졸업 프로젝트

2nd Demo

Introduction

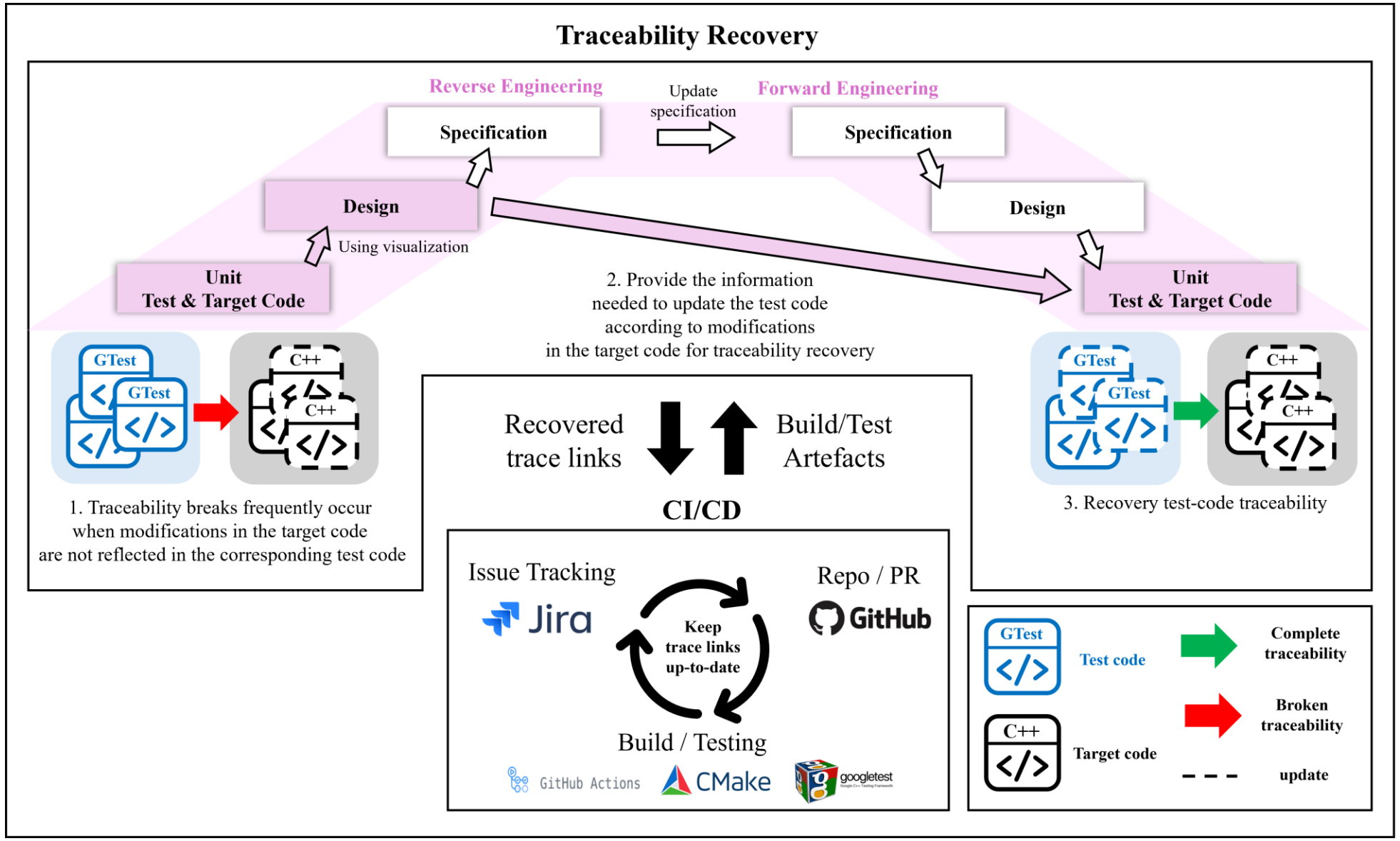


그림 1. 제안하는 Traceability Recovery 방법

Scope

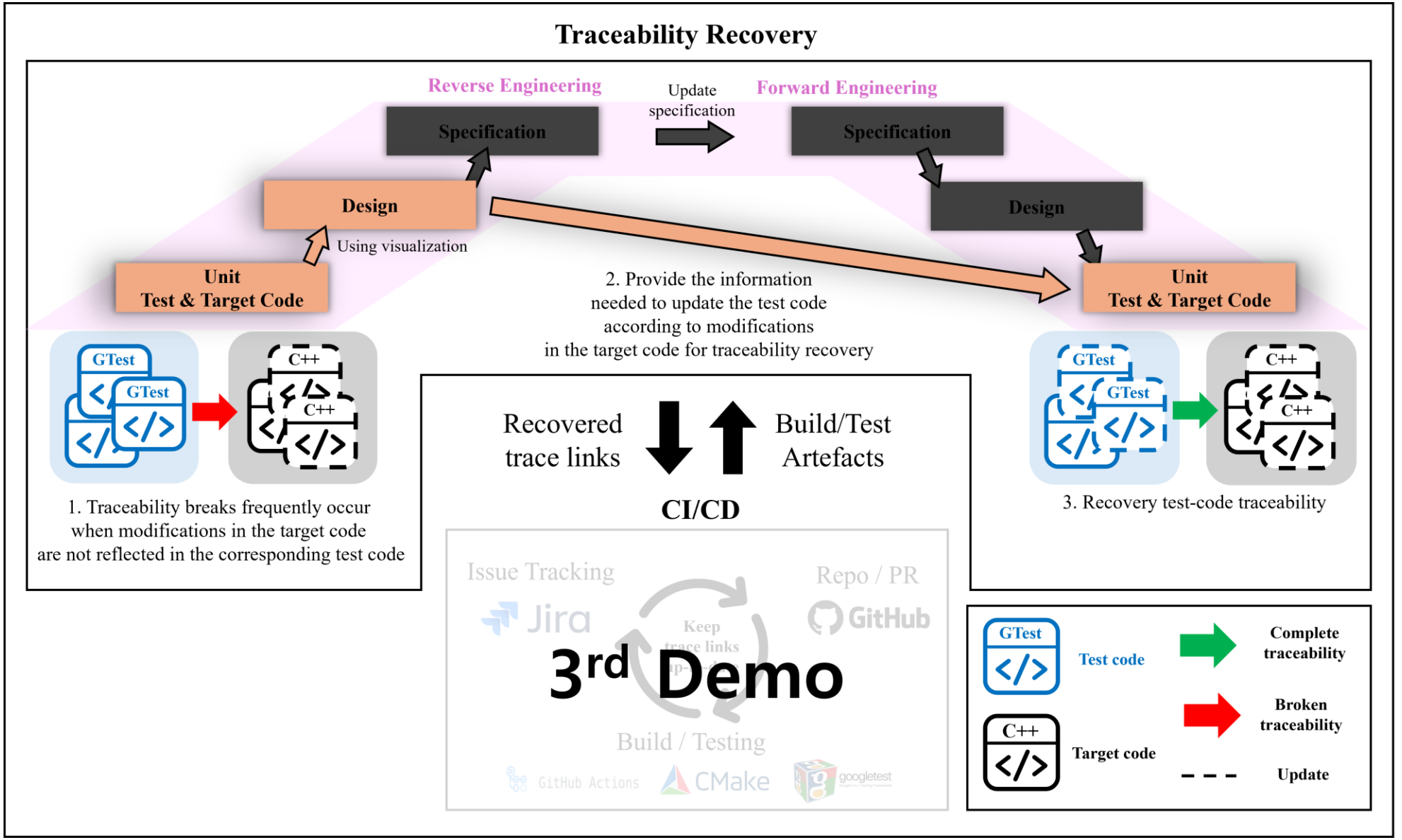


그림 1. 제안하는 Traceability Recovery 방법

Introduction

Introduction

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

그림 2. 도구별 시각화 구성요소 지원현황

Our Tool
동적
O
X
O
O
O
O
O

우리 도구의 목표

Process

```
static int Double(int n) { return 2 * n; }

void MapTester(const Queue<int>* q) {
  const Queue<int>* const new_q = q->Map(Double);
  ASSERT_EQ(q->Size(), new_q->Size());
  for (const QueueNode<int>*n1 = q->Head(), *n2 = new_q->Head();
       n1 != nullptr; n1 = n1->next(), n2 = n2->next()) {
    EXPECT_EQ(2 * n1->element(), n2->element());
  }

  delete new_q;
}
Queue<int> q0_;
Queue<int> q1_;
Queue<int> q2_;
};
```

그림 1. 기존의 소스 코드

1. 기존 코드에 계측코드를 삽입함.

Input → TraceRecoviz → output

```
static int Double(int n) {trace_enter((void*)0, __PRETTY_FUNCTION__ , n);
 { auto __trace_ret = 2 * n; trace_return((void*)0, __PRETTY_FUNCTION__, __trace_ret); return __trace_ret; } }
void MapTester(const Queue<int>* q) {trace_enter((void*)this, __PRETTY_FUNCTION__ , q);
  const Queue<int>* const new_q = q->Map(Double);
  ASSERT_EQ_LOG(q->Size(), new_q->Size());
  for (const QueueNode<int>*n1 = q->Head(), *n2 = new_q->Head();
       n1 != nullptr; n1 = n1->next(), n2 = n2->next()) {
    EXPECT_EQ_LOG(2 * n1->element(), n2->element());
  }

  delete new_q;
}
Queue<int> q0_;
Queue<int> q1_;
Queue<int> q2_;
};
```

그림 2. 계측코드가 삽입된 소스 코드

2. 계측코드가 삽입된 코드를 빌드함.

