을 내리고 향후 연구 방향을 제시한다.

2. 관련 연구

대부분의 SPL 시험 연구들은 SPL 시험의 적절성을 평가하기 위하여 피처 커버리지 기준 또는 피처 인터액션(또는 피처 조합) 커버리지 기준을 주로 사용하고 있다[4].

피처 커버리지 기준에서 시험의 적절성을 평가하는 커버리지의 기준은 SPL이 제공하는 피처이다. 피처는 “최종 사용자에게 보이는 시스템의 특징”으로 요구사항보다 추상화 수준이 높은 개념이다[7]. 이 커버리지 기준에서 시험 요구사항은 “피처”이고, SPL 피처가 시험케이스를 설계하는 기준이다. 이 커버리지 기준은 “피처가 정의하고 있는 시스템 특성이 올바르게 실행되는지 검증하는 것”으로 정의될 수 있다. 그러나, 피처 커버리지를 사용하는 연구들은 시험 커버리지를 “시험을 위해 생성한 멤버제품들에 포함된 피처의 비율”로 측정한다[8-17]. 시험할 멤버제품의 수가 많을수록 시험 대상이 많아져 시험에 많은 노력이 투입되기 때문에 시험 대상이 되는 멤버제품의 수를 줄이는 방향으로 연구들이 이루어져 왔기 때문이다. 하지만 시험 커버리지는 “피처가 실제 시험케이스에 의해 시험되었는가”로 측정되어야 한다.

피처 인터액션 커버리지 기준에서 시험의 적절성을 평가하는 커버리지의 기준은 “t개의 피처 조합이 올바르게 인터페이싱하여 주어진 요구사항대로 올바르게 수행되었는지 검증하는 것”으로 정의될 수 있다. SPL 시험에서 t-way 또는 t-wise 피처 인터액션 시험의 경우, SPL 시험은 t개의 피처 조합을 적어도 한 번은 시험해야 한다. 이 커버리지 기준에서 시험 요구사항은 “피처 인터액션”이고 t-wise 피처 조합을 실행하는 시험케이스를 설계해야 이 기준을 충족시킬 수 있다. 예를 들어, SPL 시험에서 페어와이즈 조합 시험 커버리지 기준을 충족하려면 시험은 각 피처 쌍을 시험하기 위한 시험케이스를 포함해야 한다. 그러나, 기존 연구들은 SPL로부터 생성된 멤버제품들이 t개의 피처 조합을 모두 포함하고 있는지 여부를 판단하기 위해 이 기준을 사용하고 있다[18-20]. 이는 기존 연구들이 피처 커버리지의 경우와 마찬가지의 시각으로 SPL 시험과 SPL 시험 커버리지를 바라보기 때문이다.

기존 SPL 시험 연구들은 가능한 피처 또는 피처 조합을 커버할 수 있는 SPL 멤버제품들을 생성하였는지 여부를 판단하기 위해 이 두 커버리지 기준을 적용하고 있다. 시험 커버리지 기준을 “시험케이스 설계에 시험 요구사항을 적용하는 규칙들[21]”로 정의하고 있음을 고려해 보면 기존의 연구들이 커버리지 기준을 잘못 적용하고 있음을 알 수 있다.

이들 두 시험 커버리지 기준 이외에도 시험 커버리지 기준으로 라인 커버리지 기준을 사용할 수 있다. 일부 SPL 시험 연구들은 라인 커버리지 기준을 채택하고 있다[22, 23]. [23]의 경우, SPL 시험이 실행한 라인 수를 측정하는 방법을 제시한 연구는 아니다. [22]는 SPL로부터 멤버제품을 생성하고 각 제품의 라인의 수를 측정하고는 있으나 전체 SPL 코드베이스 대비 시험이 실행한 라인의 수가 아니라 멤버제품의 라인 수를 측정하고 있다. 이들 연구 역시 앞선 커버리지 기준의 한계가 드러났던 연구들과 마찬가지로 시험 실행의 적절성을 평가하기 위한 커버리지 기준이라기보다는 시험 대상이 될 멤버제품의 수의 적절성을 평가하기 위한 커버리지 기준이 사용되었다고 볼 수 있다. 이 커버리지 기준에서 SPL 시험의 적절성을 평가하는 기준은 “시험이 커버한 SPL 코드베이스의 라인의 수”여야 한다.

기존 SPL 시험 연구들은 본래 시험 커버리지 기준에 부합하지 않은 정의를 사용하고 있으며 그에 따라 커버리지 측정 또한 잘못된 방식으로 수행되고 있다. 실제 SPL 수준에서 시험 커버리지를 측정하고 있지 않았다. 따라서 SPL의 특성을 반영하면서 본래 시험 커버리지 기준에 부합하는 시험 커버리지 기준과 커버리지 측정 방법에 대한 연구가 필요하다.

3. SPL 커버리지 측정 방법

3.1 정의와 가정

본 논문이 제안하는 방법은 다음 정의를 기반으로 하고 있다.

a) 정의1(생성적 제품라인): 생성적 제품라인이란 제품군의 모든 제품 산출물이 플랫폼 산출물들의 인스턴스화에 의하여 만들어질 수 있는 제품라인을 말한다[7].

b) 정의2(SPL 코드베이스): SPL 코드베이스란 제품라인의 멤버제품들을 도출하는데 사용될 전체 코드 조각들의 집합[5] 또는 전체 코드를 의미한다. SPL 코드베이스의 형태는 제품라인 가변성 구현 메커니즘에 따라 다를 수 있다.

c) 정의3(측정 요소, E_c): 측정 요소란 SPL 시험 커버리지를 측정하기 위해 사용할 기본 요소(Coverage measurement elements)로 시험 베이시스에 따라서 달라진다. 예를 들어 시험 베이시스가 소스 코드인 경우, 측정 요소는 라인(line), 분기(branch), 명령(instruction)이 될 수 있다.

