System Test Report

TraceRecoviz

버전: 1.7 | 작성일: 2025-10-31 |

1. Master Test Plan (MTP)

1.1 Introduction

1.1.1 Purpose

TraceRecoviz는 C++ 기반 GoogleTest 단위 테스트의 함수 호출, 반환, Assertion 이벤트를 추적하여 시퀀스 다이어그램으로 시각화하는 도구이다. 본 문서는 STP문서에 따라 System Test 수행 결과를 정리한다.

1.1.2 Scope

본 보고서는 TraceRecoviz의 System Test 수행 결과를 포함한다.

- Instrumentation: Clang LibTooling 기반 코드 계측 삽입 기능.
- TraceListener: GoogleTest 이벤트 훅을 이용한 테스트 생명주기 로그 기록.
- Logger: 로그 파일 생성, 회전, 닫기 타이밍 정합성 검증.
- Parser: 정규식 기반 로그 파싱 및 JSON 변환.
- Diagram Generator: 시퀀스 다이어그램 모델 생성 및 Assertion 이벤트 시각화 검증.

1.1.3 References

IEEE Std 829-2008 – Standard for Software and System Test Documentation

IEEE Std 830-1998 – Software Requirements Specification

TraceRecoviz SRS v1.4 (2025-10-18)

TraceRecoviz STP v1.4 (2025-10-18)

1.1.4 Definitions, Acronyms, and Abbreviations

용어	정의
TraceRecoviz	**C++/GoogleTest 기반 함수 호출·반환·Assertion 추적 및 시퀀스 다이어그램 생성 도구**
Instrumentation (INST)	**Clang LibTooling을 사용하여 테스트 소스에 자동으로 추적 코드를 삽입하는 과정**
TraceListener	**GoogleTest 의 EmptyTestEventListener를 상속하여 테스트 생명주기 이벤트를 로그로 기록하는 모듈**
CALL / RETURN / ASSERTION_CALL	**로그 라인의 행동 유형 식별자 (함수 호출, 반환, Assertion 발생)**
GoJS GraphLinksModel	**시퀀스 다이어그램 JSON 모델 형식 (nodeDataArray, linkDataArray)**
FR	**Functional Requirement (기능 요구사항)**
NFR	**Non-Functional Requirement (비기능 요구사항)**
LTP / LTC / LTR/ LR	**Level Test Plan / Case / Report / Log – IEEE 829에서 정의하는 시험 문서 유형**

1.2 Test Summary

1.2.1 Test Objectives

- 전체 흐름(Instrumentation → Listener → Logger → Parser → Diagram Generator)의 정상 동작 검증.
- SRS에서 정의된 비기능 요구사항(NFR-PERF, NFR-REL, NFR-REC) 충족 확인.
- 시스템 수준의 성능, 신뢰성, 복원성 지표 평가.

1.2.2 Test Items

구분	구성 요소	설명
INST (Instrumentation)	**inject_trace_tool.cpp, Clang LibTooling 기반**	**C++ 함수의 진입/종료/반환/Assertion 호출을 자동 삽입하는 도구. 모든 함수 호출에 대해 trace_enter/trace_return 코드가 주입되는지 검증한다.**
TraceListener	**trace_listener.cpp**	**GoogleTest 이벤트 훅(OnTestStart, OnTestEnd 등)을 통해 로그를 생성하고 파일 입출력을 관리한다.**
Logger / FileUtil	**trace_logger.cpp, file_util.cpp**	**로그 파일 생성, 회전 정책, 파일 닫기 시점의 정합성을 시험한다.**
Parser	**trace_parser.py**	**로그를 정규식 기반으로 파싱하여 JSON 구조(GraphLinksModel)로 변환하는 기능을 시험한다.**
Diagram Generator	**diagram_gen.py**	**파서 출력(JSON)을 GoJS 다이어그램 모델로 시각화 가능한지 확인한다.**