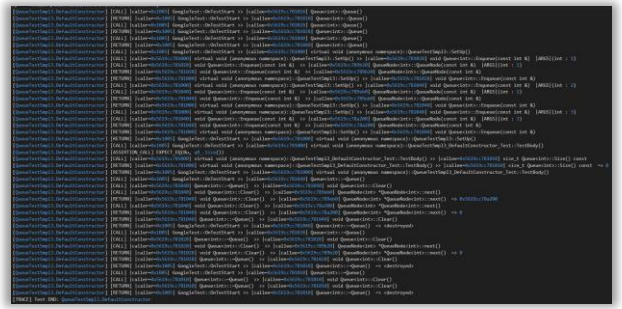


그림 3. 빌드되어 나온 log 파일

3. log를 gojs format에 맞게 변환

그림 4. json file

4. json파일을 gojs를 사용하여 html로 업로드

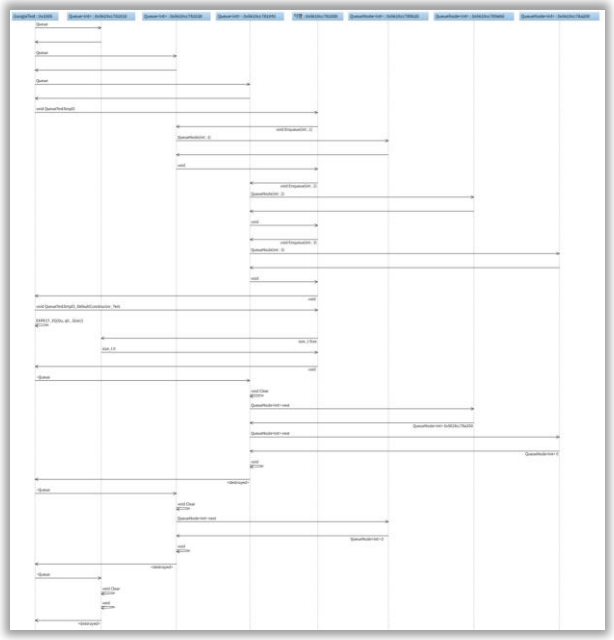


그림 5. 최종 시퀀스 다이어그램

Demo

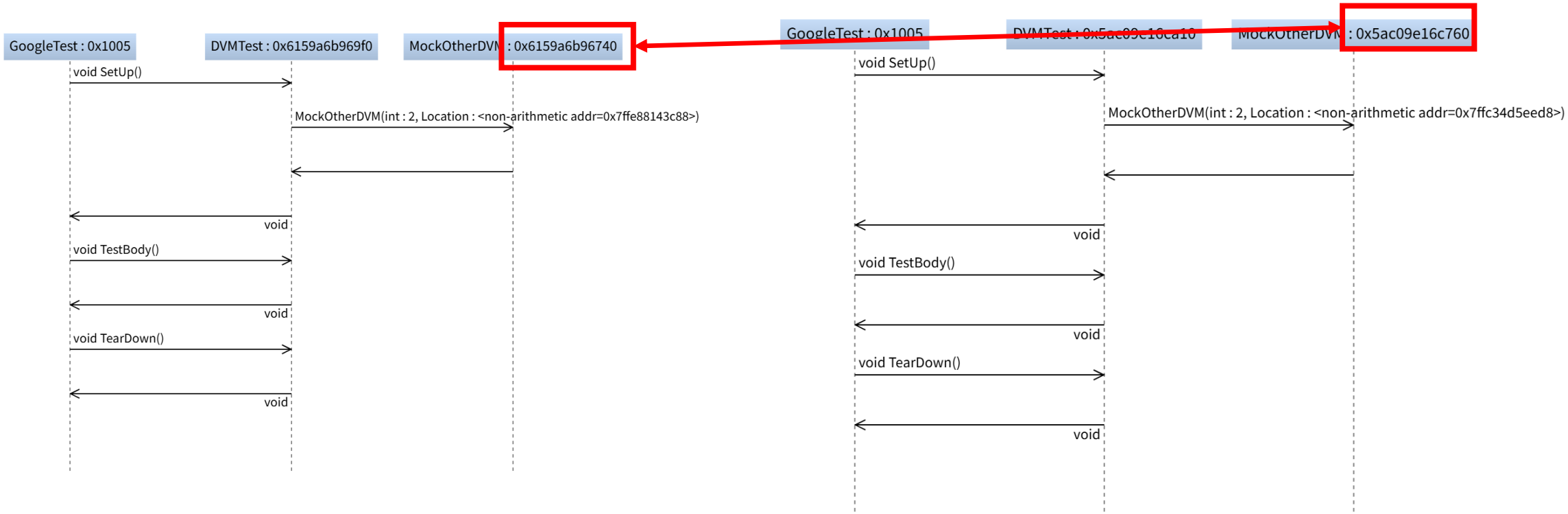
Limitations & Improvements

GitHub Actions

- CI/CD 중 시퀀스 다이어그램이 생성되도록 구성
- Github Action을 사용하여 프로그램을 동작 예정
- Github page을 사용하여 웹 브라우저를 통해 볼 수 있도록 배포

