객체지향개발방법론
Practice #1

202211287 김태인

202311252 곽수호

202111368 정선민

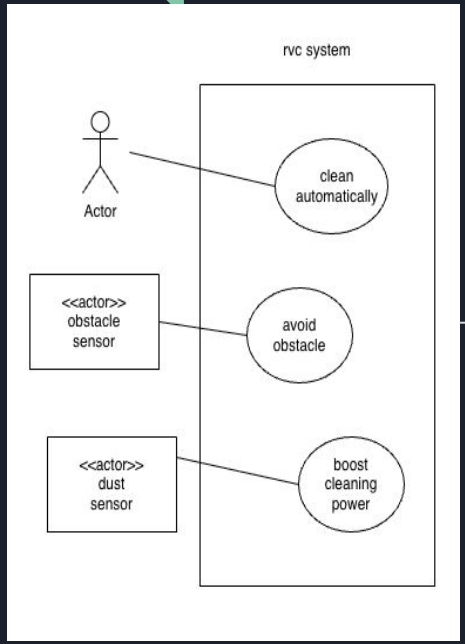
202211378 조성원

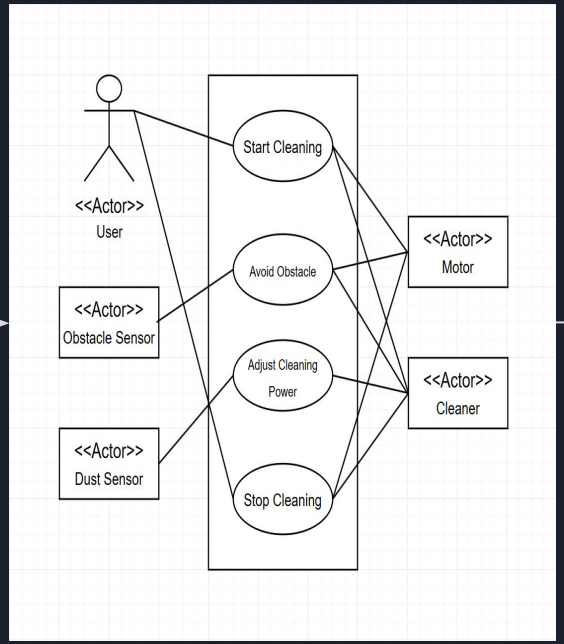
목차

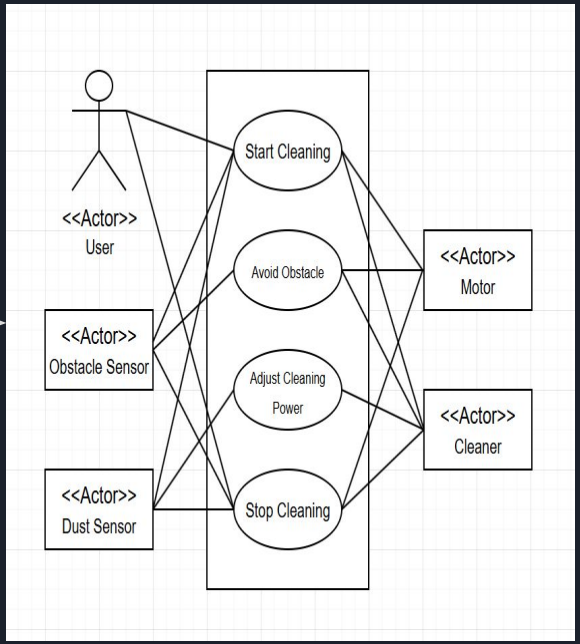
1. FR + Use case 분석
2. NFR
3. CI/CD

usecase diagram

수정후







Use case

1

Use Case : 1. Start Cleaning

Actors : User. Motor, Cleaner

Description :

- 사용자가 청소를 시작(ex. 시작버튼 누름, 모바일 구동)하면 Use case가 시작됨

- 시스템에서 일반 청소기능을 활성화하고 로봇에게 전진을 명령함

-시스템은 청소 과정 동안 센서 입력(장애물, 먼지)를 지속적으로 모니터링함

Use case

2

Use Case : 2. Avoid Obstacle

Actors : Obstacle Sensor, Motor, Cleaner

Description :

- 청소 중 장애물 센서가 전방 장애물을 감지하면 이 유스케이스가 시작됨.

- 시스템은 로봇의 동작을 중지시킴

- 시스템은 장애물 위치에 따라 회피 기동(좌/우 회전)을 수행하며, 사방이 막힌 경우 후진 후 경로를 재설정함.

- 로봇이 장애물을 피해 전진 이동을 재개하면 유스케이스가 종료됨.

