단위 테스트-코드 간 추적성 복구를 위한 시나리오 기반 시각화 도구의 비교 및 필요성 연구*

김대원O, 이영규, 허윤아, 유준범
건국대학교 컴퓨터공학부
{kimdw010130, sontouf, hya1202, jbyoo}@konkuk.ac.kr

A Comparative Study of Scenario-Based Visualization Tools for Unit Test-Code Traceability Recovery and Their Necessity

Daewon KimO, Younggyu Lee, Yoona Heo, Junbeom Yoo
Division of Computer Science and Engineering, Konkuk University
{kimdw010130, sontouf, hya1202, jbyoo}@konkuk.ac.kr

요 약

레거시 시스템에서 개발 산출물이 최신 상태로 유지되지 않거나 아예 존재하지 않는 경우, 코드가 수정되더라도 변경사항이 단위 테스트에 반영되지 않는 문제가 발생한다. 이처럼 시스템의 테스트를 설명하는 문서가 부재하거나 최신화되지 않은 경우, 개발자는 코드를 직접 분석해야 하며, 이 과정은 상당한 비용이 필요하다. 본 연구에서는 테스트 코드로부터 단위 테스트의 구조와 실행 흐름을 추출하고, 이를 시나리오 기반의 시퀀스 다이어그램으로 시각화하는 방안을 제안한다. 제안하고자 하는 방안은 테스트 시나리오의 시각화를 통해 테스트와 코드 간의 추적 관계를 명확히 하여, 테스트 유지보수의 효율성과 신뢰성을 높이는 것을 목표로 한다.

1. 서 론

레거시 시스템에서는 개발 산출물이 최신 상태로 유지되지 않거나 아예 존재하지 않는 경우, 코드가 수정되더라도 변경사항이 단위 테스트에 반영되지 않는 문제가 발생한다. 단위 테스트와 단위 코드 간 추적성의 단절은 결과적으로 테스트가 여전히 의도한 기능을 검증하는지를 판단하기 어렵게 만든다. 단위 테스트가 통과되었다고 해도, 해당 기능이 의도대로 동작하지 않는 상황이 발생할 수 있어, 개발자는 테스트 결과에 대해 신뢰하기 어렵다 [1]. 이를 해결하기 위해서는 개발자가 직접 코드와 테스트 간의 비교 및 분석을 해야 하며, 이는 많은 시간과 노력을 필요로 한다.

본 연구에서는 단위 테스트의 구조와 동작 흐름에 주목하여 이를 시각화하는 방안을 제안한다. 이때, 테스트가 어떤 흐름과 구조로 구성되었는지를 이해하고 설명하기 위한 표현 수단으로 테스트 시나리오와 시퀀스 다이어그램을 활용할 수 있다. 테스트 시나리오란 특정 기능이나 요구사항이 예상대로 동작하는지 검증하기 위해 구성된 입력, 실행 절차, 기대 결과의 일련의 과정을 말하며, 개발자의 의도와 시스템의 실제 동작을 연결해준다 [2]. 시퀀스 다이어그램은 UML (Unified Modeling Language) 표준 다이어그램으로, 객체 간의 상호작용을 시간 순서에 따라 표현하여 테스트의 실행 흐름을 분석하는 데 효과적으로 활용될 수 있다.

본 논문은 총 4장으로 구성된다. 2장에서는 본 연구의 적용 대상인 C++의 언어적 특성과 Google Test의 구조를 설명하고 시각화에 필요한 구성 요소를 정리한다. 3장에서는 관련된 정적 분석 도구들과 동적 분석 도구들의 기능을 조사하고, 추적성 복구 측면에서의 활용 여부를 분석한다. 4장에서는 논문의 결론으로 연구 결과를 정리하고, 시나리오 기반 추적성 복구를 위한 향후 연구 방향을 제시한다.