본 논문이 제안하는 방법은 다음 가정이 충족되는 조건에서 동작한다.

a) 가정1(결정적 제품라인 시험 실행): 동일 제품에 동일 시험케이스를 실행하면 시험 실행 결과는 결정적, 즉 항상 동일하다.
예를 들어, 시험케이스가 랜덤 함수를 시험한다면 시험을 실행할 때마다 다른 결과를 낼 수 있으며 이는 비결정적인 시험 실행이다.

b) 가정2(어노테이션 기반 제품라인 개발 방법): 파라미터화 방법(parametrization approach)으로도 불리고, 모든 피처들의 코드는 단일 코드베이스에서 병합되며 어노테이션은 어떤 코드가 어떤 피처에 해당하는지 표시를 한다[24].

어노테이션 기반 제품라인 개발 방법은 C 전처리기를 사용한 가변 코드 구현이 대표적인 예이다.

3.2 SPL 시험 커버리지 측정 방법

단일 제품 시험의 시험 커버리지 측정 도구인 JaCoCo[25]는 명령, 라인, 분기, 메소드(method) 단위로 커버리지를 추적한다. JaCoCo를 활용하여 SPL 코드베이스로부터 도출된 멤버제품들의 시험 커버리지를 측정한 후 가장 커버리지가 높은 값을 SPL 코드베이스의 시험 커버리지로 간주할 수 있다. 그러나 이 결과는 커버리지 비율 정보만 제공할 뿐이어서 시험이 어떤 피처의 구현 코드를 어느 정도 커버했는지 파악할 수 없다. 이는 SPL의 멤버제품 시험은 단일 제품 시험과 동일하기 때문에 멤버제품이 새로운 피처를 포함하더라도 각 시험에서 새로운 피처와 관련하여 커버한 부분을 추적하기 어렵기 때문이다. 그 결과로 시험 커버리지를 알더라도 커버리지를 높이기 위해 새로운 시험케이스를 설계하는데 활용할 수 없다.

따라서 본 연구에서는, 피처의 구현 코드에 대한 커버리지 정보를 제공하는 SPL 시험 커버리지를 측정하기 위해 멤버제품에 실행된 시험의 최소 단위인 원자단위 시험(atomic test)들의 커버리지를 추적한다. JUnit 프레임워크의 경우 원자단위 시험은 시험 메소드(test method)이다. SPL 시험 커버리지 측정은 정해진 수의 멤버제품들을 생성한 후 각 제품을 시험한 후 수행된다. 제안한 방법은 실행된 시험 메소드와 관련된 피처를 추적하여 피처와 관련된 코드 정보와 커버리지 정보를 파악한다. 이 절에서는 Fig. 1이 보여주고 있는 본 논문이 제안하는 SPL 시험 커버리지 측정 방법을 구성하는 각 단계의 세부 활동을 설명한다.

1) 단계1: 제품 계층화(Product hierarchization)