Use case

3

Use Case : 3. Adjust Cleaning Power

Actors : Dust Sensor, Cleaner

Description :

- 청소 중 먼지 센서가 먼지를 감지하면 이 유스케이스가 시작됨.

- 시스템은 사전에 정의된 시간 동안 흡입력을 높여 집중 청소(Boost)를 수행함.

- 설정된 시간이 경과하면 시스템은 흡입력을 다시 일반 수준으로 되돌림.

- 흡입력이 정상화되거나, 장애물 감지 등으로 인해 (2. Avoid Obstacle)이 시작될 경우, 또는 사용자에 의해 청소가 중단되면 유스케이스가 종료됨.

Use case

4

Use Case : 4. Stop Cleaning

Actors : User, Motor, Cleaner

Description :

- 사용자가 청소를 중단(ex. 중단버튼 누름, 모바일 구동)하면 Use case가 시작됨

- 시스템에서 일반 청소기능을 비활성화하고 로봇에게 모터 중지를 명령

-시스템은 센서(장애물, 먼지)를 비활성화함

- 유스케이스가 종료됨.

FR

Ref.#	Functional Requirements	Use-Case Number & Name	Category
R1.1	Start Process	1. Start Cleaning	**Evident**
R1.2	Check Sensor Inputs	1. Start Cleaning	Hidden
R2.1	Go Straight Forward	1. Start Cleaning	Hidden
R3.1	Detect Obstacle	2. Avoid Obstacle	Hidden
R4.1.	Pause Motor	2. Avoid Obstacle	Hidden
R4.2.1.1	Turn Left	2. Avoid Obstacle	Hidden
R4.2.1.2	Turn Right	2. Avoid Obstacle	Hidden
R4.2.2	Move Backward	2. Avoid Obstacle	Hidden
R5.1	Check Obstacle Sensor Error	2. Avoid Obstacle	Hidden
R6.1	Power Up Cleaning	3. Boost Cleaning Power	Hidden
R7.1	Stop Process	4. Stop Cleaning	**Evident**

FR

R1.1 **Start Process**	전원이 커진 상태에서 전원을 사용자가 누르면 켜진다 전원이 켜지면 정지 상태로 obstacle 여부와 dust 정보를 감지한다.	R3.1 Detect Obstacle	전진 중 장애물이 전방에 있을 경우 cleaner를 끈다.
R2.1 Go Straight Forward	앞에 장애물이 없으면 전진하면서 청소를 한다.	R4.2.1.1 Turn Left	전진 중 장애물을 만났을 때 왼쪽에 장애물이 없으면 왼쪽으로 타이머가 끝날 때까지 회전한다,
R6.1 Power Up Cleaning	전진하면서 dust 센서가 true 면 cleaner가 3초 powerup이 되고, powerup 중 dust 센서가 true일 때마다 다시 3초씩 타이머 재카운트되고 시간이 다 되면 cleaner가 일반모드로 바뀐다.	R4.2.1.2 Turn Right	전진 중 장애물을 만났을 때 왼쪽에도 장애물이 있고 오른쪽에 장애물이 없을 때 오른쪽으로 타이머가 끝날때까지 회전한다.
R7.1 **Stop Process**	켜진 상태에서 사용자가 전원을 누르면 모든 기능이 꺼진다.	R4.2.2 Move Backward	전진 중 장애물이 전방, 양측면에 모두 있을 때 양옆 중 한쪽이라도 장애물이 없을 때까지 후진하다가 멈추고 없는 방향으로 회전한 이후 다시 전진하면서 청소를 시작한다.

FR

R1.2Check Sensor Inputs	시작하고 obstacle과 dust sensor 값이 정상적으로 잘 들어오는 확인한다. 오류가 없을 시 클리너와 모터를 동작시킨다.
R4.1Pause motor	전방에 장애물을 만났을 때 motor를 정지한다.
R5.1Check Obstacle Sensor Error	앞에 장애물을 만나고 왼쪽, 오른쪽에 장애물이 없어서 어느 한 방향으로 회전할 경우, 회전 이후 회전 방향 반대 센서의 입력을 확인해서 sensor값이 잘 인식하는지 검증한다. 오류가 없을 시 클리너와 모터를 동작시킨다.

