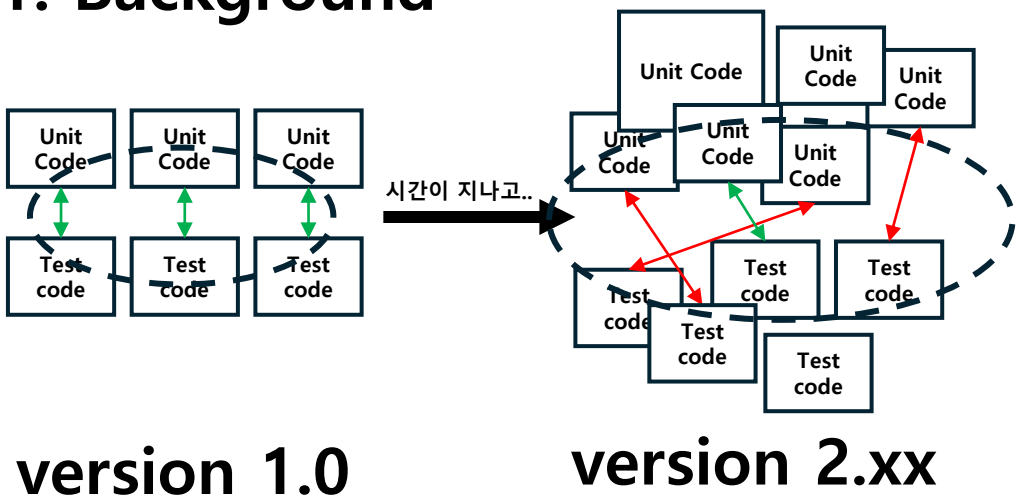
단위 테스트-코드 간의 추적성 복구

1. Background

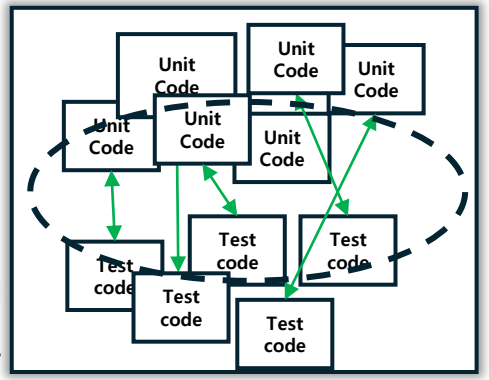


2. Problem

1. 단위 테스트와 코드간의 추적성 관계에 대한 문서화 부족

2. 개발자가 일일이 코드를 읽는데 발생하는 시간, 인적자원 비용문제

3. C++ 프로젝트 대상, 기존의 정적, 동적도구의 기능 부족



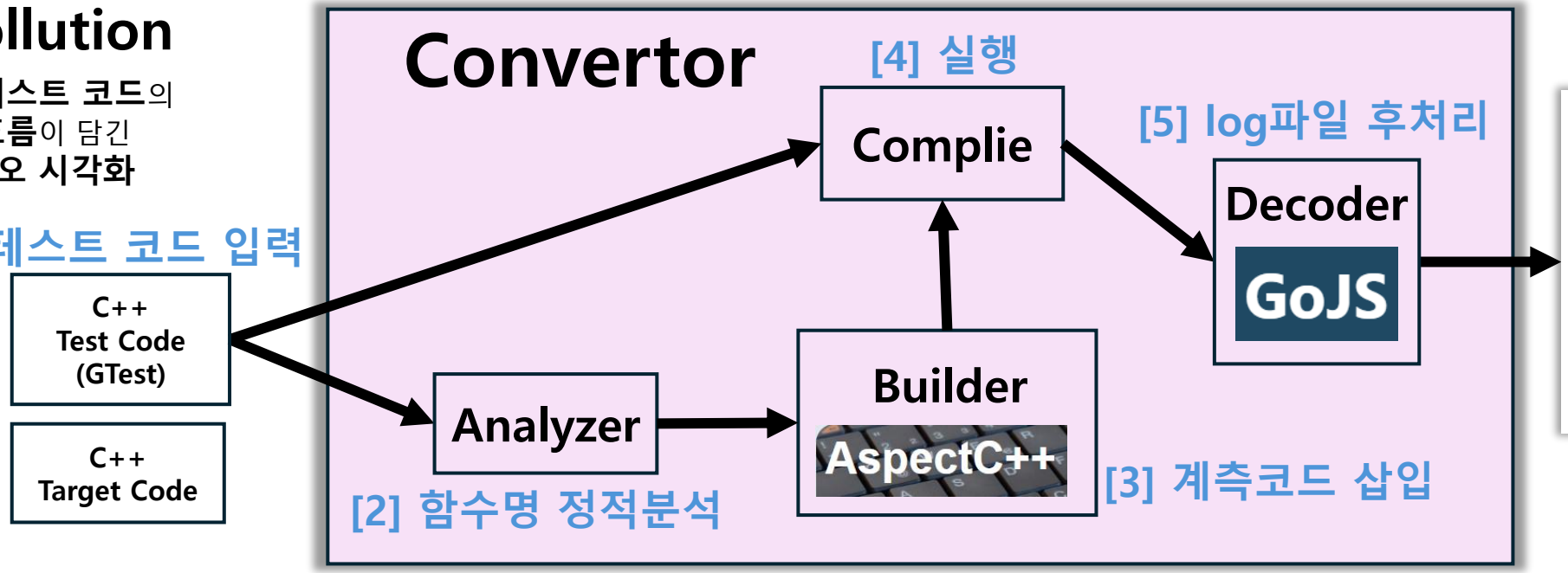
[그림 1] 단위 테스트와 코드간의 추적성 그래프

