

프로그래밍 역량 강화 전문기관, 민코딩

SSD 프로젝트 기능 추가 요청



목차

1. 기능 추가 미션 소개
2. SSD - Erase 기능 추가
3. Logger 기능 추가
4. Runner 기능 추가
5. SSD - Write Buffer 기능 추가
6. 시나리오 추가시, 재빌드 이슈 해결
7. 기능 추가 미션, 유의사항

기능 추가 미션 소개

기능 추가, 리팩토링 미션 개요

CRA과정 프로젝트 진행 스케줄

시간	1일차	2일차	3일차	4일차	5일차		
8:30 ~ 9:30	Team Building & Communication	Team Project	프로젝트 안내 Team Project	Team Project	Team Project *발표준비		
9:30 ~ 10:30							
10:30 ~ 11:30	Team Project 개발환경 셋업 및 안내						Team Project 발표, 평가 준비
11:30 ~ 12:30							
12:30 ~ 13:30	점심 시간						
13:30 ~ 14:30	개발환경 안내 Team Project	Team Project	Team Project	Team Project	발표		
14:30 ~ 15:30							
15:30 ~ 16:30	Team Project						현업활동 Guide
16:30 ~ 17:30							

범례

프로젝트 진행 (프로젝트 코치 상주)
자율적 프로젝트 진행
교육

3 ~ 4일차 미션

미션

1. SSD - Erase 명령어 추가
2. Logger 기능 추가
3. Runner 기능 추가
4. SSD - Write Buffer 기능 추가
5. 시나리오 추가시, 재빌드 이슈 해결

디자인 패턴을 사용시, 추가 점수

디자인 패턴은 여러 곳에서 발생할 수 있는 문제를 해결하는 일반화된 모범 사례입니다. 이를 클린코드의 Best Practice 라고 합니다. 이로 인해, GoF 디자인패턴을 사용하면 약간의 추가 점수가 주어집니다.

예시)

- Singleton Pattern
- Command Pattern 등

SSD – Erase 기능 추가

Erase 기능이 지원되도록, 가상 SSD와 Shell 에 명령어를 추가한다.

SSD - Erase 기능 추가

Write 명령어 처럼 **SSD 자체에 Erase 기능을 추가**하고,
Shell에서 명령어를 수행할 수 있도록 한다.

- SSD 명령어 : E [LBA] [SIZE]
 - 특정 LBA 부터 특정 Size 까지 내용을 삭제한다.
 - 삭제할 수 있는 최대 Size는 최대 10칸이다.
 - SSD에서 삭제하면 0x00000000이 된다.
- Shell 명령어 1 : erase [LBA] [SIZE]
 - SSD에 명령어를 수행한다.
- Shell 명령어 2 : erase_range [Start LBA] [End LBA]
 - Start LBA 부터 END LBA 직전 까지 내용을 삭제한다.

Logger 기능 추가

Test Shell 이 출력하는 Log를 관리해주는 시스템 제작

Logger 가 필요한 이유

테스트를 진행하면서 Fail이 발생하면 Log Message를 불량 분석을 하게 된다.

검증팀의 여러 사람들이 각자 테스트 시나리오를 작성하다 보면,
통일되지 않은 로그 포맷을 사용하지 않아, 불량 분석에 시간이 걸리기도 하고
로그들을 Text파일로 저장하는데 로그 데이터가 대량으로 쌓이면서 파일 관리가 어렵다.

위와 같은 문제를
해결해주는 기능을 추가하자.

우리 팀만의 print 함수를 만들면 된다.
→ 화면 출력이 되면서 동시에 파일 Write가 되는 print 함수

Logger 기능 1 : 통일된 로그

로그 포맷

- 출력 포맷 : [날짜 시간] 함수명() : 로그메세지
- 함수명이 들어가는 공간에는 **30칸**을 확보하여 함수명을 기입한다.