NFR

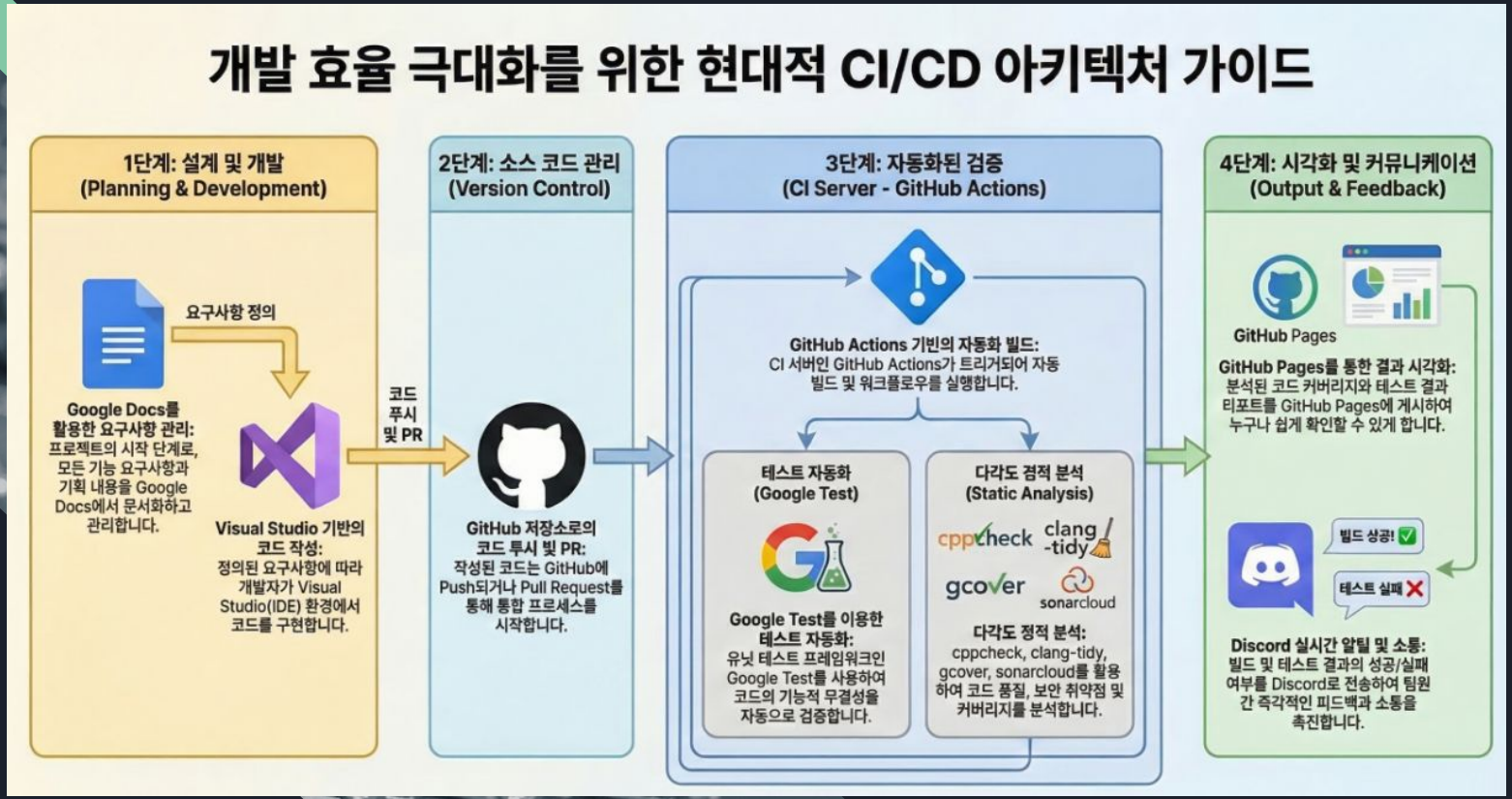
Product Requirements #	Description
PR1. 반응 시간	장애물 감지 후 회피 동작 시작까지의 지연은 100ms 이하로 한다.
PR2. 상태 갱신 주기	센서 입력 및 상태 전이는 10ms 단위로 갱신한다.

Organizational Requirements #	Description
OR1. 제어 범위	하드웨어 세부 제어는 고려하지 않는다.
OR2. 사용 제어	사용자는 전원을 키거나 끄는 동작만 수행한다. 나머지 기능들은 RVC가 자동으로 수행한다
OR3. 언어 제약	코드는 C++로 작성한다
OR4. CI 서버	CI 서버로 Github Action을 사용한다
OR5. 테스트 자동화	Google test로 테스트를 자동화 한다
OR6. 정적 코드 분석	정적 코드 분석 도구로 SonarCloud, Cppcheck, Clang-tidy를 사용한다