SPL 개발 라이프사이클을 통하여 구축한 SPL 코드베이스의 품질을 보증하는 방법 중 하나는 시험 효과가 높은 멤버제

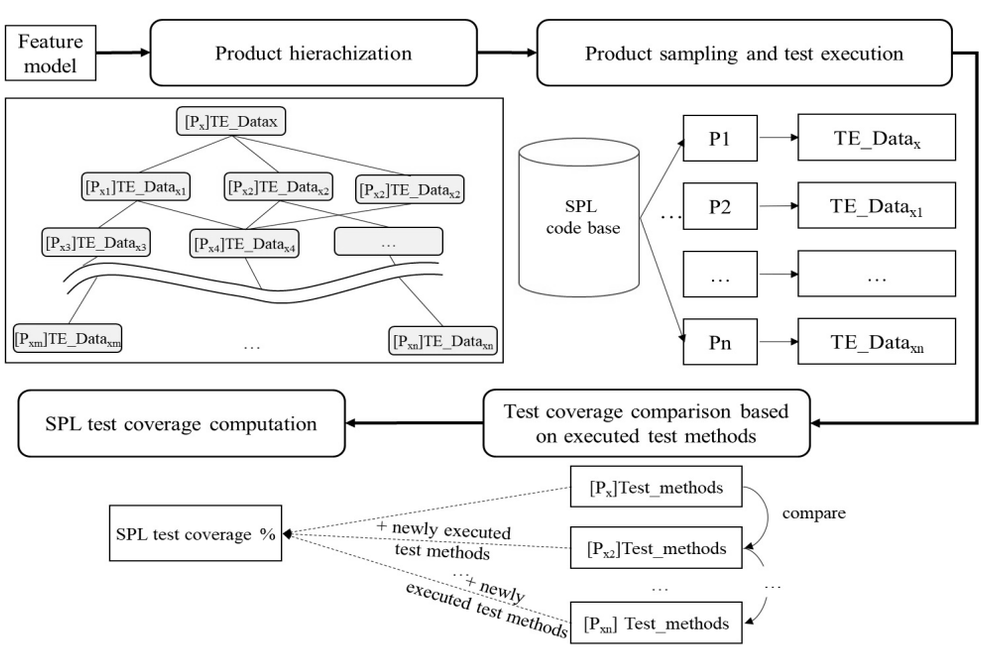

Fig. 1. Overview of Code Coverage Measurement for Configurable Software Systems

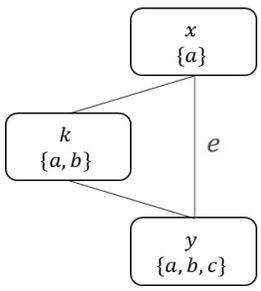

Fig. 2. Relations among Products in Product Hierarchy Graph

품을 선정하고 시험하는 것이다. 또한, 멤버제품을 구성할 때 상호배타적인 피처들로 구성하는 것이 중요하다. 이를 위해 이 단계에서는 Fig. 1의 첫 번째 활동과 같이 피처 모델을 입력으로 하여 각 제품을 구성하는 피처들의 수와 피처 집합 간의 포함 관계를 고려하여 제품 계층 그래프(product hierarchy graph)를 구성한다.

제품 계층화는 주어진 제품들을 그래프 형태로 연결하여 그 계층을 표현한다. 정점은 제품이며, 간선으로 연결된다. 가장 위(부모)의 제품은 피처 수가 가장 적은 제품이며, 가장 아래(자식)의 제품은 피처 수가 가장 많은 제품이 된다. 자식 제품은 반드시 부모 제품의 모든 피처를 포함해야 한다. 같은 부모를 지녔으면서 서로가 부모와 자식 관계인 제품은 존재할 수 없다.

G는 제품 계층 그래프로 $G=(P,E)$이다. P는 제품들을, E는 제품들의 계층을 표현하는 간선들이다. 임의의 제품 x,y,k에 대하여 각 제품의 피처 집합이 F_x,F_y,F_k이라고 가정하자. 이때, 정점 P를 연결하는 간선 $E=\{(x,y)|(F_x \subset F_y)\wedge(\nexists F_k : F_x \subset F_k \wedge F_k \subset F_y)\}$이다. 가령, Fig. 2와 같은 계층 그래프에서 x와 y를 잇는 간선 e는 존재할 수 없다. $F_x \subset F_k \wedge F_k \subset F_y$를 만족하는 제품가 존재하기 때문이다.

따라서 가장 높은 유사도를 보이면서 피처를 완전히 포함하는 제품들끼리 부모와 자식 관계가 성립한다. Table 1의 예와 같이 a와 b, a와 c, b와 d, c와 d는 서로 부모-자식 관계이며, b와 c는 형제 노드가 된다.

Table 2는 Table 1의 제품들 간의 유사도를 자카드 유사도(Jaccard similarity) 메트릭을 사용하여 계산한 결과이다. 인접한 노드의 제품일수록 유사도 값이 높고 거리가 멀수록 유사도 값이 낮음을 확인할 수 있다.

Fig. 3은 Table 1의 샘플링한 멤버제품들의 계층 그래프이다.

2) 단계2: 제품 선정 및 시험 실행(Product sampling and test execution)

멤버제품들의 시험을 통해 달성한 SPL 시험 커버리지를 측정하려면 각 멤버제품에서 새로 추가된 피처들과 해당 코드들에 대한 시험 커버리지가 추적되어야 한다. 이 단계에서는 멤버제품들에서의 시험 결과 측정된 시험 커버리지에 따라 SPL 코드베이스의 시험 커버리지를 측정하기 위해서는 Fig. 1과 같이 SPL 개발의 플랫폼 개발 프로세스를[6] 통해

Table 1. Example Feature Combination

Product	Feature set
a	Base, Color
b	Base, Color, Extendedsudoku
c	Base, Color, States
d	Base, Color, Extendedsudkou, State

Table 2. Jaccard Similarity for Products in Table 1

	a	b	c	d
a	1	**0.667**	**0.667**	0.5
b	-	1	0.5	**0.75**
c	-	-	1	**0.75**
d	-	-	-	1

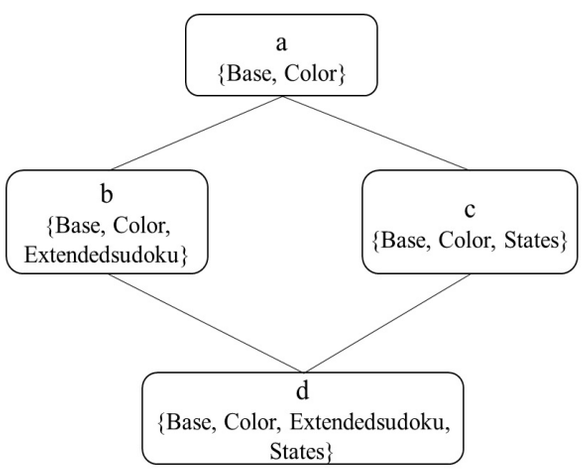

Fig. 3. Product Hierarchy Graph Expression of Member Products in Table 1

개발된 SPL 코드베이스로부터 제품들을 생성(instantiation)하고 생성한 제품과 관련된 시험케이스들을 실행하여 제품을 시험한다. 플랫폼 시험 설계 단계에서 이미 시험케이스를 작성하였다면 SPL 코드베이스에는 시험케이스들이 포함되어 있다. 그렇지 않은 경우나 필요한 경우에는 시험 실행 단계에서 시험케이스를 새로 설계하고 추가할 수 있다.