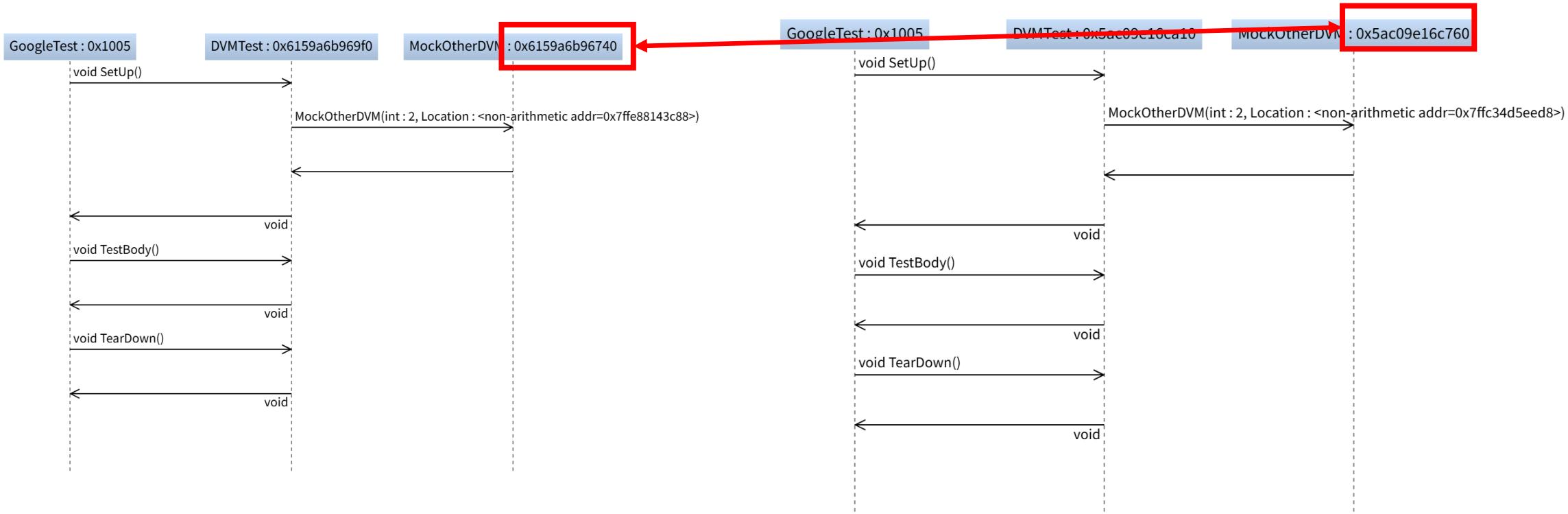
향후 개선 방향

- 같은 객체이나 주소값이 서로 다름
- 추후 매칭 알고리즘이나 개발자가 수정할 수 있게 개선 예정



Limitations & Improvements

- 같은 객체이나 주소값이 서로 다름
- 추후 매칭 알고리즘이나 개발자가 수정할 수 있게 개선 예정



감사합니다

Catagorization

- Gtest Smaple3에
리펙토링 방식 중 메서드 추출 (Extract Method) 적용

```
+   template <typename F>
+   void PushTransformed(Queue* out, F function) const {
+     for (const QueueNode<E>* node = head_; node != nullptr; node = node->next_) {
+       out->Enqueue(function(node->element()));
+     }
+   }
    template <typename F>
    Queue* Map(F function) const {
      Queue* new_queue = new Queue();
-     for (const QueueNode<E>* node = head_; node != nullptr; node = node->next_) {
-       new_queue->Enqueue(function(node->element()));
-     }
+     PushTransformed(new_queue, function);
      return new_queue;
    }
```