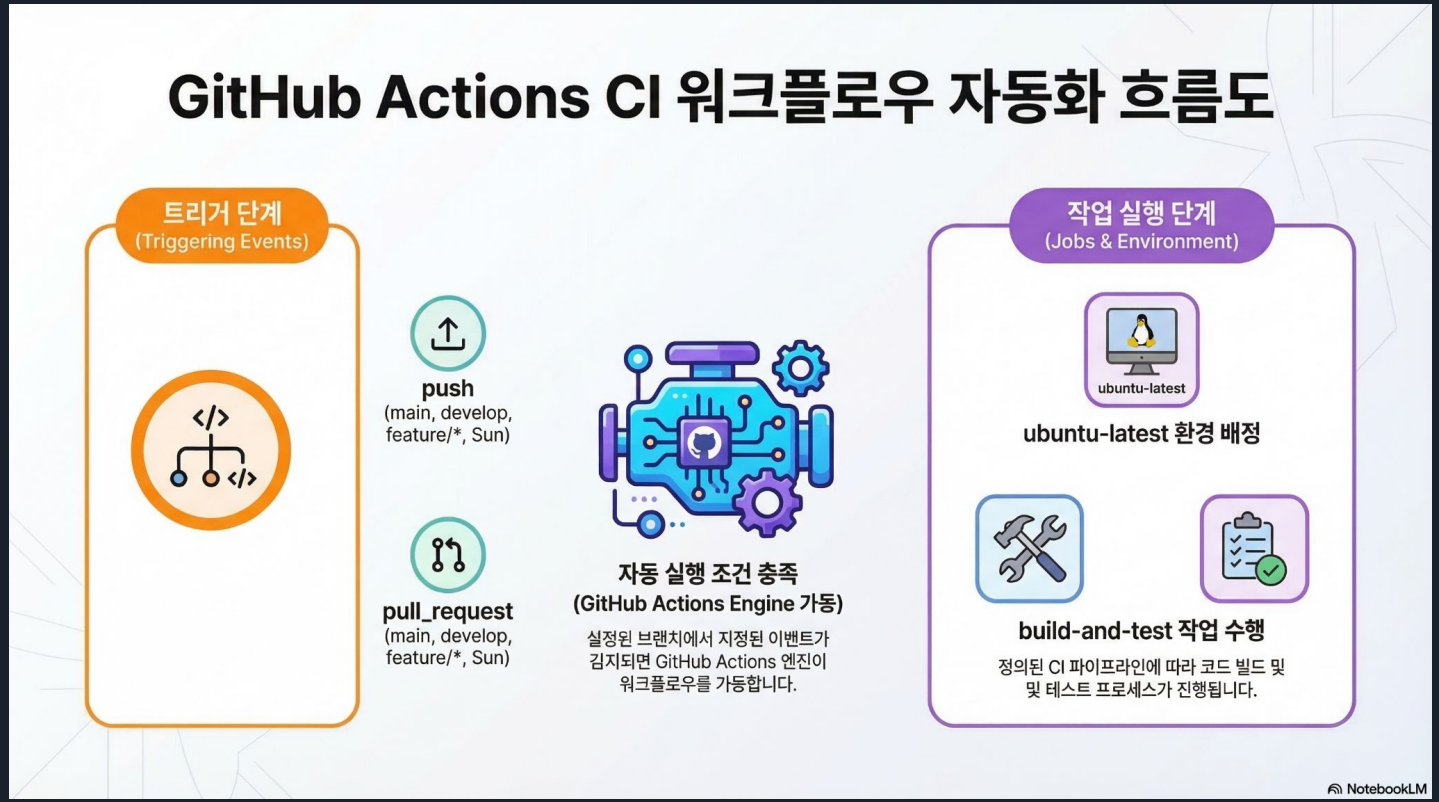
QA

Quality Attributes#	Description
QA1. 신뢰성 (Reliability)	모든 정상 입력 범위에서 상태 전이는 예측 가능해야 한다
QA2. 정확성 (Accuracy)	장애물 감지 및 먼지 감지 판정은 오차 ±5% 이내에서 동작해야 한다.
QA3. 안정성 (Stability)	1분 이상 지속 주행 시 센서 입력 누락이나 상태 오동작이 발생하지 않아야 한다.
QA4. 유지보수성 (Maintainability)	제어 알고리즘, 상태 전이 로직, 센서/모터 드라이버는 독립 모듈로 구성되어야 하며, 코드 수정 시 다른 모듈의 변경 없이 유지보수가 가능해야 한다.
QA5. 확장성 (Extensibility)	RVC는 센서 확장·변경이 가능하고, 특정 구역 반복 청소, 모바일 앱 연동, 머신러닝 기반 효율적 청소 기능 추가시 최소한의 코드 변경으로 확장 가능해야 한다
QA6. 가시성 (Visibility)	내부 상태(State), Sensor 값, Cleaner 모드 등은 디버깅 혹은 시뮬레이션 모드에서 실시간으로 출력되어야 한다.
QA7. 테스트 용이성 (Testability)	실제 하드웨어 없이도 시뮬레이션 입력을 통해 로직 검증이 가능해야 한다.
QA8. 안전성 (Safety)	모든 모터 제어 명령은 유효한 센서 입력을 전제로만 수행되며, 비정상 상태에서는 즉시 정지해야 한다.
QA9. 일관성 (Consistency)	동일 조건의 입력에 대해 항상 동일한 상태 전이 및 제어 결과가 나와야 한다.