3) 단계3: 시험 커버리지 비교(Test coverage comparison)

단계 2에서 선정하여 시험을 실행한 멤버제품들에 대해서 제품 계층 그래프의 상위 노드에 위치한 제품을 시작으로 시험 실행 데이터로부터 원자단위 시험들이 실행한 코드 라인을 추적한다. 부모와 자식 제품들에서 실행된 원자단위 시험들의 시험 커버리지 데이터를 비교하여 일련의 제품 시험에서 시험이 새로이 실행한 정의3의 측정 요소를 포함하도록 시험 커버리지 값을 갱신한다. 시험 측정 요소 E_c는 코드 커버리지를 사용하는 경우, 라인, 분기 또는 명령어가 될 수 있다. 각 멤버제품 P_i가 m개의 피처 F를 포함하고 $F_j(E_c)$가 피

처 F_i의 시험 커버리지라고 할 때, P_i의 시험 커버리지 T_c는 Equation (1)과 같이 계산될 수 있다:

$$T_c(P_i) = \sum_{j=1}^{m} F_j(E_c) \quad (1)$$

4) 시험 커버리지 계산(SPL test coverage computation)

이 단계에서는 단계3의 결과를 토대로 총 SPL 코드베이스의 코드 라인 수 대비 SPL 시험에서 실행한 전체 라인 수로 최종 SPL 시험의 코드 커버리지 값을 계산한다. n개의 제품으로 구성된 SPL의 시험 커버리지 $T_c(SPL)$은 Equation (2)와 같이 계산될 수 있다:

$$T_c(SPL) = \bigcup_{i=1}^{n} T_c(P_i) \quad (2)$$

4. 실험 및 결과

논문에서 제안한 SPL 시험 커버리지 측정 방법을 검증하기 위하여 제안한 방법을 이클립스 플러그인으로 구현하였다. 시험 커버리지 측정 도구로는 단일 소프트웨어 시험에서 널리 사용되는 JaCoCo를 사용하였고 시험케이스 구현 및 실행은 JUnit 프레임워크를 활용하였다. 이 장에서는 제안한 방법을 검증하기 위하여 수행한 실험 설계 및 결과에 대해 설명한다.

4.1 실험 설계

실험에 사용한 제품군 사례들은 어노테이션 기반 방법(annotation-based approach)으로 개발된 사례들이다[6].[1] 구성가능한 소프트웨어 제품라인에서는 멤버제품의 피처 구성을 선택하면 구체적인 개별 소프트웨어 시스템과 관련 산출물들이 함께 생성되는데, 실험에 사용한 제품군 사례들의 산출물은 생성된 시스템의 실행가능한 코드와 관련된 시험 스위트들을 포함한다. 실험은 [6]이 제공하는 제품군의 사례들 중 11개를 선정하여 수행되었다. 실험 대상으로 선정한 제품군의 목록과 세부 정보는 Table 3과 같다.

제품군 사례는 실험의 신뢰성을 높이기 위해 [6]이 제공하는 제품군의 사례들 중에서 그 규모와 복잡도가 다양한 구성이 되도록 선정하였다. 선정한 제품군의 전체 코드베이스의 코드 라인 수는 189줄부터 2376줄까지 고루 분포한다.

본 실험에서는 직접 피처 모델로부터 피처를 선택하여 시험 대상 제품을 구성하지 않고, 시험 실행부터 시험 커버리지 측정까지의 전 과정을 자동화하기 위해 개발 중인 도구 상에서 서로 다른 유효한 제품을 최대 250개 무작위로 생성하도록 하여 실험하였다. 최대 250개의 제품을 선정한 이유는 유효한 모든 제품을 실험하는 것은 현실적으로 어렵기 때문이다. 예를 들어 Table 3의 FeatureAMP8의 경우 유효한 제품의 개수는 계산해 본 결과 15,708개였으며 확인한 결과 하나의 제품을 시험하는데 소요되는 시간은 최소 5분이었다. 따라서 24시간 이내에 작업을 완료할 수 있는 적절한 제품의 수를 선정하여 실험을 진행하였다.

또한, 본 실험의 주목적이 멤버제품에 대한 시험 결과를 이용하여 SPL 시험 커버리지를 정확하게 측정할 수 있는지 여부를 확인하는 것이기 때문에, 시험할 제품을 얼마나 효율적으로 선정하느냐는 중요한 문제가 아니다. 실험은 16GB 램, 3.4GHz 12CPU Ryzen 프로세서에서 수행되었다.

Table 3. Size Metrics of PL Subjects

Configurable PL Subject	Size metrics			
	#LOC	# of packages	# of classes	# of methods
ATM	1,160	2	27	100
BankAccount	189	3	9	22
Chess	2,149	7	22	162
Elevator	405	3	9	60
FeatureAMP1	1,350	4	15	93
FeatureAMP8	2,376	2	6	106
GPL	1,235	3	17	78
Minepump	244	2	7	26
Notepad	1,564	4	17	90
Sudoku	949	2	13	51
VendingMachine	472	2	7	21

4.2 실험 결과

Fig. 4는 실험 사례 중 Elevator 제품군의 피처 모델이다.

Table 4는 Elevator 제품라인으로부터 샘플링된 세 가지 제품의 피처 조합을 보여주고 있다.

Fig. 5는 Elevator의 제품 계층 그래프이다. 제품a, 제품b, 제품c는 Fig. 5의 상위 노드에 위치한 제품들이다.