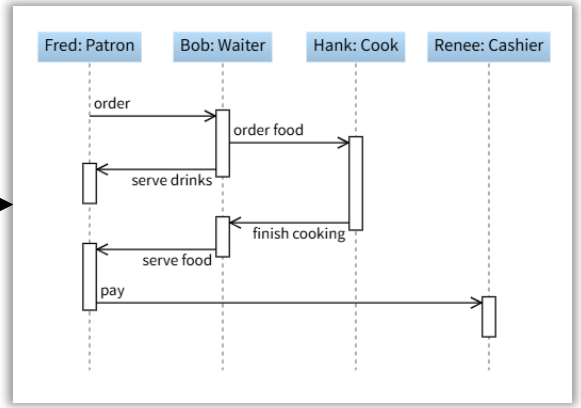
도구 / 시각화 구성요소	SequenceDiagram for C++(JetBrains Plugin)	UModel (Altova)	clang-uml	DYNO	Enterprise Architect
분석 방식	정적	정적	정적	동적	동적
메서드 호출 흐름	O	X	O	O	O
조건문 및 분기 구조 표시	O	O	O	X	X
오픈소스	X	X	O	X	X
객체 라이프 사이클 표시	X	X	X	O	O
객체별 라이프라인 표시	X	X	X	X	X
GoogleTest 테스트 구성요소 표시	X	X	△	X	△
인자값, 반환값 표시	X	X	X	X	△