1.2.3 Test Features

요구사항 ID	시험 항목	설명
FR-INST-001~005	함수 진입/종료 계측	모든 함수에 trace_enter, trace_return 이 정확히 삽입되는지, return 문이 없는 함수(생성자/소멸자 포함)에서 누락되지 않는지 시험한다.
FR-LOG-001~003	로그 파일 생성 및 포맷	테스트 케이스 단위로 로그 파일이 생성되고, [TRACE] 접두어와 표준 로그 포맷이 준수되는지 검증한다.
FR-FMT-001~003	로그 라인 정규식 일치성	파서 정규식(^₩[(?P<testname>.+?)₩] ...)과 완벽히 일치하는 로그가 생성되는지 확인한다.
FR-AST-001~002	Assertion 로깅	EXPECT_*, ASSERT_* 호출 시 대응되는 [ASSERTION_CALL] 로그가 기록되는지 시험한다.
FR-PAR-001~005	파서 기능	로그에서 caller/callee, 반환값, assertion 정보를 정확히 추출하고 GoJS JSON 구조를 생성하는지 시험한다.
FR-SDR-001~007	시퀀스 다이어그램 모델링	CALL-RETURN 대응하는지, 미종결 호출 처리, Assertion 시각화, 객체별 고유 ID 매핑을 검증한다.

1.2.4 Environment

항목	내용
OS	WSL2 on Window 11
CPU	Intel(R) Core(TM) Ultra 7 155H(3.80 GHz)
Memory	32.0GB(31.6GB 사용 가능)
Compiler	clang++-18

2. Test Report – System Test

2.1 Introduction

본 장은 TraceRecoviz 시스템의 System Test 수행 결과를 기술한다.

3.2 FR-INST - Instrumentation Verification

FR-INST-001 - 함수 진입부 계측 삽입 검증

시험 목적: 모든 함수 진입 시점에 trace_enter() 호출이 자동으로 삽입되는지 확인한다

시험 입력: 입력 소스: 300개의 함수가 포함된 Google Test 샘플 코드

시험 절차: 소스 코드를 LibTooling AST 기반으로 파싱한다. 각 FunctionDecl 노드에 진입 훅(trace_enter) 삽입. 계측 후 산출된 코드 내 trace_enter 호출 존재 여부를 정적 분석.

기대 결과: 모든 함수 진입점에서 trace_enter 호출이 존재해야 한다.

실제 결과: 전체 300개 중 300개 함수 모두 삽입 성공. 삽입된 trace 호출이 기존 함수 구조에 영향을 주지 않음.

결과: Pass

FR-INST-002 – 함수 반환부 계측 삽입 검증

시험 목적: 모든 함수 반환 경로에 trace_return() 호출이 삽입되는지 확인한다.

시험 입력: 300개의 함수가 포함된 Google Test 샘플 코드

시험 절차: ReturnStmt 노드를 AST 탐색하여 모든 반환 경로에 trace_return()을 추가하고, 여러 분기 return을 가진 함수의 모든 경로를 검사한다.

기대 결과: 모든 return 문에 trace_return()이 삽입되어야 한다.

실제 결과: 273개 함수에서 정상 삽입, 27개 함수는 명시적 return이 없어 FR-INST-

003에서 검증.

결과: PASS

FR-INST-003 – 명시적 return 없는 함수 계측 검증

시험 목적: 생성자, 소멸자, void 함수 등 명시적 return이 없는 함수 말미에 trace_return()이 자동 삽입되는지 확인한다.

시험 입력: FR-INST-002에서 확인 안된 샘플 코드.

시험 절차: AST에서 ReturnStmt가 없는 FunctionDecl 탐색 후 함수 블록 마지막에 trace_return() 삽입 여부 확인.

기대 결과: 명시적 return이 없는 함수에서도 trace_return() 자동 삽입.

실제 결과: 전체 27개 중 27개 함수 모두 삽입 성공.

결과: PASS

3.3 FR-LOG — Logging Functionality

FR-LOG-001 – 로그 파일 생성 검증

시험 목적: 테스트 케이스별 로그 파일이 자동으로 생성되는지 검증한다.

시험 입력: Google Test 10개 케이스를 포함한 프로젝트 실행.

시험 절차: 테스트 실행 후 build/log/ 디렉터리 내 로그 파일 존재 여부와 파일명 중복 여부를 확인한다.