로그 포맷 예시

[26.01.01 17:04] sendMessageWithData()	: send and wait sync message.
[26.01.01 17:04] checkMessage()	: FAIL - Response message didn't arrive.
[26.01.01 17:04] release()	: Bye bye.



logger.print("Bye Bye") 호출시

Logger 기능 2 : 로그파일 관리

실제로 대부분의 불량 분석은 Last log 10MB 로 해결된다.
하지만 하나의 파일이 TB 이상이 된 상태로 불량분석에 시간이 오래걸리게 된다.
우리의 프로젝트에서는 조금 더 사이즈를 축소하여 10MB가 아닌
10KB 단위로 파일을 관리해보자.



until_260307_17h_12m_1
1s.log

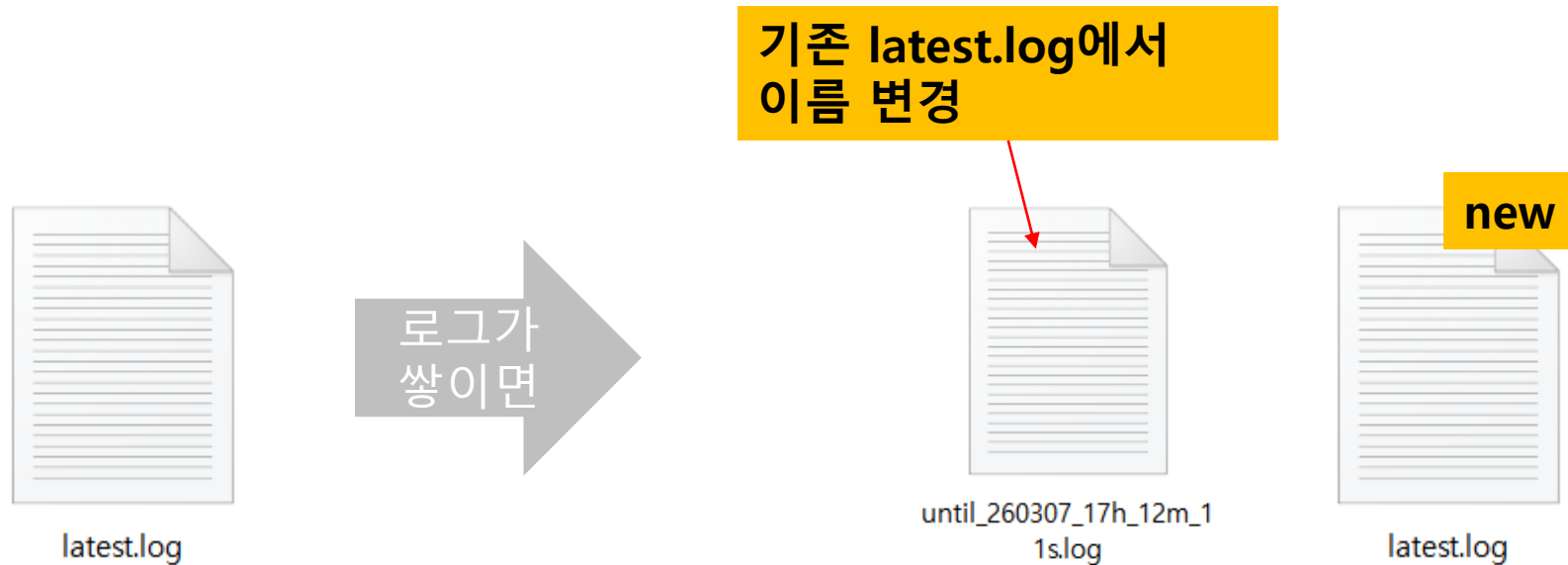


until_260307_17h_45m_3
2s.log

Logger 기능 2 : 로그 기록 규칙

로그 기록 규칙

1. 로그를 남기기 위해 latest.log 파일을 생성한다.
2. 10kb 까지 기록을 수행한다.
3. 10kb가 넘으면 해당 파일의 이름을 "until_날짜_시간.log" 으로 변경한다.