[그림 2] 기존 도구들의 기능 비교표

3. Sollution

- 단위 테스트 코드의
- 실행 흐름이 담긴
- 시나리오 시각화





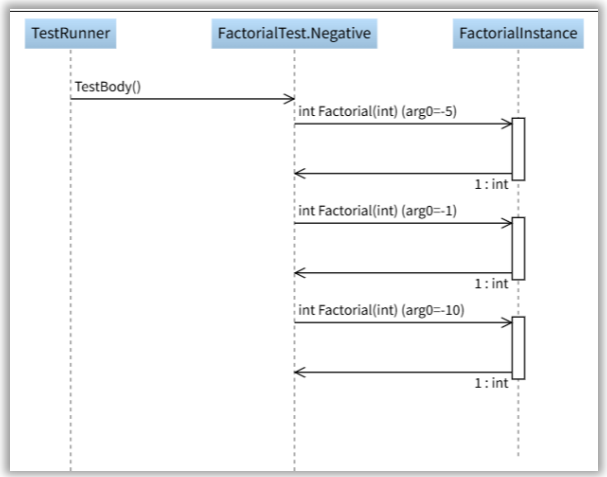
Sequence diagram

4. Process

```
#include "sample1.h"

#include <limits.h>

#include "gtest/gtest.h"
namespace {

TEST(FactorialTest, Negative) {
  // This test is named "Negative", and belongs to the "FactorialTest"
  // test case.
  EXPECT_EQ(1, Factorial(-5));
  EXPECT_EQ(1, Factorial(-1));
  EXPECT_GT(Factorial(-10), 0);
}

// Tests factorial of 0.
TEST(FactorialTest, Zero) { EXPECT_EQ(1, Factorial(0)); }

// Tests factorial of positive numbers.
TEST(FactorialTest, Positive) {
  EXPECT_EQ(1, Factorial(1));
  EXPECT_EQ(2, Factorial(2));
  EXPECT_EQ(6, Factorial(3));
  EXPECT_EQ(40320, Factorial(8));
}

// Tests IsPrime()

// Tests negative input.
TEST(IsPrimeTest, Negative) {
  // This test belongs to the IsPrimeTest test case.

  EXPECT_FALSE(IsPrime(-1));
  EXPECT_FALSE(IsPrime(-2));
  EXPECT_FALSE(IsPrime(INT_MIN));
}

// Tests some trivial cases.
TEST(IsPrimeTest, Trivial) {
  EXPECT_FALSE(IsPrime(0));
  EXPECT_FALSE(IsPrime(1));
  EXPECT_TRUE(IsPrime(2));
  EXPECT_TRUE(IsPrime(3));
}

// Tests positive input.
TEST(IsPrimeTest, Positive) {
  EXPECT_FALSE(IsPrime(4));
  EXPECT_TRUE(IsPrime(5));
  EXPECT_FALSE(IsPrime(6));
  EXPECT_TRUE(IsPrime(23));
}
```

```
[LOG    ] Run 시작: void testing::Test::Run()
[FactorialTest.Negative]  [CALL   ] (0) int Factorial(int) (arg0=-5)
[FactorialTest.Negative]  [RETURN1] (0) int Factorial(int) => 1
[FactorialTest.Negative]  [CALL   ] (0) int Factorial(int) (arg0=-1)
[FactorialTest.Negative]  [RETURN1] (0) int Factorial(int) => 1
[FactorialTest.Negative]  [CALL   ] (0) int Factorial(int) (arg0=-10)
[FactorialTest.Negative]  [RETURN1] (0) int Factorial(int) => 1
[LOG    ] Run 끝: void testing::Test::Run()

[LOG    ] Run 시작: void testing::Test::Run()
[FactorialTest.Zero]  [CALL   ] (0) int Factorial(int) (arg0=0)
[FactorialTest.Zero]  [RETURN1] (0) int Factorial(int) => 1
[LOG    ] Run 끝: void testing::Test::Run()

[LOG    ] Run 시작: void testing::Test::Run()
[FactorialTest.Positive]  [CALL   ] (0) int Factorial(int) (arg0=1)