Fig. 6은 구현한 도구가 a, b, c의 세 제품들에 대해 연속으로 시험을 수행한 후 Equation (1)과 Equation (2)를 적용하여 SPL 수준의 시험 커버리지를 시각화한 결과이다. 측정 요소는 라인을 사용하였다. 적색은 “커버되지 않았음”, 황색은 “일부분(분기 중 하나)이 커버되었음”, 녹색은 “커버되었음”을 의미한다. 제품a만 시험했을 때보다 피처가 더 많은 제품b를 시험했을 때 적색이 점차적으로 감소함을 확인할 수 있다. 즉, 피처가 많은 제품을 시험할수록, 커버리지 데이터 역시 증가함을 확인할 수 있다. 이는 곧 추가된 피처에 대해 시험이 적절히 수행되고 있다고 판단할 수 있는 근거가 된다. 결론적으로 Elevator PL에 대한 시험은 자식 제품에 대한 시험이 추가될수록 부모 제품의 시험과 비교했을 때 시험 커버

1) 실험 대상 제품군들은 https://fischerjf.github.io/challenge/에서 다운로드받을 수 있다. 이 웹사이트는 선택한 제품군 이외에도 다양한 제품군 사례들을 SPL 코드베이스와 함께 제공하고 있다.

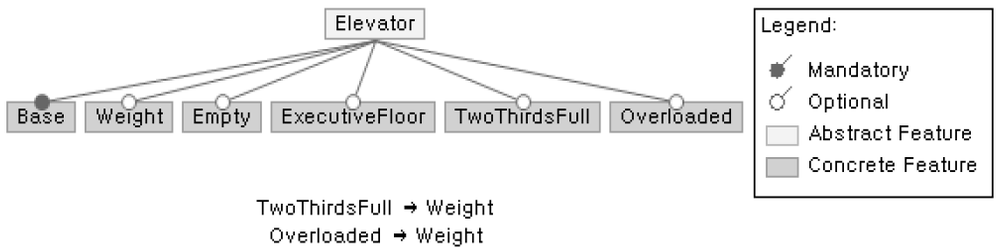

Fig. 4. Feature Model for the Elevator PL

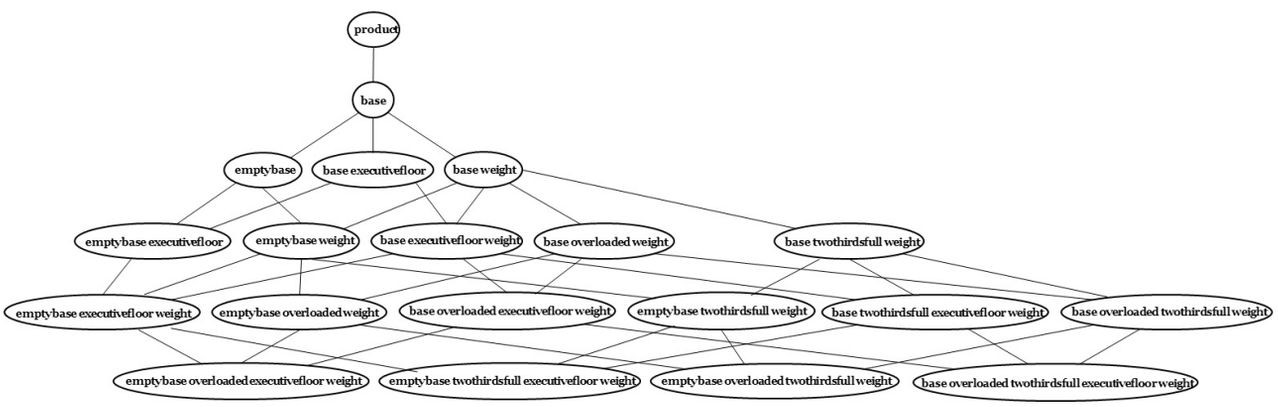

Fig. 5. Product Hierarchy Graph of the Elevator PL

Table 4. Sample Products with Selected Features of the Elevator PL

Sample product	Features selected
a	Base
b	Base, Weight
c	Base, Weight, Empty

```
public boolean leaveElevator(Person p) {
    if (persons.contains(p)) {
        persons.remove(p);
        p.leaveElevator();

        if (Configuration.weight) {
            weight -= p.getWeight();
        }
        if (Configuration.empty) {
            if (this.persons.isEmpty())
                Arrays.fill(this.floorButtons, false);
        }
        return true;
    }
    return false;
}
```

a b c

Fig. 6. Feature Coverage Difference among Sampled Products of Elevator PL

리지가 증가하고 있으며, 이를 통해 시험이 적절히 이루어지고 있다고 판단할 수 있었다.

Fig. 7은 실험 예시 중 VendingMachine PL의 피처 모델이다.

Table 5는 VendingMachine PL로부터 샘플링된 세 가지 제품의 피처 조합을 보여주고 있다.

Fig. 8은 샘플링된 제품의 제품 계층 그래프이다. Fig. 8에서 빨강색 원으로 표시된 제품은 부모 제품에 비해 시험 커버리지가 증가하지 않은 제품이다.

Fig. 9는 a, b, c의 세 제품들에 대해 연속으로 시험을 수행한 후 역시 Equation (1)과 Equation (2)를 적용하여 현재까지 달성한 SPL 수준의 시험 커버리지를 시각화한 결과이다. 제품b는 제품a에 비해 시험 커버리지가 변화되었지만, 제품c는 제품b에 비해 시험 커버리지가 변화하지 않았다. 이는 곧 제품c의 일부 피처에 대한 시험이 제대로 이루어지지 않았음을 의미한다. 제품c는 제품b에 “Terminal” 피처가 새롭게 추가된 제품이므로, “Terminal” 피처에 대한 시험이 제대로 이루어지지 않았음을 예측할 수 있다. VendingMachine PL은 127개의 제품 중 103개의 제품이 부모 제품에 비해 시험 커버리지가 증가하지 않았으며 “Terminal”, “Keyboard”, “Availability” 피처에 대한 시험이 적정하게 수행되지 않은 것으로 확인되었다. 이는 해당 피처와 관련된 시험이 재설계되어야 함을 의미한다.