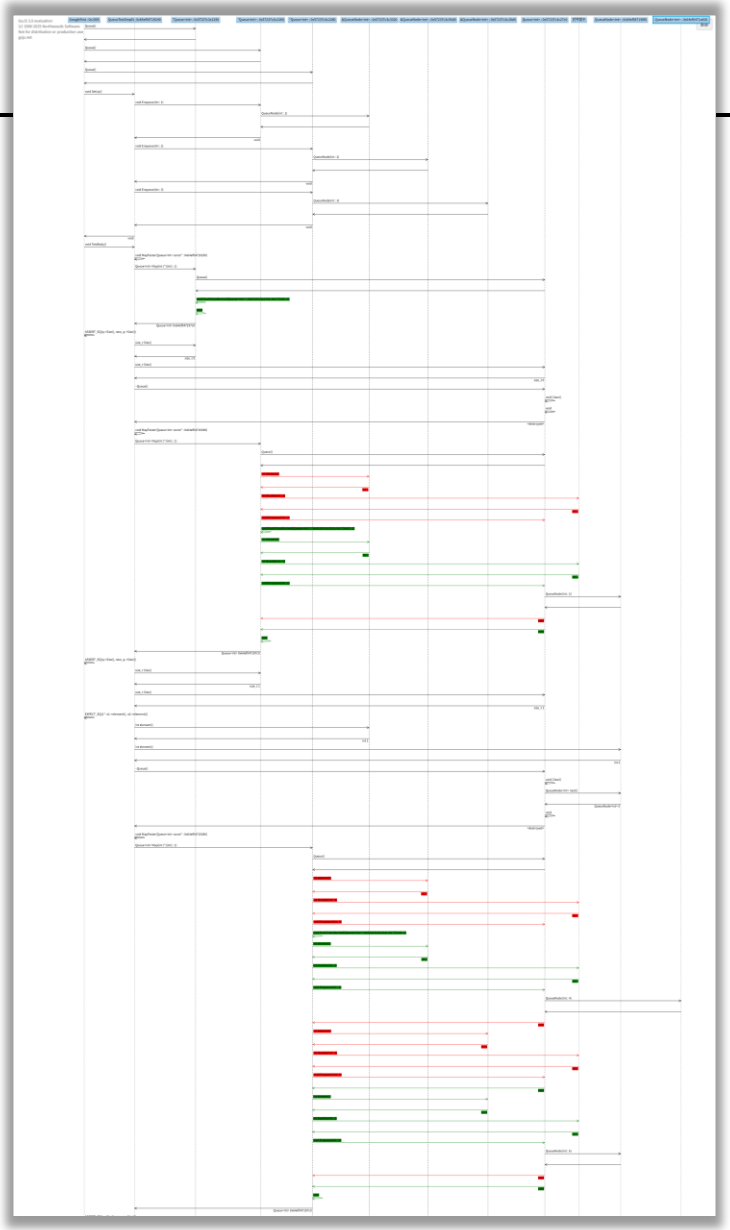
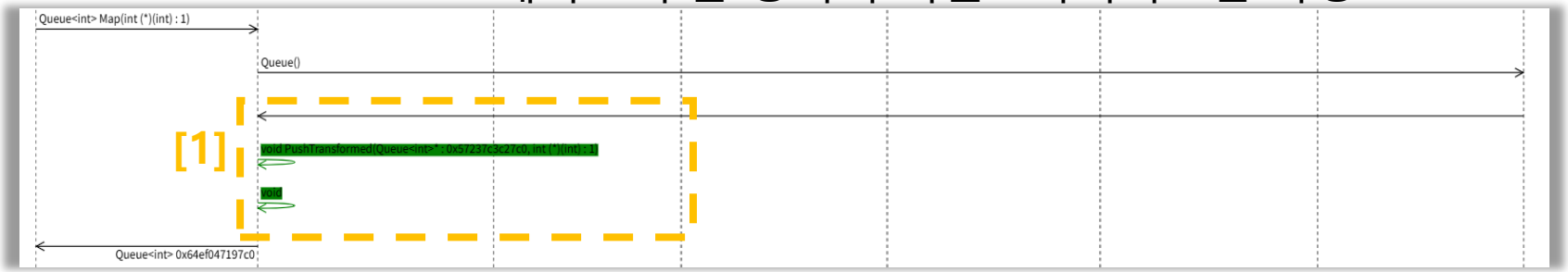


그림 4. 코드 수정 후 테스트를 실행한 시퀀스다이어그램

Catagorization 1. 리펙토링 - 메서드 추출 (Extract Method)

1-1 : 메서드 추출 방식의 시퀀스다이어그램 특징 1



1-2 : 메서드 추출 방식의 시퀀스다이어그램 특징 2

시퀀스다이어그램 특징

1. 초록색 – 새로운 함수 호출
2. 빨간색 부분과 초록색부분이 같이 있다면 거의 동일한 형태의 호출 시퀀스가 보임.
3. 빨간색과 초록색 부분 이외에서 동일한 호출 시퀀스를 보이는 부분이 있음.

Catagorization

2. 상속관계의 새로운 클래스 추가

```
Class QueueTestSmpl3 : public testing::Test {
    Queue<int> q0_;
+   Queue<int>* q1_ = new QueueNormalized<int>();
    Queue<int> q2_;
};
```

```
+ template <typename E>
+ class QueueNormalized : public Queue<E> {
+   public:
+     explicit QueueNormalized()
+         : Queue<E>() {
+     }
+     void Enqueue(const E& element) override {
+       Queue<E>::Enqueue(element / 10);
+     }
+};
```