[FactorialTest.Positive]  [RETURN1] (0) int Factorial(int) => 1
[FactorialTest.Positive]  [CALL   ] (0) int Factorial(int) (arg0=2)
[FactorialTest.Positive]  [RETURN1] (0) int Factorial(int) => 2
[FactorialTest.Positive]  [CALL   ] (0) int Factorial(int) (arg0=3)
[FactorialTest.Positive]  [RETURN1] (0) int Factorial(int) => 6
[FactorialTest.Positive]  [CALL   ] (0) int Factorial(int) (arg0=8)
[FactorialTest.Positive]  [RETURN1] (0) int Factorial(int) => 40320
[LOG    ] Run 끝: void testing::Test::Run()

[LOG    ] Run 시작: void testing::Test::Run()
[IsPrimeTest.Negative]  [CALL   ] (0) bool IsPrime(int) (arg0=-1)
[IsPrimeTest.Negative]  [RETURN1] (0) bool IsPrime(int) => 0
[IsPrimeTest.Negative]  [CALL   ] (0) bool IsPrime(int) (arg0=-2)
[IsPrimeTest.Negative]  [RETURN1] (0) bool IsPrime(int) => 0
[IsPrimeTest.Negative]  [CALL   ] (0) bool IsPrime(int) (arg0=-2147483648)
[IsPrimeTest.Negative]  [RETURN1] (0) bool IsPrime(int) => 0
[LOG    ] Run 끝: void testing::Test::Run()

[LOG    ] Run 시작: void testing::Test::Run()
[IsPrimeTest.Trivial]  [CALL   ] (0) bool IsPrime(int) (arg0=0)
[IsPrimeTest.Trivial]  [RETURN1] (0) bool IsPrime(int) => 0
[IsPrimeTest.Trivial]  [CALL   ] (0) bool IsPrime(int) (arg0=1)
[IsPrimeTest.Trivial]  [RETURN1] (0) bool IsPrime(int) => 0
[IsPrimeTest.Trivial]  [CALL   ] (0) bool IsPrime(int) (arg0=2)
[IsPrimeTest.Trivial]  [RETURN1] (0) bool IsPrime(int) => 1
[IsPrimeTest.Trivial]  [CALL   ] (0) bool IsPrime(int) (arg0=3)
[IsPrimeTest.Trivial]  [RETURN1] (0) bool IsPrime(int) => 1
[LOG    ] Run 끝: void testing::Test::Run()

[LOG    ] Run 시작: void testing::Test::Run()
[IsPrimeTest.Positive]  [CALL   ] (0) bool IsPrime(int) (arg0=4)
[IsPrimeTest.Positive]  [RETURN1] (0) bool IsPrime(int) => 0
[IsPrimeTest.Positive]  [CALL   ] (0) bool IsPrime(int) (arg0=5)
[IsPrimeTest.Positive]  [RETURN1] (0) bool IsPrime(int) => 1
[IsPrimeTest.Positive]  [CALL   ] (0) bool IsPrime(int) (arg0=6)
[IsPrimeTest.Positive]  [RETURN1] (0) bool IsPrime(int) => 0
[IsPrimeTest.Positive]  [CALL   ] (0) bool IsPrime(int) (arg0=23)
[IsPrimeTest.Positive]  [RETURN1] (0) bool IsPrime(int) => 1
[LOG    ] Run 끝: void testing::Test::Run()
```