Table 6은 Table 3의 사례들의 SPL 코드베이스로부터 제품들을 생성하여 제안한 방법으로 시험한 후 SPL 시험 커버리지를 추적한 결과이다. 먼저, Table 6의 SPL 커버리지 열을 통하여 우리는 Notepad PL을 제외한 나머지 SPL 사례들의 코드베이스가 최소 70%의 라인 커버리지를 달성하도록

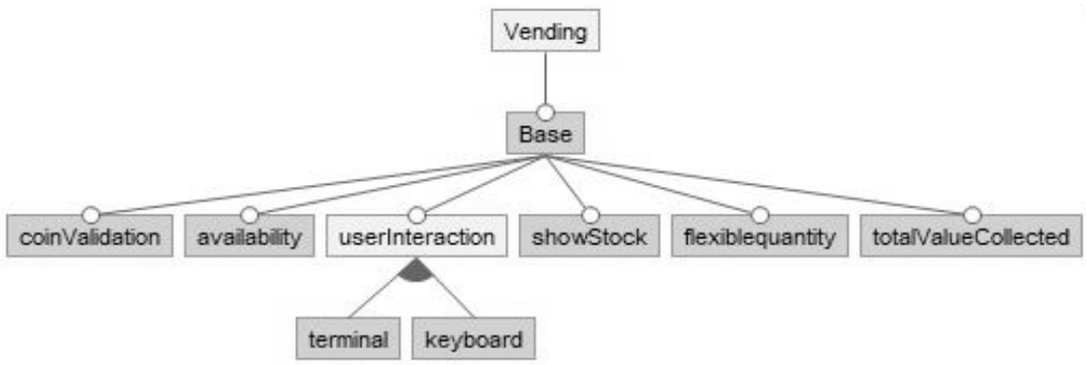

Fig. 7. Feature Model for the VendingMachine PL

Table 5. Sample Products with Selected Features of VendingMachine PL

Sample product	Features selected
a	Base, TotalValueCollected
b	Base, TotalValueCollected, CoinValidation
c	Base, TotalValueCollected, CoinValidation, Terminal

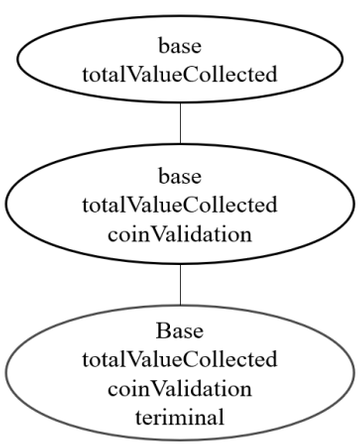

Fig. 8. Part of VendingMachine PL's Product Hierarchy Graph

시험케이스를 정의하고 있음을 알 수 있다. 사실 [6]은 모든 제품군의 사례들이 70% 이상의 커버리지를 달성할 수 있도록 시험케이스를 정의하였다고 기술하고 있다. 두 번째로 Table 6의 KM 열을 보면, Elevator PL과 Notepad PL을 제외한 모든 시스템에서 시험은 투입된 결함의 40% 이상을 검출해 내었음을 알 수 있다. 세 번째로 #1)과 #2)열을 통하여 우리는 Bankaccount PL, FeatureAMP1 PL, Minepump PL, VendingMachine PL에 대한 시험에 온전히 실행하지 못한 피처들이 있음을 파악할 수 있다. 특히, FeatureAMP1 PL은 시험 설계에 대한 재검토가 필요함을 알 수 있다.

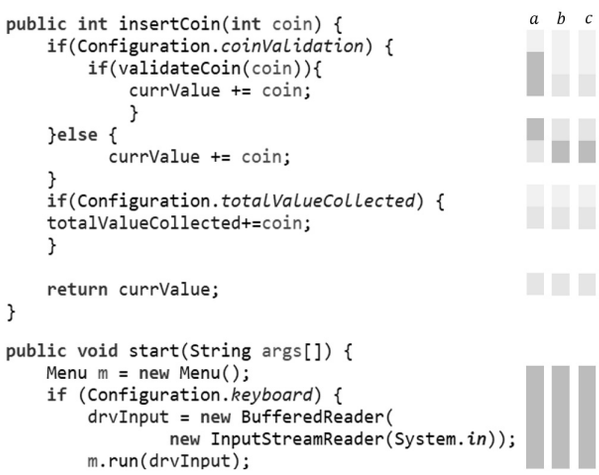

Fig. 9. Feature Coverage Difference among Sampled Products of VendingMachine PL

5. 토 의

본 연구의 의의는 다음과 같이 요약될 수 있다.

a) 제안한 SPL 시험 커버리지 측정 방법은 피처 커버리지 [8-14] 및 피처 인터액션 커버리지[18-20] 연구들과 달리 SPL 시험에 의해 실행된 실제 측정 요소를 기준으로 SPL 시험 커버리지를 측정하였다는 점이다. 기존 연구들은 SPL 시험 커버리지를 언급하긴 하였지만 실제로 측정하였다고 보기 어렵다.