CI/CD 환경


개발 효율 극대화를 위한 현대적 CI/CD 아키텍처 가이드
1단계: 설계 및 개발
(Planning & Development)
요구사항 정의
Google Docs를 활용한 요구사항 관리:
프로젝트의 시작 단계로, 모든 기능 요구사항과 기획 내용을 Google Docs에서 문서화하고 관리합니다.
Visual Studio 기반의 코드 작성:
정의된 요구사항에 따라 개발자가 Visual Studio(IDE) 환경에서 코드를 구현합니다.
코드 푸시 및 PR
2단계: 소스 코드 관리
(Version Control)
GitHub 저장소로의 코드 푸시 및 PR:
작성된 코드는 GitHub에 Push되거나 Pull Request를 통해 통합 프로세스를 시작합니다.
3단계: 자동화된 검증
(CI Server - GitHub Actions)
GitHub Actions 기반의 자동화 빌드:
CI 서버인 GitHub Actions가 트리거되어 자동 빌드 및 워크플로우를 실행합니다.
테스트 자동화
(Google Test)
Google Test를 이용한 테스트 자동화:
유닛 테스트 프레임워크인 Google Test를 사용하여 코드의 기능적 무결성을 자동으로 검증합니다.
다각도 검적 분석
(Static Analysis)
cppcheck
clang-tidy
gcover
sonarcloud
다각도 정적 분석:
cppcheck, clang-tidy, gcover, sonarcloud를 활용하여 코드 품질, 보안 취약점 및 커버리지를 분석합니다.
4단계: 시각화 및 커뮤니케이션
(Output & Feedback)
GitHub Pages
GitHub Pages를 통한 결과 시각화:
분석된 코드 커버리지와 테스트 결과 리포트를 GitHub Pages에 게시하여 누구나 쉽게 확인할 수 있게 합니다.
빌드 성공!
테스트 실패
Discord 실시간 알림 및 소통:
빌드 및 테스트 결과의 성공/실패 여부를 Discord로 전송하여 팀원 간 즉각적인 피드백과 소통을 촉진합니다.

CI - Github Action



yml 스크립트

yml : 데이터를 표현하면서 사람이 읽기 쉽도록 구성되어 있는 마크업언어.

CI/CD환경에서 파이프라인을 구성할 때 워크플로우, 즉 작업단계와 대상들을 정의해 놓는데 사용.

추가로 애플리케이션 배포할 때 추가 구성요소들의 버전제어, 리소스 제한 등을 한 파일에 모아 설정해서 템플릿으로 사용.

```yaml
name: CI

on:
  push:
    branches: [ "main", "develop", "feature/*", "Sun", "bug_fix" ]
  pull_request:

jobs:
  build-and-test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v4
        with:
          fetch-depth: 0 #소나클라우드 분석위한 깃 히스토리 가져오기

      - name: Install build & analysis tools
        run: |
          sudo apt-get update
          sudo apt-get install -y build-essential
          sudo apt-get install -y cmake g++ clang-tidy cppcheck make lcov python3-pygments gcovr

      - name: Configure
        run: |
          cmake -S . -B build \
          -DENABLE_TESTS=ON \
          -DCMAKE_BUILD_TYPE=Debug \
          -DCMAKE_CXX_FLAGS="--coverage" \
          -DCMAKE_EXE_LINKER_FLAGS="--coverage" \
          -DCMAKE_C_COMPILER=/usr/bin/gcc \
          -DCMAKE_CXX_COMPILER=/usr/bin/g++ \
          -DCMAKE_C_COMPILER_LAUNCHER= \
          -DCMAKE_CXX_COMPILER_LAUNCHER= \
          -DCMAKE_EXPORT_COMPILE_COMMANDS=ON

```

워크플로우 순서

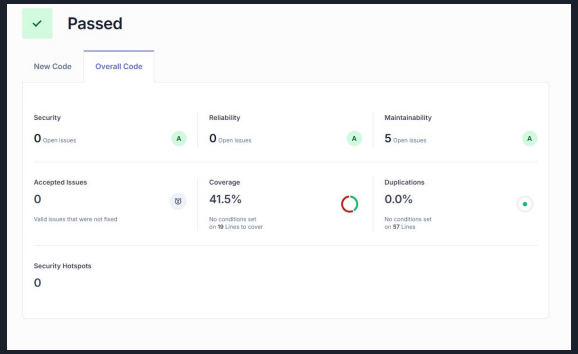
1. 빌드 - 컴파일이 되는지 확인

2. 동적 분석 테스트, 커버리지 확인 - 코드가 의도대로 동작하는가? (ctest 동적분석)
   & 작성한 테스트 코드로 검사가 안된 부분이 있는가? (커버리지, lcov, gcov)

3. 정적분석 - 코드에서 텍스트 그 자체를 테스트함. 논리적 버그나 쓸데없거나 쓰이지 않는 코드가 있는지 확인함.