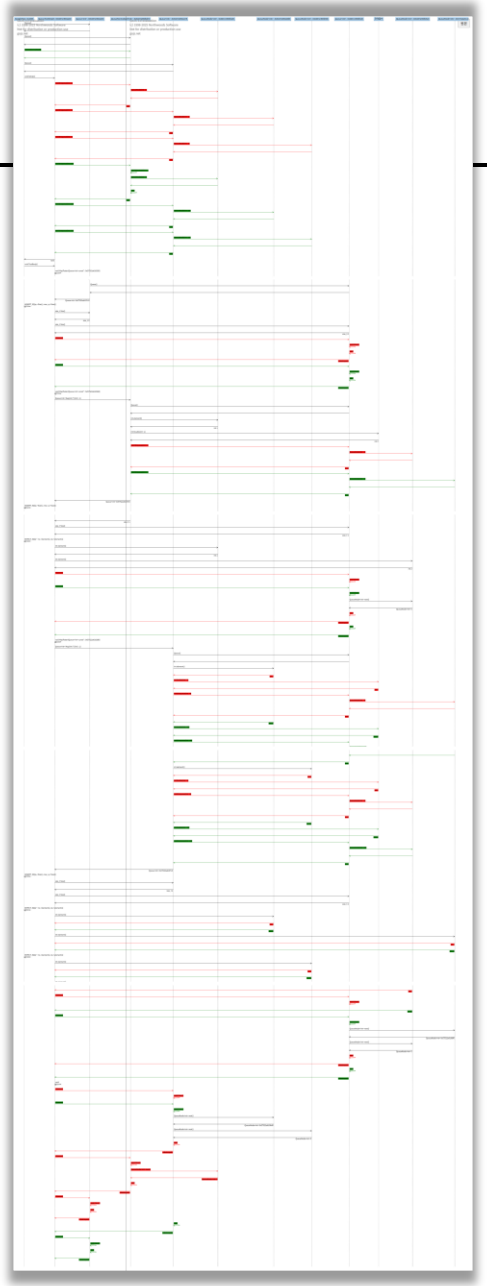
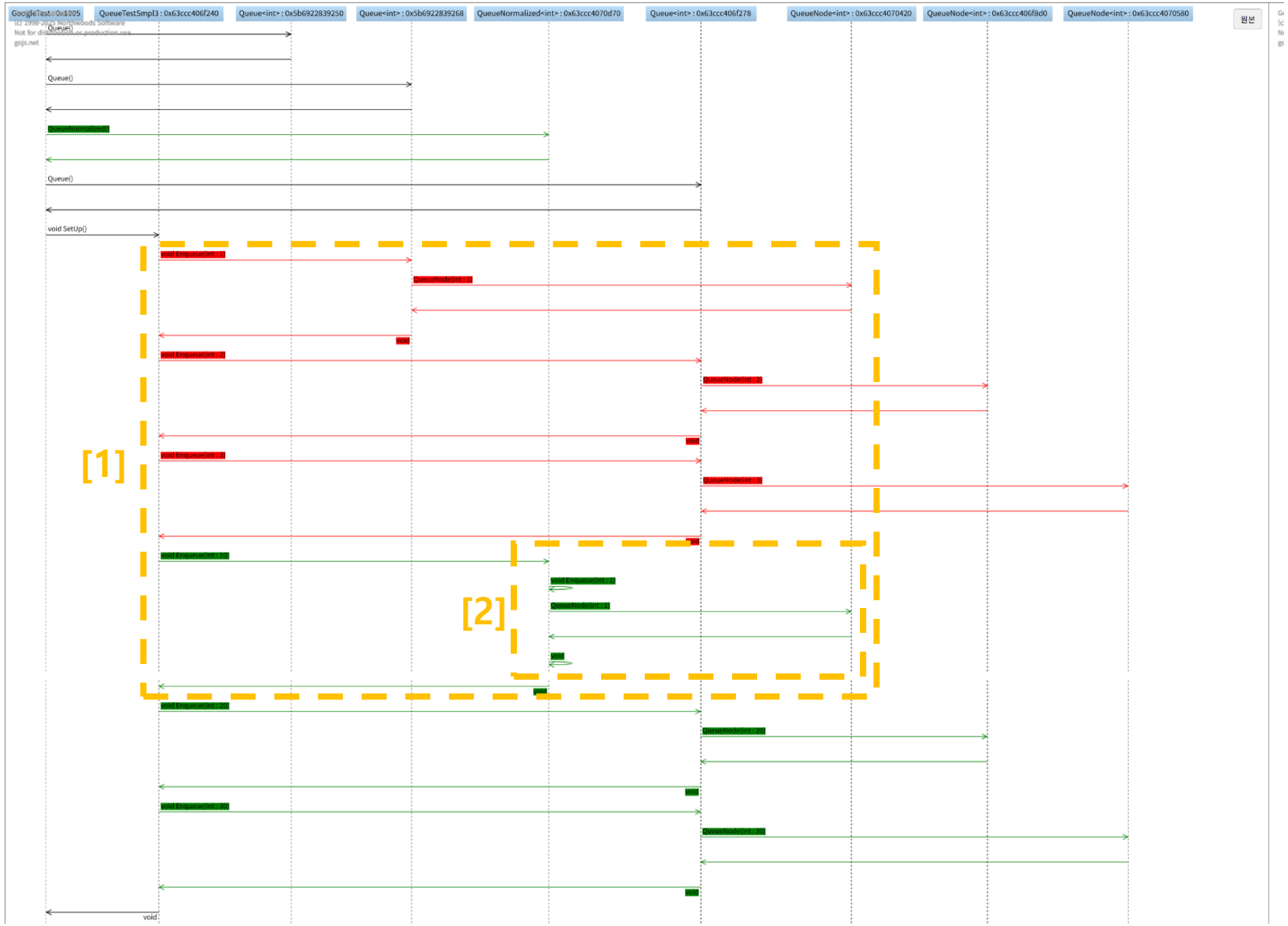


그림 5. 코드 수정 후 테스트를 실행한 시퀀스다이어그램

Catagorization

시퀀스 다이어그램의 다형성 표현



시퀀스다이어그램 특징

1. 동일한 함수 호출인데 다른 클래스의 객체와 상호작용함.
2. 부모 클래스의 멤버함수가 호출되었을때 자식 객체에서 호출됨.