b) 라인 커버리지[9, 10]를 사용한 연구들은 멤버제품별로만 시험이 실행한 라인을 추적했지만, 제안한 방법은 멤버제품별 시험 커버리지 측정 결과에 SPL 플랫폼 수준의 시험 커버리지인 SPL 시험 커버리지도 측정하였다는 점에서 기존 연구들과 차별화된다.

c) 세 번째 결과에서 확인할 수 있었던 것처럼 제안한 방법의 시험 커버리지 추적은 시험이 커버하지 못한 피처들을 정확하게 추적하였다. 시험 커버리지 추적에서 단일

Table 6. Results of Experiments

Name	Variability metrics			Test Suite metrics				
	Features	Total valid products	Generated products	Test cases executed	SPL test coverage	KM	#1)	#2)
Atm	7	80	79	76	91%	91%	0	0%
Bankaccount	10	144	143	42	92%	92%	2	57%
Chess	3	8	6	77	72%	72%	0	0%
Elevator	6	19	18	59	92%	10%	0	0%
FeatureAMP1	28	6732	250	18	85%	46%	11	56%
FeatureAMP8	27	15708	250	78	82%	42%	0	0%
GPL	13	73	58	51	83%	60%	0	0%
Minepump	7	64	64	34	91%	65%	3	83%
Notepad	17	256	250	25	59%	15%	0	0%
Sudoku	6	20	20	35	80%	67%	0	0%
VendingMachine	8	256	127	37	97%	83%	3	81%

KM: percentage of Killed Mutants
#1) number of features identified as not fully covered
#2) percentage of products not fully covered compared to the parent product

제품 시험에서와 같이 명령, 라인, 분기에 대한 시험 실행 여부만을 추적하면 한 피처(f1)가 다른 피처 (f2)를 포함하는 제약조건인 'requires'에서 문제가 발생할 수 있다. f1은 f2를 'requires'하기 때문에 f2의 시험 커버리지는 f1의 시험 커버리지를 포함할 가능성이 있다. f1을 포함하는 제품의 경우, 더 많은 시험이 더 정교하게 수행됐음에도 불구하고 시험 커버리지가 증가하지 않은 것으로 판단될 수 있다. 그러나 제안한 시험 커버리지 추적 방법은 같은 코드 부분일지라도 서로 다른 시험에 의해서 얼마나 정밀하게 커버되는지 파악할 수 있도록 관련 정보를 제공하기 때문에 시험 커버리지의 증감을 더 정확하게 판단할 수 있다.

d) 본 연구가 제안한 방법을 통하여 시험 커버리지를 측정하면 시험 커버리지에 변화를 주지 않는 피처들을 추출할 수 있었다. 기존 연구들이 사용한 SPL 시험 커버리지에 대한 시각과 달리, 제안한 방법은 특정 피처가 존재하건 존재하지 않건 시험 커버리지가 동일하다면 이는 해당 피처들에 대한 시험이 제대로 이루어지지 않는다는 것을 의미한다. 제안한 방법은 시험이 커버하지 못한 피처들에 대한 정보를 제공하기 때문에 시험 엔지니어가 가능한 모든 피처의 측정 요소에 대해 시험하는 방향으로 시험케이스를 설계할 수 있도록 도움을 준다.

6. 결론 및 향후연구

본 논문에서 우리는 피처 집합의 포함 관계를 기준으로 제품들을 계층화하고 SPL 코드베이스로부터 제품들을 생성하고 시험하여 얻은 각 제품의 시험 커버리지들을 종합하여 SPL 시험 커버리지를 측정하는 방법을 제안하고 검증하였다. 그 결과 우리는 SPL 시험 커버리지를 정량적으로 측정할 수 있었으며, 특정 피처에 대한 시험케이스의 충분성 여부, 즉 피처가 증가하였으나 시험 커버리지가 증가하지 않은 경우를 식별할 수 있었다. 제안한 방법은 기존의 피처 커버리지 기준, 피처 인터액션 커버리지 기준, 라인 커버리지 기준 등 기존 SPL 시험 커버리지 측정 방법이 반영하지 못한 SPL의 특성을 반영하면서 단일 소프트웨어 시스템 시험에서의 시험 커버리지의 역할에 부합하는 SPL 시험 커버리지 측정 방법이라는 점에서 의의가 있다.

그렇지만 본 논문이 제안한 SPL 시험 커버리지 측정 방법은 어노테이션 기반 개발 방법으로 구축한 구성가능한 소프트웨어 제품라인에 대해서만 그 효과가 확인되었다. 실험과 프로토타이핑한 도구 또한 구성가능한 소프트웨어 제품라인에 국한되어 있다. 향후에는 조합 기반 제품라인 개발 방법(composition-based approach)에서도 적용할 수 있도록 제안한 방법을 확장하고 그 효과를 검증하는 연구를 수행할 계획이다.

References

[1] K. Pohl, G. Böckle, and F. van der Linden, "Software product line engineering: Foundations, principles, and techniques," Springer, 2005.

[2] P. M. Clements and L. M. Northrop, "Software product lines: Practices and patterns," Addison Wesley, 2002.

[3] J. Lee, S. Kang, and D. Lee, "A survey on software product line testing," in *Proceedings of the 16th International Software Product Line Conference (SPLC'12)*, Salvador, Brazil, pp.2-7, Sep. 2012.