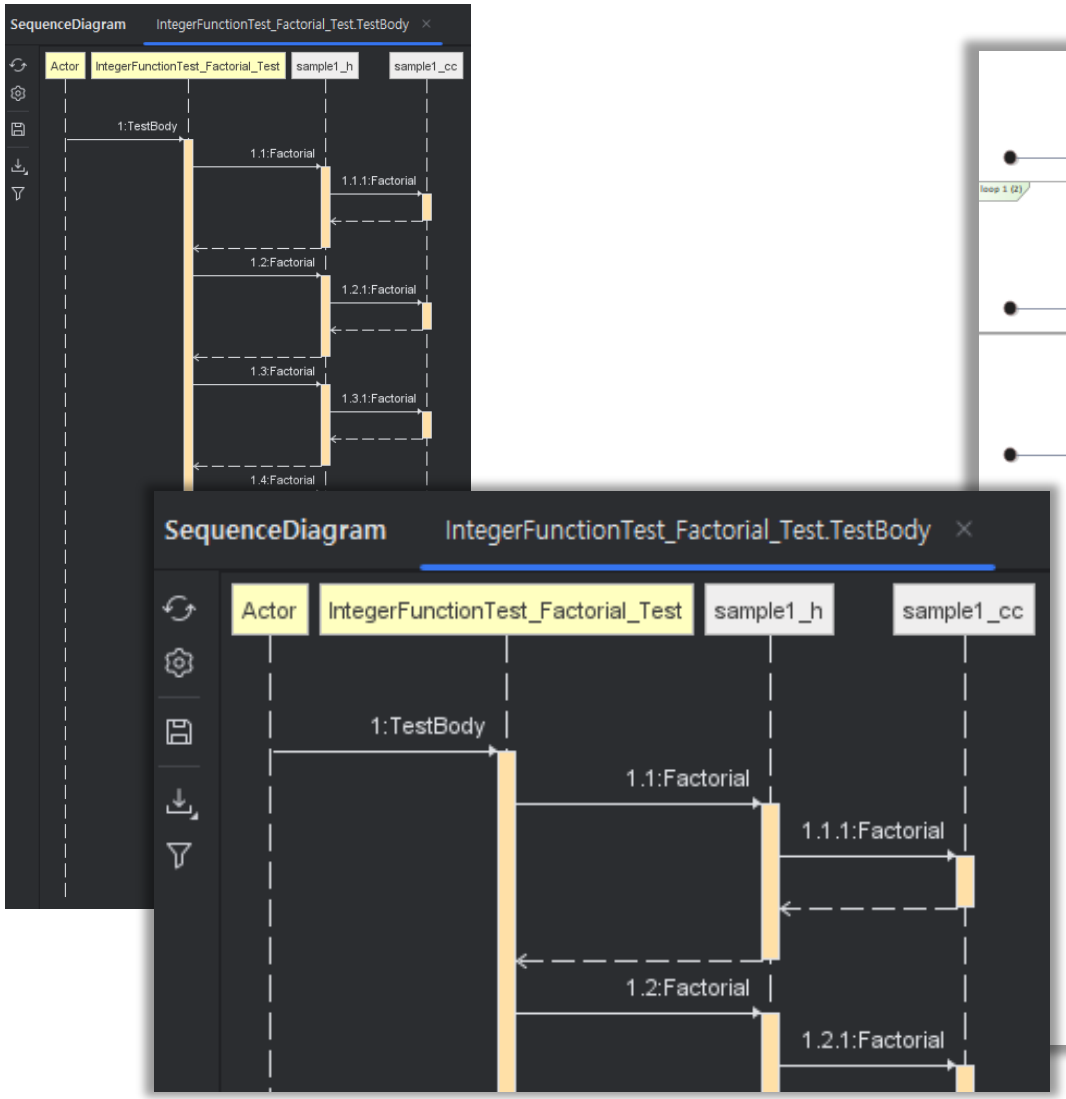

```
{
  "class": "go.GraphLinksModel",
  "nodeDataArray": [
    {
      "key": "TestRunner",
      "text": "TestRunner",
      "isGroup": true,
      "loc": "0 0",
      "duration": 16
    },
    {
      "key": "FactorialTest.Negative",
      "text": "FactorialTest.Negative",
      "isGroup": true,
      "loc": "180 0",
      "duration": 15
    },
    {
      "key": "FactorialInstance",
      "text": "FactorialInstance",
      "isGroup": true,
      "loc": "360 0",
      "duration": 14
    },
    {
      "group": "FactorialInstance",
      "start": 2,
      "duration": 2
    },
    {
      "group": "FactorialInstance",
      "start": 6,
      "duration": 2
    },
    {
      "group": "FactorialInstance",
      "start": 10,
      "duration": 2
    }
  ],
  "linkDataArray": [
    {
      "from": "TestRunner",
      "to": "FactorialTest.Negative",
      "text": "TestBody()",
      "time": 1
    },
    {
      "from": "FactorialTest.Negative",
      "to": "FactorialInstance",
      "text": "int Factorial(int) (arg0=-5)",
      "time": 2
    },
    {
      "from": "FactorialInstance",
      "to": "FactorialTest.Negative",
      "text": "1 : int",
      "time": 4
    },
    {
      "from": "FactorialTest.Negative",
      "to": "FactorialInstance",
      "text": "int Factorial(int) (arg0=-1)",
      "time": 6
    },
    {
      "from": "FactorialInstance",
      "to": "FactorialTest.Negative",
      "text": "1 : int",
      "time": 8
    },
    {
      "from": "FactorialTest.Negative",
      "to": "FactorialInstance",
      "text": "int Factorial(int) (arg0=-10)",
      "time": 10
    },
    {
      "from": "FactorialInstance",
      "to": "FactorialTest.Negative",
      "text": "1 : int",
      "time": 12
    }
  ]
}
```