4. 소나클라우드 - 앞선 테스트들의 결과 자료들을 모아서 이 코드를 사용해도 되는지 판단함.

SonarCloud



- 소스 코드를 분석하여 버그, 취약점, 코드 스멜을 탐지하고 시각화하는 SaaS 서비스.
- GitHub와 연동하여 개발자가 코드를 GitHub에 Push하면 자동으로 분석을 수행하여, 코드 상태를 점검하는 역할을 함.
- 본 프로젝트에서 코드 스멜 탐지용 Cppcheck (정적 분석: 안정성)Clang-Tidy (정적 분석: 품질 & 스타일)과 동적분석 커버리지 검사용 gcov (동적 분석: 테스트 완결성)를 사용함

+ 분석 도구들의 결과를 받아 시각화함

Static Analysis

Cppcheck:

- 실제 배경 환경을 반영하면서 프로젝트 핵심 코드만 분석
- Warning: 잠재적 오류나 위험 패턴 감지
- Style : 코드 품질 저하 요소, 코드 스멜 탐지
- Performance: 성능 비효율 가능성 탐지
- Portability: 플랫폼이 달라졌을 때 문제되는 코드 탐지

+ SonarCloud 통합 및 리포트 : 분석 결과를 SonarCloud로 전송하여 대시보드에서 코드 스멜과 취약점을 시각화하고 이력 관리

Intentionality | Not clear

unused variable 'unused'

Unused local variables should be removed cpp:S1481

Software qualities impacted: Maintainability Low

Open · Not assigned · Code Smell · Minor

Tags
unused +

Line affected
L24

Effort
5 min

Introduced
59 minutes ago

Where is the issue? | Why is this an issue? | How can I fix it? | Activity

Open in IDE

src/main.cpp

See all issues in this file

```
1  #include <iostream>
2  #include "rvc_controller.h"
3
4  class TestScanner {
5  public:
6      void Check_Status() { // snake_case + non-const -> Clang-Tidy°¡ ¾È¾îÇÔ
7          // 3. modernize-use-nullptr: NULL 'ëº¸Ã nullptr »ç¿ë ±ÇÀå
8          int* ptr = NULL;
9
10         // 4. readability-identifier-naming: VariableCase (camelCase¿¡¼ß ÇÔ)
11         int My_Local_Variable = 10;
12
13         if (ptr == NULL) {
14             std::cout << "Scanning..." << My_Local_Variable << std::endl;
15         }
16     }
17 };
18
19 int main() {
20     TestScanner scanner;
21     scanner.Check_Status();
22     RvcController c;
23     std::cout << "RVC Controller: " << c.next_action(false,false,false,false) << "\n";
24     int unused = 1; // cppcheck: unusedVariable - 'unused' is assigned a value that is never used.
       unused variable 'unused'
25     return 0;
26 }
```

Static Analysis

Clang-tidy:

- 컴파일러 수준의 정밀한 정적 분석을 통해 코드 표준화 및 자동 리팩토링 수행
- Bugprone : 논리적 오류 방지
- Modernize : 최신 표준 적용
- Readability : 명명 규칙 및 가독성 등 여러 설정들 존재

+ SonarCloud 통합 및 리포트 : 분석 결과를 SonarCloud로 전송하여 대시보드에서 코드 스멜과 취약점을 시각화하고 이력 관리

Consistency | Not conventional

Use the "nullptr" literal.

"nullptr" should be used to denote the null pointer cpp:S4962

Software qualities impacted: Maintainability High

Open · Not assigned · Code Smell · Critical

Tags
bad-practice ... +

Line affected
L8

Effort
1 min

Introduced
1 hour ago

Where is the issue? | Why is this an issue? | Activity | More info

Open in IDE

src/main.cpp

See all issues in this file

```
1  #include <iostream>
2  #include "rvc_controller.h"
3
4  class TestScanner {
5  public:
6      void Check_Status() { // snake_case + non-const -> Clang-Tidy*¡ %È¾ÍÇÔ
7          // 3. modernize-use-nullptr: NULL 'Ë¥Á nullptr »ç¿ë ±ÇÀå
8          int* ptr = NULL;
9
10         // 4. readability-identifier-naming: VariableCase (camelCase¿¡¼ ÇÔ)
11         int My_Local_Variable = 10;
12
13         if (ptr == NULL) {
14             std::cout << "Scanning..." << My_Local_Variable << std::endl;
15         }
16     }
17 };
18
19 int main() {
20     TestScanner scanner;
21     scanner.Check_Status();
22     RvcController c;
23     std::cout << "RVC Controller: " << c.next_action(false,false,false,false) << "\n";
24     int unused = 1; // cppcheck: unusedVariable - 'unused' is assigned a value that is never used.
25     return 0;
26 }
```