Catagorization

시퀀스다이어그램에 잘 안보이는 요소
변수값의 변동

```
class OnTheFlyPrimeTable : public PrimeTable {
 public:
  bool IsPrime(int n) const override {
    if (n <= 1) return false;
+     n = std::abs(n);
    for (int i = 2; i * i <= n; i++) {
      if ((n % i) == 0) return false;
    }
    return true;
  }

  int GetNextPrime(int p) const override {
    if (p < 0) return -1;
+     p += 2;
    for (int n = p + 1;; n++) {
      if (IsPrime(n)) return n;
    }
  }
};
```

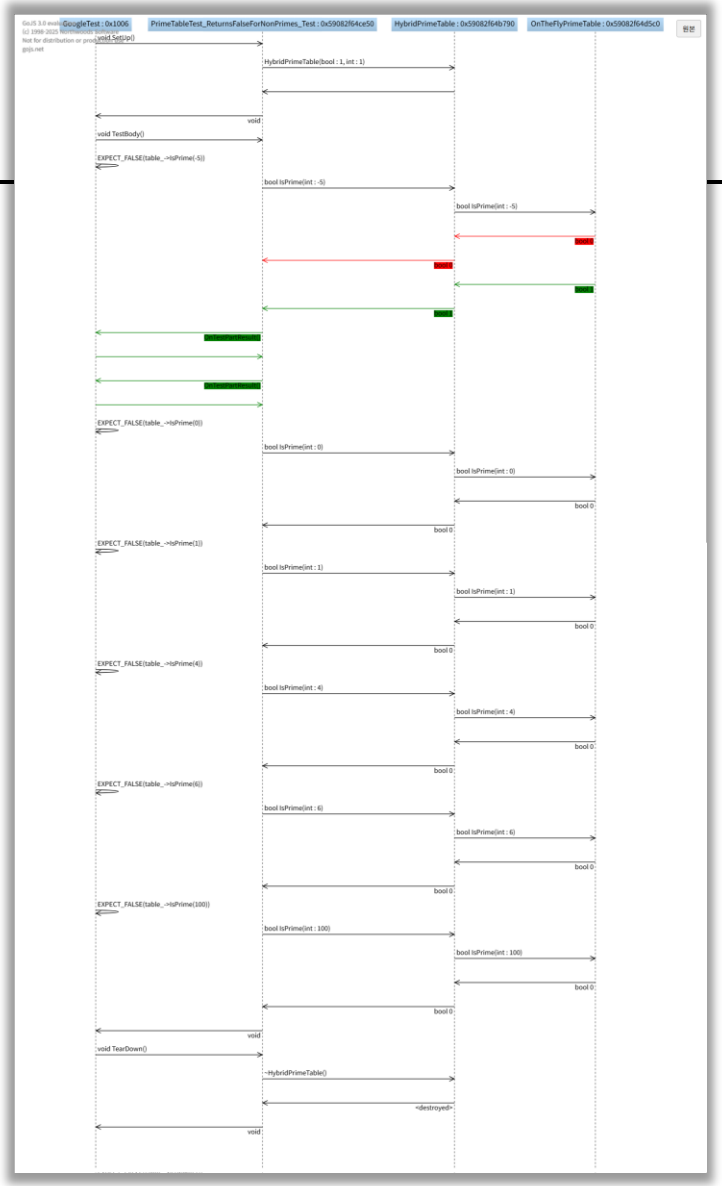
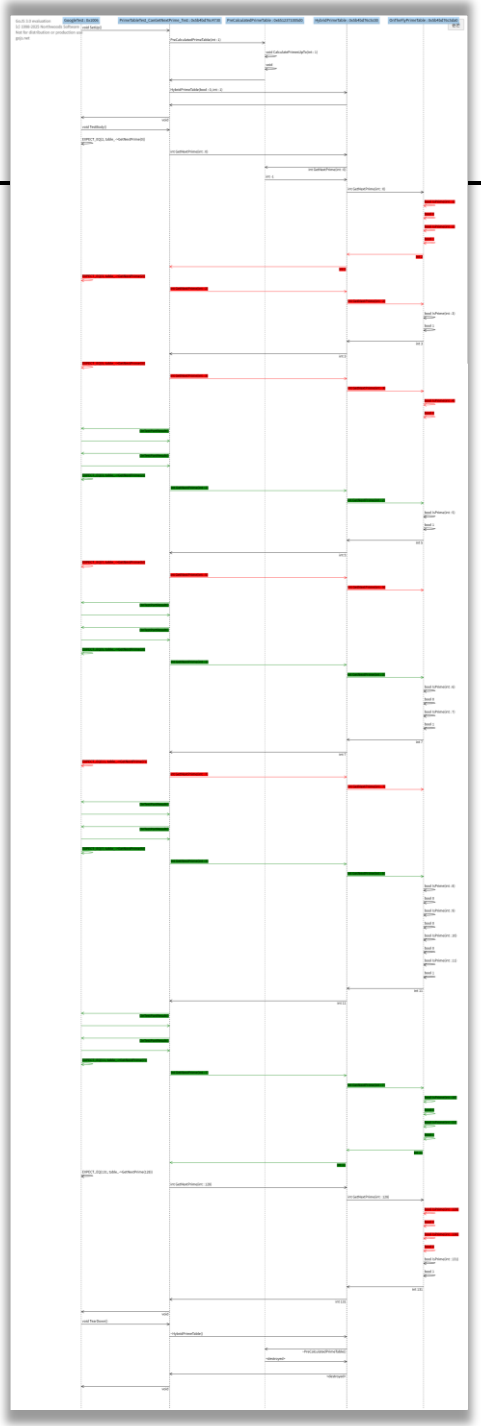
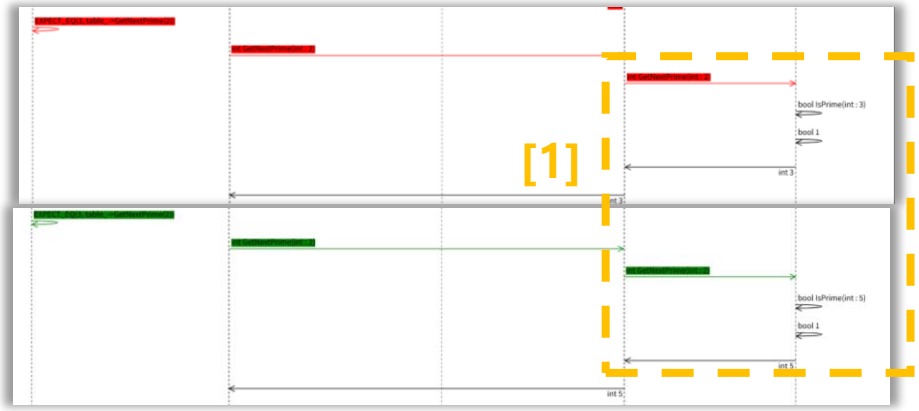