[1]unit test code **[2]계측코드 삽입 후** **[3] json 변환** **[4] 시각화**

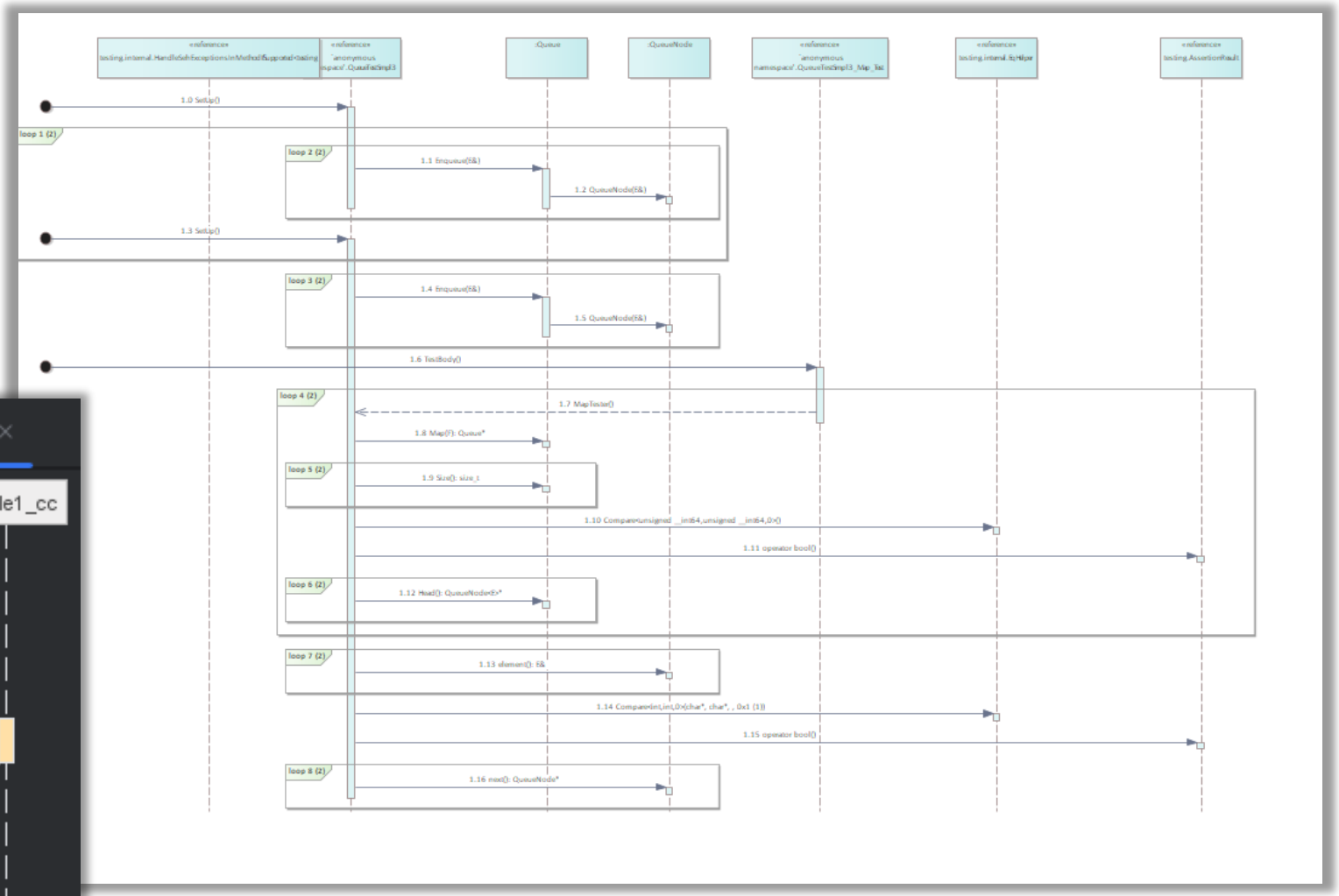
4. 기존 도구의 기능지원 한계점

SequenceDiagram (jetbrain plugin)