Use the "nullptr" literal.

int* ptr = NULL;

Use the "nullptr" literal.

Dynamic Analysis

Gcov:

- 실제 배경 환경을 반영하면서 프로젝트 핵심 코드만 분석
- Line Coverage : 각 코드 행이 실행되었는지 확인.
- Function Coverage : 어떤 함수가 호출되었는지 확인.
- Branch Coverage : if나 switch문에서 true 케이스와 false 케이스가 각각 실행되었는지 확인.

+ GitHub Pages & SonarCloud 시각화 : lcov를 통해 생성된 HTML 리포트를 GitHub Pages에 게시하거나, gcovr를 통해 XML로 변환 후 SonarCloud 대시보드에서 소스 코드별 합격/불합격 구간을 시각화.

OOAD_TeamProj > src > rvc_controller.cpp

Coverage 94.4%

New code: last 90 days

```
1  …  #include "rvc_controller.h"
2  …
3  …  const char* RvcController::next_action(bool front_obstacle,
4  …                                         bool left_obstacle,
5  …                                         bool right_obstacle,
6  …                                         bool dust_detected) const
7  …  {
8  …
9  …      if (dust_detected) {
10 …          return "BOOST_CLEANING";
11 …      }
12 …
13 …      if (front_obstacle && left_obstacle && right_obstacle) {
14 …          return "REVERSE_THEN_TURN";
15 …      }
16 …
17 …      if (front_obstacle) {
18 …          return "STOP_AND_AVOID_TURN";
19 …      }
20 …      return "GO_STRAIGHT";
21 …  }
```

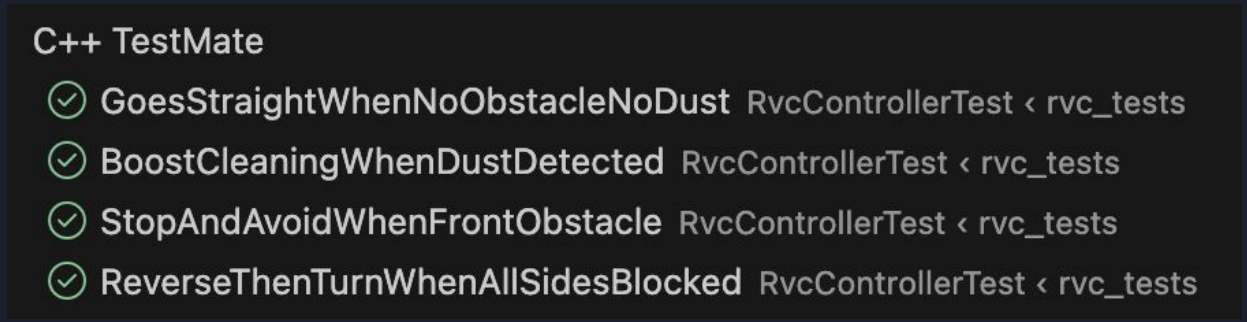
Test automation - Google test

- GoogleTest를 외부에서 가져와서 테스트 실행 파일을 만듦
- 테스트 코드가 프로젝트 본체와 GTest를 쓰게 연결
- 테스트를 자동 인식해서 CTest에 등록

장점:

1. 테스트 자동화가 쉬움 (GitHub Actions에서 자동 실행)
2. FetchContent로 GTest를 알아서 가져오므로 팀원이 설치 X
3. 유지보수가 새 테스트를 추가해도 자동으로 등록해줘서 편함