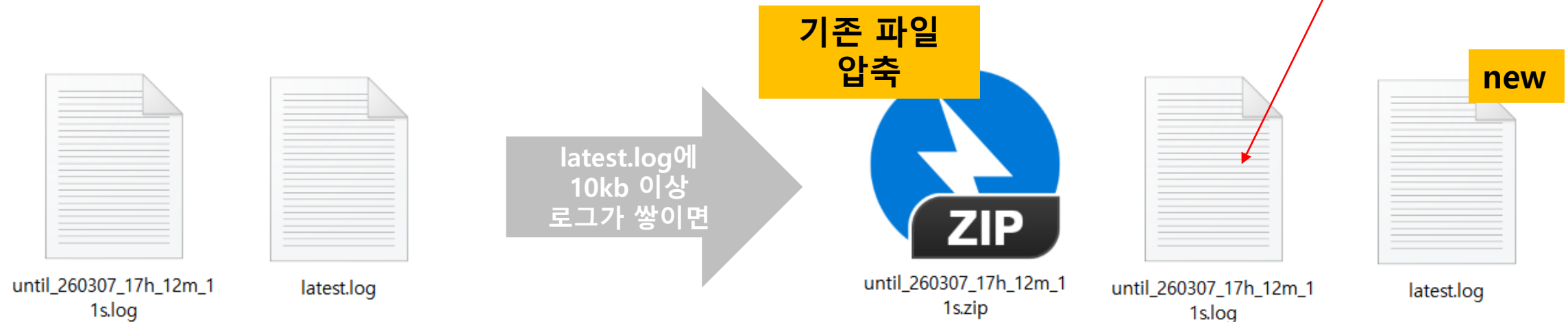
파일명 : 마지막 로그 기록의 날짜 + 시간

Logger 기능 2 : 로그 압축 규칙

로그 압축 규칙

1. 두 개의 until_날짜_시간.log 파일이 생성되면, 가장 이전에 생성된 log파일을 압축한다.
2. 압축 Library를 썼다고 가정하고, 파일명만 .log → .zip 으로 변경한다.

기존 latest.log에서
이름 변경

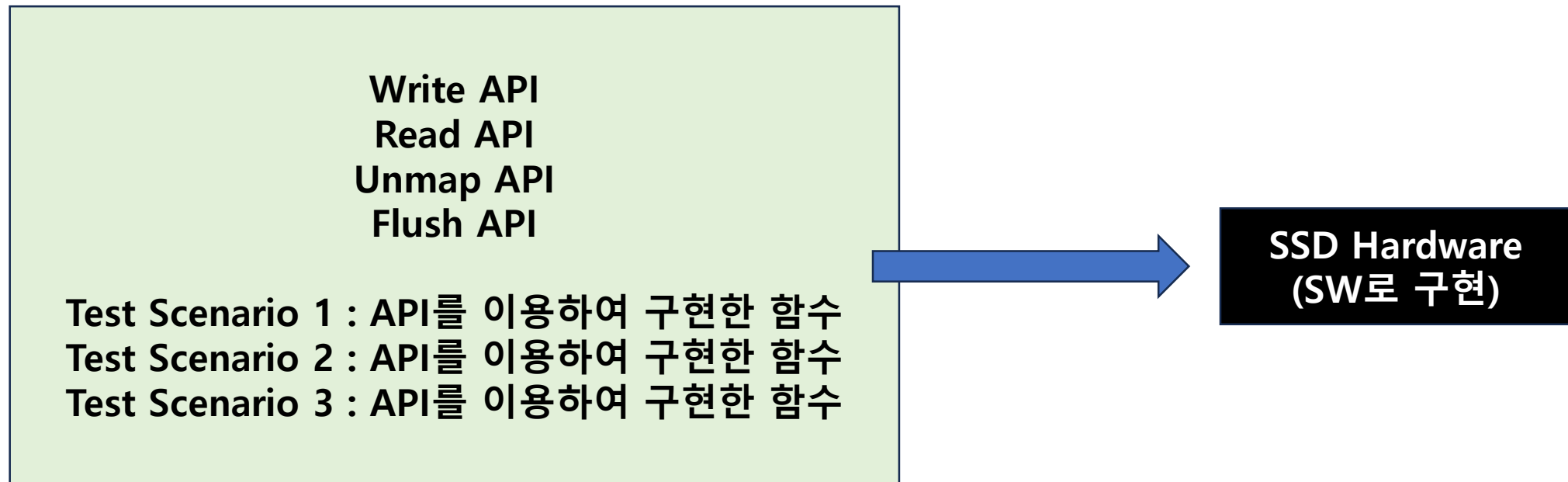


Runner 기능 추가

Test Script Name을 한 Text file에 모아, 순차적으로 수행하는 기능

현재 구조

각 API들을 사용하여, Test Scenario들이 만들어져있다.



Runner

테스트 시나리오들은
하나의 함수 형태로 만들어져 있으며,
Shell에서 하나의 시나리오 씩 수행할 수 있다.