기대 결과: 테스트별 독립 로그 파일 생성, 파일명 충돌 없음.

실제 결과: 10개 케이스 모두 개별 로그 파일 정상 생성.

결과: PASS

FR-LOG-002 – 로그 이벤트 시작/종료 검증

시험 목적: 각 로그에 [TRACE_START], [TRACE_END] 이벤트가 존재하는지 검증한다.

시험 입력: 10개의 테스트 실행 후 생성된 로그 파일.

시험 절차: 각 파일의 첫 줄과 마지막 줄에서 이벤트 키워드 확인.

기대 결과: 모든 로그에 시작/종료 이벤트 존재.

실제 결과: 모든 로그에서 이벤트 확인됨.

결과: PASS

FR-LOG-003 – 로그 표준 포맷 검증

시험 목적: 모든 로그 라인이 [TRACE] 접두어로 시작하는지 검증한다.

시험 입력: build/log/ 경로의 로그 파일 10개.

시험 절차: 정규식 ^₩[TRACE₩]로 전체 라인을 검사.

기대 결과: [TRACE] 접두어 일치율 100%.

실제 결과: 전체 라인 일치.

결과: PASS

3.4 FR-FMT — Log Format Verification

FR-FMT-001 – 로그 라인 포맷 검증

시험 목적: 각 로그 라인이 파서 정규식(^₩[(?P<testname>.+?)₩])에 일치하는지 검증한다.

시험 입력: 총 10개의 로그 파일.

시험 절차: Python 정규식 매칭으로 전체 라인 검사.

기대 결과: 모든 로그 라인이 포맷과 일치.

실제 결과: 불일치 없음.

결과: PASS

FR-FMT-002 – 구분자/공백 일관성 검증

시험 목적: 로그 내 구분자(|, 공백, 탭) 일관성 유지 여부 검증.

시험 입력: 10개 로그 파일.

시험 절차: 파서로 각 라인의 구분자 패턴 검사.

기대 결과: 구분자 및 공백 형식이 모든 라인에서 동일해야 함.

실제 결과: 형식 일관성 유지.

결과: PASS

FR-FMT-003 – 인코딩 검증

시험 목적: 로그 파일이 UTF-8 인코딩 규격을 준수하는지 확인한다.

시험 입력: 로그 파일 10개.

시험 절차: 파일 인코딩 분석 도구로 BOM, 비ASCII 문자 검사.

기대 결과: BOM 없음, UTF-8 인코딩 유지.

실제 결과: 모든 파일 UTF-8 일치.

결과: PASS

3.5 FR-AST — Assertion Logging

FR-AST-001 – Assertion 호출 로그 기록 검증

시험 목적: EXPECT_*, ASSERT_* 호출이 [ASSERTION_CALL] 이벤트로 기록되는지 검증한다.

시험 입력: Assertion이 포함된 Google Test 샘플 코드 20개.

시험 절차: 각 테스트 실행 후 로그에서 [ASSERTION_CALL] 존재 여부 확인.

기대 결과: 모든 Assertion이 기록되어야 함.

실제 결과: 20개 모두 정상 기록.

결과: PASS

FR-AST-002 – Assertion 실패 이벤트 기록 검증

시험 목적: Assertion 실패 시 ASSERTION_FAIL 이벤트가 생성되는지 검증한다.

시험 입력: 실패 케이스 5건 포함한 테스트 실행.

시험 절차: 로그 파일에서 ASSERTION_FAIL 이벤트 존재 여부 확인.

기대 결과: 실패 케이스마다 ASSERTION_FAIL 생성.

실제 결과: 5건 모두 정상 생성.

결과: PASS

3.6 FR-PAR — Parser Verification

FR-PAR-001 – CALL 이벤트 파싱 검증

시험 목적: [CALL] 로그가 JSON 노드로 변환되는지 검증한다.

시험 입력: 472개의 CALL/RETURN 로그 포함 파일.

시험 절차: Parser 실행 후 JSON의 CALL 노드 수 확인.

기대 결과: CALL 이벤트 수 = JSON 노드 수.

실제 결과: 일치.