Test automation - Google test



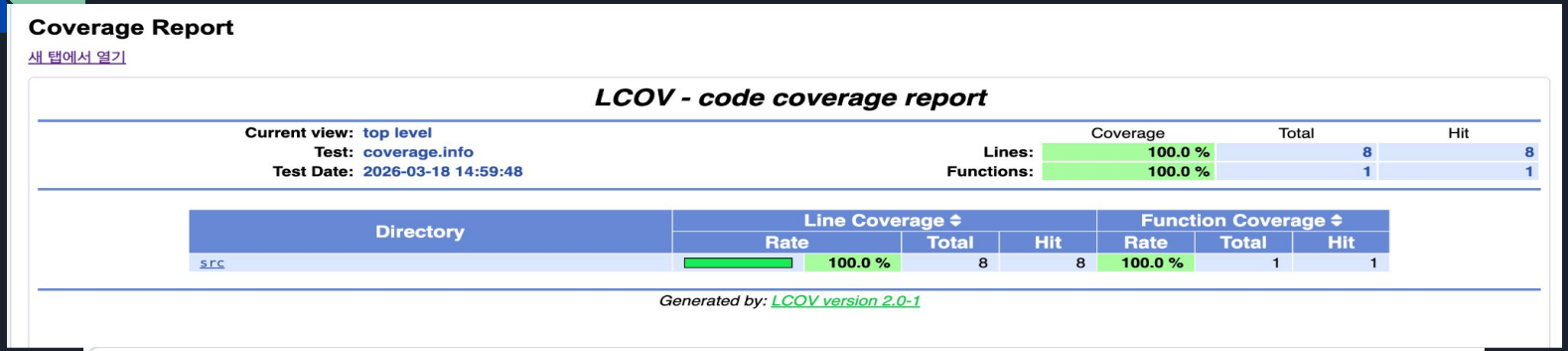
- VS Code의 C++ TestMate 확장을 통해 테스트를 더 쉽게 관리

디스코드와 연동하여 빠른 결과 공유

Discord Webhook 알림
GitHub Actions의 워크플로 실행 상태(성공, 실패 등)를 Discord 채널로 실시간 전송하는 자동화 도구

GitHub 사이트에 일일이 들어가지 않아도, 평소 협업 도구인 Discord에서 즉시 빌드 결과를 확인 가능

GitHub pages



Coverage Report

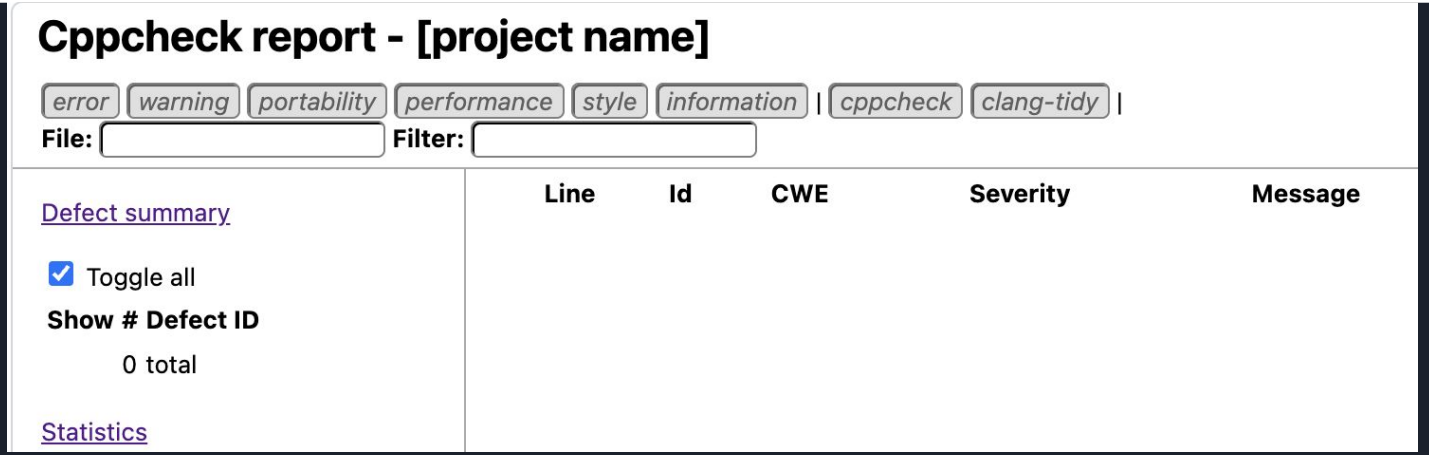
새 탭에서 열기

LCOV - code coverage report

			Coverage	Total	Hit
Current view:	top level	Lines:	100.0 %	8	8
Test:	coverage.info	Functions:	100.0 %	1	1
Test Date:	2026-03-18 14:59:48				

Directory	Line Coverage			Function Coverage		
	Rate	Total	Hit	Rate	Total	Hit
src	100.0 %	8	8	100.0 %	1	1

Generated by: LCOV version 2.0-1



Cppcheck report - [project name]

error | warning | portability | performance | style | information | cppcheck | clang-tidy |

File: **Filter:**

Defect summary

Toggle all

Show # Defect ID

0 total

Statistics

Line	Id	CWE	Severity	Message

감사합니다.