[4] J. Lee, S. Kang, and P. Jung, "Test coverage criteria for software product line testing: Systematic literature review," *Information and Software Technology*, Vol.122, pp.1-22, 2020.

[5] P. Jung, S. Kang, and J. Lee, "Efficient regression testing of software product lines by reducing redundant test executions," *Applied Sciences*, Vol.10, No.23, pp.1-21, 2020.

[6] F. Ferreira, M. Viggiato, M. Souza, and E. Figueiredo, "Testing configurable software systems: The failure observation challenge," in *Proceedings of the ACM Software and Systems Product Line Conference (SPLC'20)*, pp.1-6, Oct. 2020.

[7] S. Kang, J. Lee, Y. Han, and P. Jung, "Introduction to software product line development," HongReung Publishing, Company, 2021.

[8] M. B. Cohen, M. B. Dwyer, and J. Shi, "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach," *IEEE Transactions On Software Engineering*, Vol.34, No.5, pp.633-650, 2008.

[9] S. Oster, F. Markert, and P. Ritter, "Automated incremental pairwise testing of software product lines," in *Proceedings of the ACM Software and Systems Product Line Conference (SPLC'10)*, pp.196-210, Sep. 2010.

[10] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. Traon, "Automated and scalable t-wise test case generation strategies for software product lines," in *Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST'10)*, pp.459-468, Apr. 2010.

[11] C. H. P. Kim, D. S. Batory, and S. Khurshid, "Reducing combinatorics in testing product lines," in *Proceedings of the 10th International Conference on Aspect-oriented Software Development (AOSD'11)*, pp.57-68, Mar. 2011.

[12] M. Kowal, S. Schulze, and I. Schaefer, "Towards efficient SPL testing by variant reduction," in *Proceedings of the 4th International Workshop on Variability & Composition (VariComp'13)*, pp.1-6, Mar. 2013.

[13] D. Shimbara and O. Haugen, "Generating configurations for system testing with common variability language," *Lecture Notes in Computer Science*, Vol.9369, pp.221-237, 2015.

[14] M. Al-Hajjaji, S. Krieter, T. Thüm, M. Lochau, and G. Saake, "IncLing: Efficient product-line testing using incremental pairwise sampling," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE'16)*, pp.144-155, Oct. 2016.

[15] A. Strickler, J. A. P. Lima, S. R. Vergilio, and A. T. R. Pozo, "Deriving products for variability test of feature models with a hyper-heuristic approach," *Applied Soft Computing*, Vol.49, pp.1232-1242, 2016.

[16] A. Hervieua, D. Marijanb, A. Gotliebb, and B. Baudry, "Practical minimization of pairwise-covering test configurations using constraint programming," *Information and Software Technology*, Vol.71, pp.129-146, 2016.

[17] M. Al-Hajjaji, J. Krüger, S. Schulze, T. Leich. and G. Saake, "Efficient product-line testing using cluster-based product prioritization," in *Proceedings of the IEEE/ACM 12th International Workshop on Automation of Software Testing (AST'17)*, pp.16-22, May 2017.

[18] J. Shi, M. Cohen, and M. Dwyer, "Integration testing of software product lines using compositional symbolic execution," in *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering (FASE'12)*, pp.270-284, 2012.

[19] M. Al-Hajjaji, T. Thüm, J. Meinicke, M. Lochau, and G. Saake, "Similarity-based prioritization in software product-line testing," in *Proceedings of the 16th International Software Product Line Conference (SPLC'14)*, Sep. pp.197-206, 2014.

[20] J. A. Galindo, H. Turner, D. Benavides, and J. White, "Testing variability-intensive systems using automated analysis: An application to android," *Software Quality Journal*, Vol.24, pp.365-405, 2016.

[21] P. Ammann and J. Offutt, "Introduction to Software Testing," 2nd Ed., Cambridge University Press, 2016.

[22] L. Vidacs, F. Horvath, J. Mithalicza, B. Vancsics, and A. Beszedes, "Supporting software product line testing by optimizing code configuration coverage," in *Proceedings of the IEEE 8th International Conference on Software Testing, Verification and Validation Workshops (ICSTW'15)*, pp.1-7, 2015.

[23] E. Uzuncaova, S. Khurshid, and D. Batory, "Incremental test generation for software product lines," *IEEE Transactions on Software Engineering*, Vol.36, No.3, pp.309-322, 2010.

[24] S. Apel, D. Batory, C. Kastner, and G. Saake, "Feature-Oriented Software Product Lines: Concepts and Implementation," Springer, 2013.

[25] Java Code Coverage for Eclipse, [Internet], https://www.jacoco.org, last accessed in 27th Dec. 2021.

한 수 빈

https://orcid.org/0000-0001-8393-3436
e-mail : hanidiot@gmail.com
2017년 ~ 현 재 전북대학교
소프트웨어공학과 학사과정
관심분야 : Computer Graphics & Software Product Line

이 지 현

https://orcid.org/0000-0003-4512-806X
e-mail : jihyun30@jbnu.ac.kr
1993년 전북대학교 정보통신공학과(학사)
2000년 전북대학교 전자계산교육(석사)
2005년 전북대학교 컴퓨터과학과(박사)
2016년 ~ 현 재 전북대학교
소프트웨어공학과 교수
관심분야 : Software Product Line & Architecture Reconstruction

고 서 연

https://orcid.org/0000-0003-0669-4456
e-mail : tjduscn@jbnu.ac.kr
2020년 ~ 현 재 전북대학교
소프트웨어공학과 학사과정
관심분야 : Software Testing & Data Analysis