결과: PASS

FR-PAR-002 – RETURN 이벤트 파싱 검증

시험 목적: [RETURN] 로그가 JSON 링크로 정확히 매핑되는지 검증한다.

시험 절차: CALL-RETURN 매칭 검사 스크립트 실행.

기대 결과: 모든 CALL에 대응 RETURN 존재.

실제 결과: 472쌍 완전 매칭.

결과: PASS

FR-PAR-003 – Assertion 이벤트 파싱 검증

시험 목적: [ASSERTION_CALL] 로그가 JSON에 올바르게 반영되는지 검증한다.

시험 절차: 파서 출력 JSON에서 Assertion 노드 수 검증.

기대 결과: 로그 수와 동일해야 함.

실제 결과: 일치.

결과: PASS

FR-PAR-004 – JSON 구조 정합성 검증

시험 목적: JSON 키(caller, callee, return) 구조가 일관성 있게 유지되는지 확인한다.

시험 절차: JSON Schema 검사 수행.

기대 결과: 누락/중복 키 없음.

실제 결과: 정합성 유지.

결과: PASS

FR-PAR-005 – 노드 초기화 검증

시험 목적: 파서가 최초 등장한 노드에 대해 key, text, loc, size, isGroup, duration 필드를 초기화하는지 확인한다.

시험 입력: 다중 caller/callee를 포함한 로그 파일(1,200건).

시험 절차: Parser 실행 후 JSON(GraphLinksModel) 결과 생성. nodeDataArray 중 최초 등장 노드 필드 존재 여부 확인. 누락 필드 건수 집계.

기대 결과: 모든 신규 노드가 6개 필드를 정확히 초기화.

실제 결과: 모든 노드 정상. 누락 없음.

결과: PASS

3.7 FR-SDR — Sequence Diagram Recovery

FR-SDR-001 — 매개변수 타입/값 기록

시험 목적: 각 호출 이벤트마다 모든 매개변수의 런타임 타입 식별자와 값 표현을 기록(포인터/객체 혼합 시 매개변수 수만큼 (타입, 값/ID) 항목 존재).

시험 입력: 함수 인자 호출 25건.

시험 절차: CALL 라인의 ARGS 섹션에서 인자수·타입/값 쌍 확인.

기대 결과: 인자수 = 쌍 개수, 타입/값 정확.

실제 결과: 전부 일치.

결과: PASS

FR-SDR-002 — 반환 타입/값 기록

시험 목적: 완료된 호출에 대해 반환 타입 식별자와 값 표현을 기록. 생성자/소멸자는 <constructed>, <destroyed> 토큰 사용.

시험 입력: 반환/ctor/dtor 혼합 40건.

시험 절차: RETURN 라인에서 타입, 값 확인.

기대 결과: 반환 있는 함수는 (타입, 값/ID), ctor/dtor

실제 결과: 규정대로 기록.

결과: PASS

FR-SDR-003 — 객체 런타임 ID 부여

시험 목적: 실행 동안 각 객체 인스턴스에 고유·안정적 런타임 ID 부여·기록(동일 클래스라도 인스턴스별 ID 상이, 별도 라이프라인).

시험 입력: 동일 클래스 A/B 상호 호출 시나리오.

시험 절차: nodeDataArray의 object_id 일관성 및 Lifeline 분리 확인.

기대 결과: A/B 다른 ID, 라이프라인 2개 생성.

실제 결과: 요건 충족.

결과: PASS

FR-SDR-004 — CALL-RETURN 1:1 및 순서 재구성

시험 목적: 프로그램 가시 범위 모든 호출에 대해 정확히 1개의 CALL과 1개의 RETURN을 기록, 정렬 시 호출 전후 관계 보존.

시험 입력: 중첩 호출 20건 포함 로그.

시험 절차: CALL/RETURN 카운트·정렬 후 전후 관계 검사.

기대 결과: 20개의 CALL-m개의 RETURN, 순서 보존.

실제 결과: 100% 대응/보존.

결과: PASS

3.8 Conclusion

STP에 정의된 25개 세부 FR 모두 시험 수행 완료.
기능적 요구사항 대부분 충족, 모든 주요 기능 정상 작동.