* 본 연구는 원자력안전위원회의 재원으로 소형모듈원자로규제연구추진단의 지원을 받아 수행한 원자력안전연구사업의 연구결과입니다. (No. RS-2024-00509643)

2. 단위 테스트 시각화를 위한 분석 대상

가장 먼저 C++ 및 GoogleTest 환경을 대상으로 시각화에 필요한 요소를 선정하였다. C++는 정적 언어이면서 런타임 메타정보가 부족하기 때문에 테스트 실행 흐름이나 객체 생성을 추적하기가 상대적으로 더 어렵고 [3], GoogleTest는 xUnit 계열의 다양한 테스트 구성요소를 명시적으로 포함하고 있는 프레임워크이기 때문에 시각화 대상의 구성이 복잡하다 [4]. 따라서 이 두 환경은 단위 테스트 시각화의 필요성을 드러내기에 적합하다.

2.1. C++의 언어적 특성

본 연구에서는 C++ 언어로 작성된 시스템을 대상으로 한다. C++의 언어적 특성 중 코드 실행 흐름의 정확한 추적을 어렵게 만드는 요소들은 다음과 같다:

1) 메모리 주소의 직접 참조: C++는 포인터를 통해 메모리 주소를 직접 다루기 때문에, 실행 시 객체의 정확한 상태를 외부에서 해석하기 어렵고 객체 식별이나 값 추적에 어려움이 있다.
2) 런타임 메타정보의 부족: C++에는 Java와 달리 reflection API나 풍부한 런타임 메타정보가 거의 제공되지 않는다. Java에서는 실행 중 객체 상호작용을 쉽게 수집할 수 있지만 [5], C++에서는 이러한 기능 부재로 인해 실행 추적을 위해 별도의 계측이나 디버거 훅 (hook)이 필요하다.
3) 다형성: C++ 템플릿은 컴파일 시에 타입별로 코드가 생성되고, 가상 함수는 런타임에 실제 구현이 결정된다. 정적 분석만으로는 이런 다형적 호출의 실제 대상 함수를 알기 어렵고, 템플릿

인스턴스화된 클래스들도 일일이 추론해야 하는 부담이 있다.

2.2. Google Test의 테스트 구성요소와 시각화 관점 분석

C++ 언어에서 가장 널리 사용하는 단위 테스트 프레임워크는 xUnit 계열의 GoogleTest (gtest) 이다 [6]. 개발자가 단위 테스트가 의도한 대로 동작하는 지를 판단하려면, 테스트 코드가 어떻게 구성되어 실행되는지를 이해하는 것이 중요하다. 특히 테스트 실행 전 초기 설정, 테스트 본문의 구조, 검증 절차 등은 테스트의 목적과 의미를 명확히 이해하는 데 핵심적인 요소이다. 이러한 구성요소는 테스트 시나리오 기반 시각화의 주요 대상이 되며, 이를 통해 테스트 흐름과 의도를 명확히 드러낼 수 있다. 예를 들어, 반복적인 초기화 및 정리 과정을 담당하는 Fixture는 테스트 구조 내에서 SetUp 및 TearDown과 결합되어 중요한 의미를 갖는다.

다음의 〈그림 1〉은 GoogleTest에서 Fixture가 정의된 경우를 포함한 xUnit 기반 테스트 프레임워크의 테스트 수행 절차를 도식화한 것이다. 〈표 1〉은 GoogleTest에서 사용하는 주요 개념 중 시각화에 반영되어야 할 핵심 구성요소들을 정리한 것이다. 테스트 시나리오 기반 시각화를 위해서는 이러한 구성요소들을 다이어그램 상에 명확히 표현할 필요가 있다.

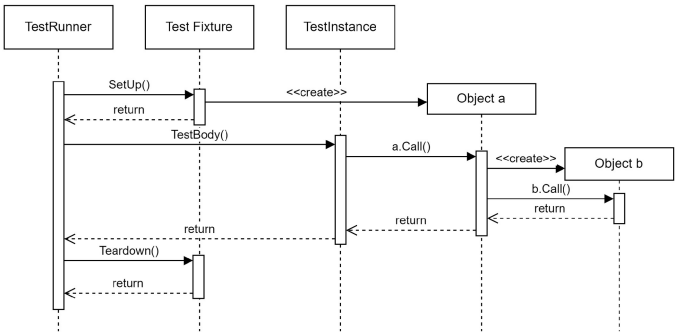

그림 1. xUnit 기반 Test Framework의 테스트 수행 절차

표 1. 시각화에 필요한 GoogleTest 개념

테스트 구성요소	역할
Fixture	테스트에 필요한 데이터나 환경을 준비
Stub	특정 기능을 dummy 객체로 대체하는 부분을 명시
Assertion	입력값과 기댓값을 명시
Parameterized Test	매개변수를 사용하여 같은 테스트를 여러 데이터로 실행하는 것을 식별
Test Case, Test Suite	특정한 기능을 검증하는 여러 개의 테스트가 어떤 그룹에 속하는지 표시
EventListener 등	이 외에 테스트 프레임워크에서 사용되는 객체들