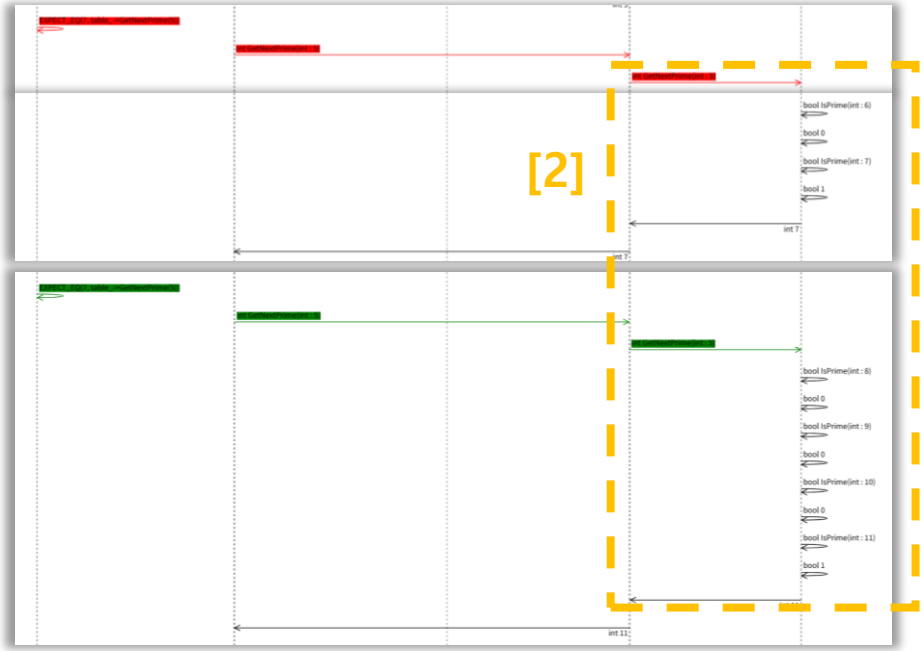
그림 6. 코드 수정 후 테스트를 실행한 시퀀스다이어그램



Catagorization



3-1



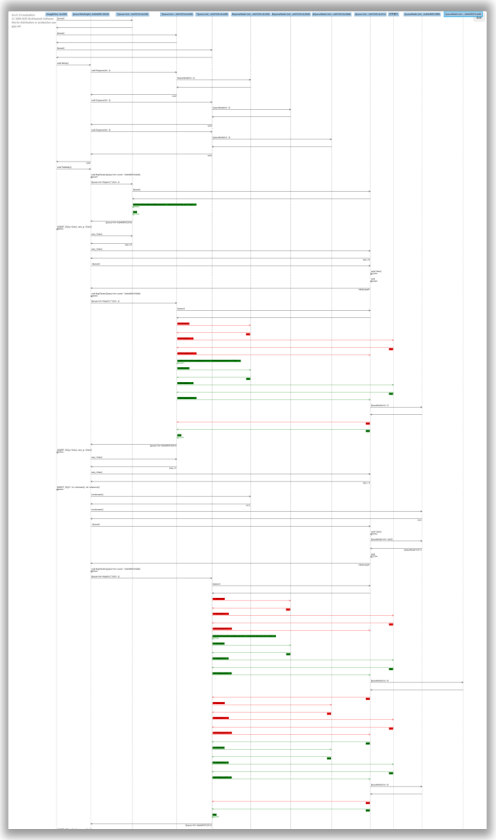
3-2

시퀀스다이어그램 특징

1. 기존과 같은 호출인데 인자값과 반환값이 변함
2. 내부 호출 개수가 변함
3. 테스트가 실패함

Catagorization

시퀀스다이어그램의 수정부분 역추적



코드 수정 원인

코드 수정결과