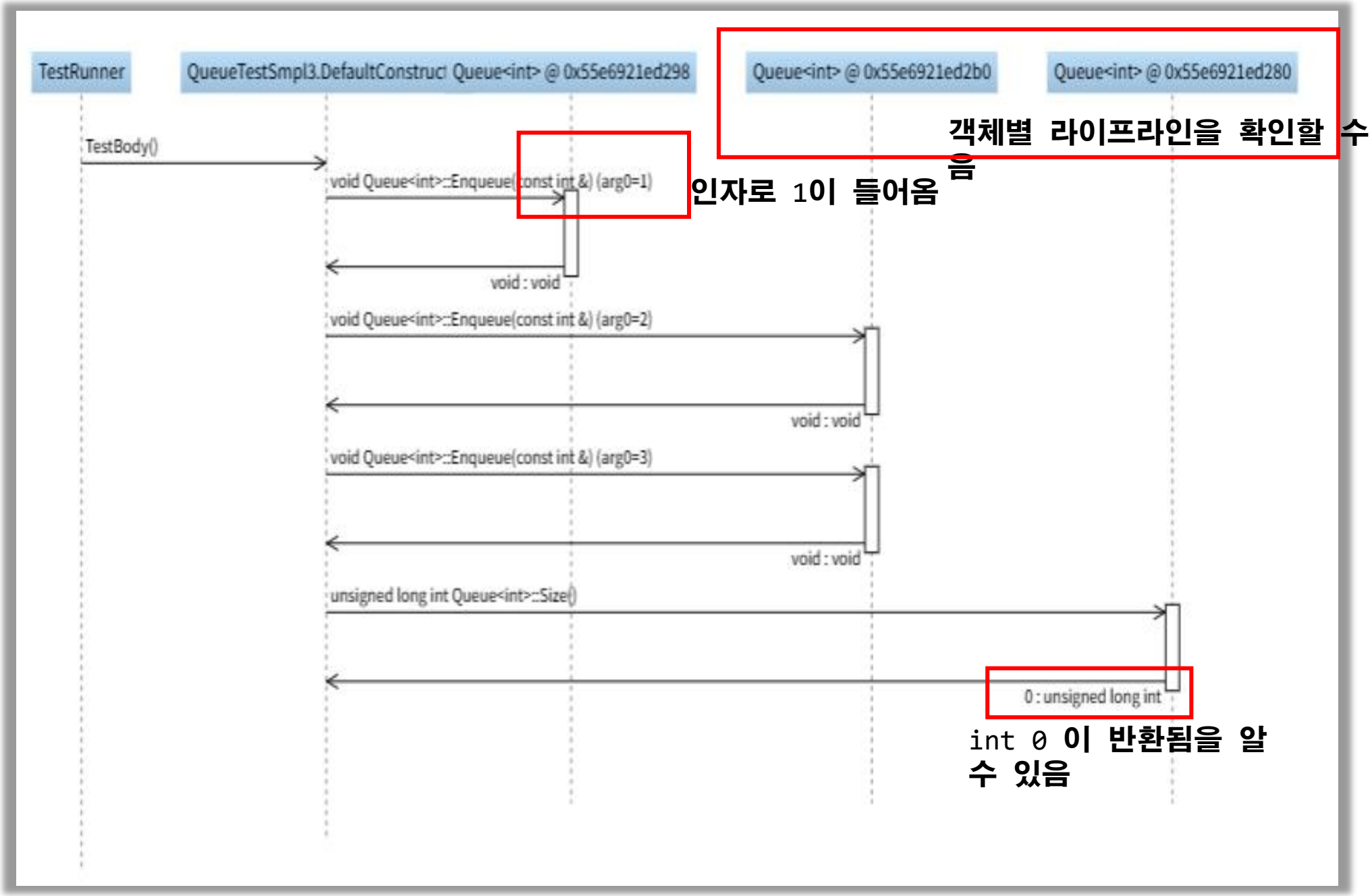
- 함수 호출 시의 파라미터, 리턴 값이 명시되지 않음

Enterprise Architect



- 객체별 라이프라인이 정확하지 않음. 클래스 기반으로 그려져 다른 객체가 호출할 시 이를 파악할 수 없음

5. 현재 도구의 보완점

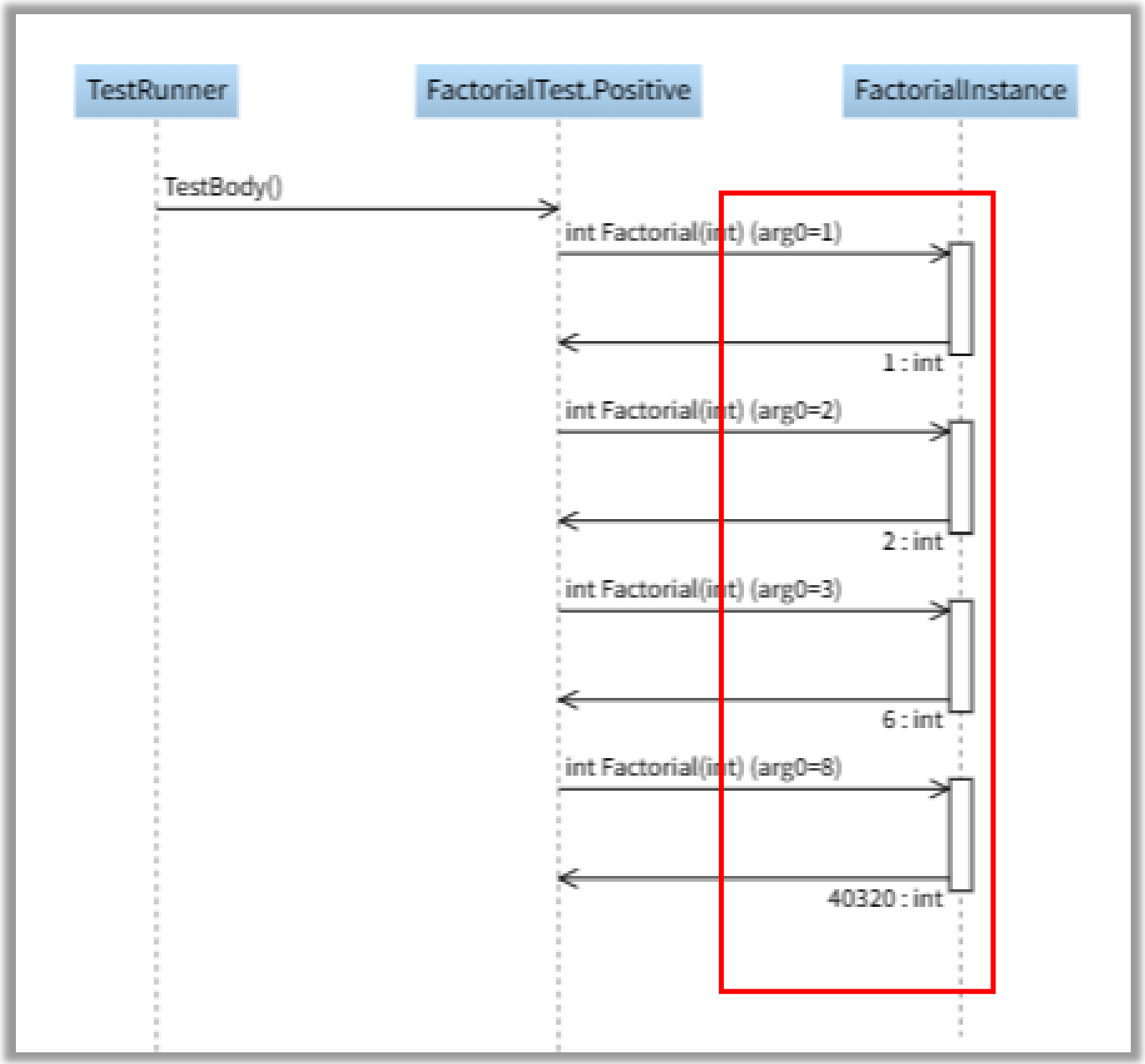


객체별 라이프라인을 확인할 수 있음

인자로 1이 들어옴

int 0 이 반환됨을 알 수 있음

5. 현재 도구의 보완점




```
input : 1
Factorial(input) : 1

input : 2
Factorial(input) : 2

input : 3
Factorial(input) : 6

input : 8
Factorial(input) : 40320
```