3. 기존 시각화 도구 분석

시각화 도구는 정적 분석 도구 (static analysis tools)와 동적 분석 도구 (dynamic analysis tools)로 나뉘며, 각각의 도구는 시퀀스 다이어그램 생성을 위한 정보 수집 방식과 표현 방식에서 차이를 보인다. 정적 분석은 코드의 구문 구조를 기반으로 하여 함수 호출 관계, 클래스 간 의존성 등을 추출하는 반면, 동적 분석은 실제 프로그램 실행 중에 수집된 런타임 데이터를 바탕으로 시각화를 수행한다.

정적 분석 도구 중 대표적인 예로는 clang-uml [7], IntelliJ IDE Sequence Diagram plugin [8]이 있다. 이들 도구는 정적 코드 구조로부터 함수 간 호출 관계를 추출하여 시각적으로 표현할 수 있으나, 단위 테스트 환경에서 요구되는 실행 흐름 정보나 객체 상태 변화는 제공하지 못한다. 특히 clang-uml은 템플릿과 매크로 기반의 복잡한 코드 구조를 다룰 수 있는 장점이 있지만, 테스트 실행 시의 맥락이 반영되지 않으므로 테스트 시나리오 기반 시각화에는 적합하지 않다.

동적 분석 도구인 Enterprise Architect (EA) [9]는 런타임에 함수 호출 정보를 기록하여 시퀀스 다이어그램을 생성할 수 있으며, 실제로 Google Test 환경에서 실행한 코드의 흐름을 시각적으로 추적할 수 있는 유용한 기능을 제공한다. 그러나 이 도구 역시 객체 식별, 인자값 추적, 다중 인스턴스 구분 등 테스트 목적에 필요한 정보 표현에는 제약이 있다. DYNO [3]와 같은 연구 기반 도구는 런타임 정보를 수집하여 다이어그램을 생성하는 데 강점을 가지지만, 현재 공개된 구현체는 제한적이며 테스트 도구와의 통합성이 낮다.

즉, 정적 분석 도구들은 코드 상의 구조나 호출 관계는 보여주지만, 테스트의 실제 실행 맥락 (context)이나 객체의 상태 변화, 인자값 흐름 등은 표현하지 못한다는 한계가 있다. 반면, 동적 분석 도구들은 실행 중 수집한 정보를 바탕으로 시각화를 제공할 수 있으나, 테스트의 구성요소를 비롯하여 동적인 값과 상태의 정보까지 제공하는 데에는 한계가 있었다. 특히 객체의 생명주기나 동일 클래스 내의 인스턴스 구분, 인자 및 반환값과 같은 정보는 대부분의 도구에서 충분히 표현하지 못하였다.

3.1. 도구 간 비교 및 한계점

아래의 〈표 2〉는 정적 분석 및 동적 분석 도구에 대해 시각화 구성요소별 지원 여부를 비교한 결과이다. 특히 시각화 과정 중 다음의 네 가지 요소가 핵심적으로 요구된다.

1) 생명주기의 시각화: 객체의 생성 및 소멸을 명확하게 표현하는 생명주기의 시각화가 필요하다.
2) 객체별 라이프라인의 구별: 동일한 클래스에 속하더라도 서로 다른 인스턴스를 개별 객체로 구분해 표현하는 객체별 라이프라인의 구별이 중요하다.
3) 테스트 구성요소의 시각화: 2.2절에서 언급된 테스트 흐름 내에서 Fixture, Setup, Assertion 등 Google Test의 구성요소들을 시각적으로 표현해야 한다.
4) 함수 호출 시 전달되는 값의 흐름 시각화: 함수 호출 시 전달되는 인자값과 반환값의 흐름을 다이어그램 상에서 명확히 보여주는 기능이 필요하다.

표 2. 도구별 시각화 구성요소 지원현황

도구 / 시각화 구성요소	SequenceDiagram for C++ (JetBrains Plugin)	UModel (Altova)	clang-uml	DYNO	Enterprise Architect
분석 방식	정적	정적	정적	동적	동적
메서드 호출 흐름	O	X	O	O	O
조건문 및 분기 구조 표시	O	O	O	X	X
오픈소스	X	X	O	X	X
객체별 생명주기 표시	X	X	X	O	O
객체별 라이프라인 표시	X	X	X	X	X
GoogleTest 테스트 구성요소 표시	X	X	X	X	X
인자값, 반환값 표시	X	X	X	X	X

4. 결론 및 향후 연구 방향

4.1. 연구 결과

본 논문은 먼저 레거시 시스템에서 발생하는 단위 테스트와 실제 코드 사이의 추적성 단절 문제를 해결하기 위한 접근으로써 테

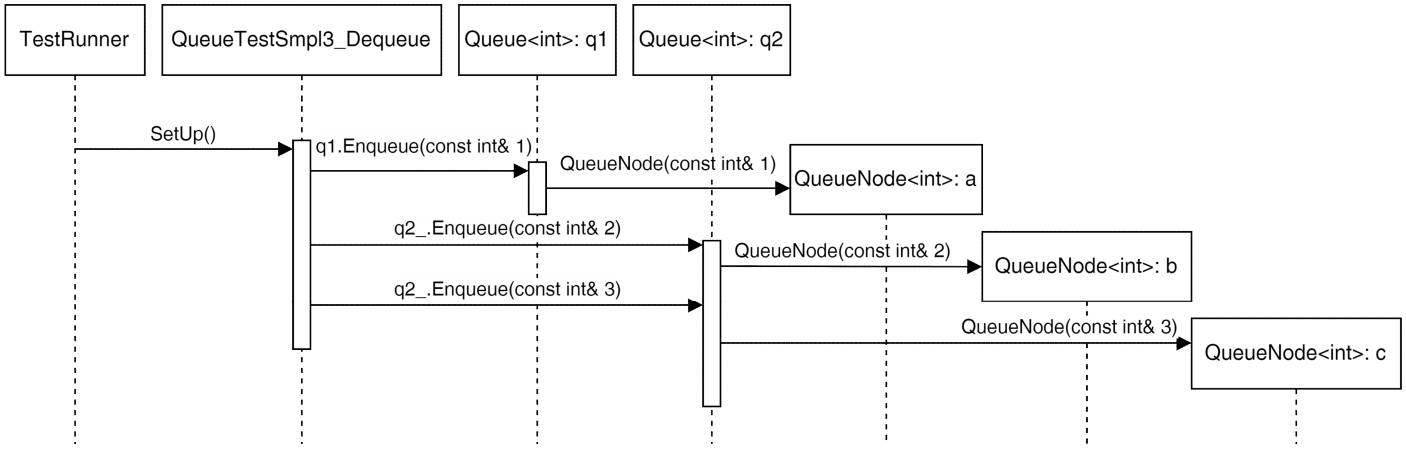

그림 2. GoogleTest sample code 3의 SetUp을 나타낸 시나리오 기반 시퀀스 다이어그램

스트 시나리오 기반 시각화의 필요성과 방향성을 살펴보았다. 다음으로, C++의 언어적 특성을 분석하였으며, 단위 테스트의 실행 흐름과 구성요소 중 시각화에 필요한 요소들을 선별하였다. 그리고 그 요소들이 기존 시각화 도구들에서 표현되는 방식과 기존 도구들의 한계점을 분석하였다.

단위 테스트와 코드 간의 추적성을 효과적으로 복구하기 위해서는 단순히 함수 호출 관계만을 시각화하는 것 이상으로, 테스트 실행의 맥락을 중심으로 구성요소의 의미와 흐름을 표현하는 방식이 필요하다. 즉, 어떤 객체가 어떤 시점에 생성되고 어떤 인자를 전달받아 호출되었는지, 해당 테스트가 어떤 조건을 전제로 하고 어떤 기대 결과를 검증하고자 하는지를 시각화 과정에서 드러낼 수 있어야 한다. 기존의 도구들은 이러한 요구사항을 충족하지 못하고 있으며, 추적성 복구를 위한 시각화 도구의 기능 확장 또는 새로운 접근 방식이 필요하다. 〈그림 2〉는 개발할 것으로 기대되는 도구를 통해 그려질 시퀀스 다이어그램의 예시이다. QueueNode〈int〉의 a, b, c는 실행 시 할당되는 임의의 주소값이다. 2.2절에서 언급된 테스트 구성요소가 포함되었으며, GoogleTest에서 제공되는 테스트 샘플 중 세 번째 샘플에 대하여 직접 그린 시퀀스 다이어그램이다.

4.2. 추적성 복구를 위한 후속 연구 방향

단위 테스트와 코드 간의 추적성 단절 문제를 보다 효과적으로 해결하기 위해서는, 테스트의 실행 흐름과 의도를 정밀하게 반영할 수 있는 맞춤형 시각화 기법의 개발이 필요하다. 기존 시각화 도구들은 일반적인 프로그램의 제어 흐름이나 함수 호출 관계 중심으로 설계되어 있어, 단위 테스트가 내포하는 목적, 전제 조건, 기대 결과와 같은 핵심 요소들을 명확히 표현하기 어렵다는 한계가 있다.

이에 따라, 단위 테스트 특유의 구조와 실행 절차를 정교하게 반영할 수 있는 테스트 도메인 특화 시각화 방식이 요구된다. 구체적으로 테스트 프레임워크의 실행 생명주기를 명시적으로 모델링하고, 테스트의 목적과 의미를 드러내는 의도 기반 메타정보(semantic metadata)를 함께 통합하는 접근이 필요하다. 예를 들어, SetUp 구성의 역할이나 Assertion의 의미를 다이어그램 내에서 시각적으로 강조함으로써, 개발자는 테스트의 목적과 실행 맥락을 빠르게 이해할 수 있으며, 코드 해석에 소요되는 인지적 부담도 크게 줄일 수 있다.

본 연구에서는 C++에 초점을 맞추었지만, 축적된 지식을 바탕으로 Python의 PyTest, Java의 JUnit 등 다른 언어의 단위 테스트에도 시나리오 기반 시각화 개념을 확장 적용할 수 있을 것이다.

Reference

[1] B. Li, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, and N. A. Kraft, "Automatically Documenting Unit Test Cases," Proceedings of the 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST), IEEE, 2016.

[2] C. Kaner, "An Introduction to Scenario Testing," Florida Institute of Technology, Technical Report, April 2013.

[3] T. Raitalaakso, "Dynamic Visualization of C++ Programs with UML Sequence Diagrams," Master's Thesis, Tampere University of Technology, Finland, 2000P.

[4] B. Cornelissen, A. van Deursen, L. Moonen, and A. Zaidman, "Visualizing Testsuites to Aid in Software Understanding," Technical Report TUD-SERG-2006-003, Software Engineering Research Group, Delft University of Technology, 2006.

[5] V. Gestwicki and B. Jayaraman, "JIVE: Java Interactive Visualization Environment," Proceedings of the OOPSLA '04 Conference, ACM, 2004.

[6] Google, "GoogleTest Primer," [Online]. Available: https://google.github.io/googletest/primer.html

[7] B. Kryza, "clang-uml: C++ to UML diagram generator (Version 0.6.1)," GitHub, Jan. 2025. [Online]. Available: https://github.com/bkryza/clang-uml

[8] SequenceDiagram Core Plugin, "SequenceDiagram Core Plugin for JetBrains IDEs (Version 4.x)," JetBrains Marketplace, [Online]. Available: https://plugins.jetbrains.com/plugin/8286-sequencediagram-core

[9] Sparx Systems, "Visualize, Debug and Profile Executing Code in Enterprise Architect," Sparx Systems Web Documentation, [Online]. Available: https://sparxsystems.com/enterprise-architect/visualize-debug-profile/visualize-debug-profile.html