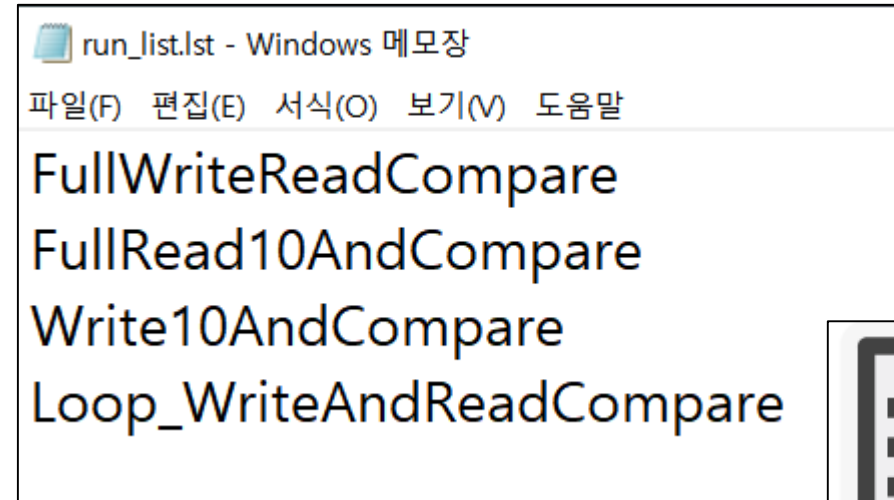
검증 팀에서는
미리 제작한 시나리오 이름을 txt파일로 저장한 후,
txt파일 이름만 입력하면, 기입된 시나리오들을
하나씩 순차적으로 실행하는 기능을 추가하고자 한다.

예시

run_list 파일에 실행시키고 싶은 시나리오 이름들을 기입한다.
그리고 cmd 창에서

shell.exe run_list.lst

이라고 수행하면
테스트가 하나씩 수행된다.



Test Scenario 4개 기입 예시



결과 화면

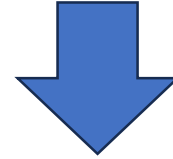
1 화면에는 동작 Text 외, 다른 로그 Text가 출력되지 않는다.
스크립트 이름 --- Run ... 이 먼저 출력된다.

2 테스트가 끝나고 Pass가 되면 Pass 출력 후,
다음 스크립트가 수행된다.

3 테스트가 Fail로 끝나면 FAIL! 출력 후
즉시 프로그램이 종료되며
더이상 스크립트를 수행하지 않는다.

1

```
FullWriteReadCompare --- Run...Pass
FullRead10AndCompare --- Run...Pass
Write10AndCompare --- Run...
```



2

```
FullWriteReadCompare --- Run...Pass
FullRead10AndCompare --- Run...Pass
Write10AndCompare --- Run...Pass
Loop_WriteAndReadCompare --- Run...
```



3

```
FullWriteReadCompare --- Run...Pass
FullRead10AndCompare --- Run...Pass
Write10AndCompare --- Run...Pass
Loop_WriteAndReadCompare --- Run...FAIL!
```

SSD - Write Buffer 기능 추가

Write Buffer 및 Flush 명령어 기능 추가

SSD – Write Buffer 기능 1

Write Buffer 기본 기능

- SSD에 Write / Erase를 수행할 명령어를 모아, 한번에 수행한다.
- SSD 자체 기능을 추가한다.
- Write Buffer 에 10개의 명령어 까지 Buffering 될 수 있다.
- Write Buffer 에 10개의 명령어가 차면, 명령어를 한꺼번에 수행한다.

SSD - Flush 명령어 추가

- Write Buffer에 명령어가 다 채워지지 않아도, 즉시 SSD에 적용시킨다.
- SSD 명령어 이름 : F
Shell 명령어 이름 : flush

SSD – Write Buffer 기능 2

Fast Write 기능

- Buffer 내부에 연속적인 Range에 대한 Erase 명령어가 있어, 두 명령어를 한번의 명령어로 교체할 수 있다면 이를 교체한다.
- 같은 LBA에 Write와 Erase 명령어가 들어왔다면, Write 명령을 수행할 필요가 없기에 Erase만 수행한다.

Fast Read 기능

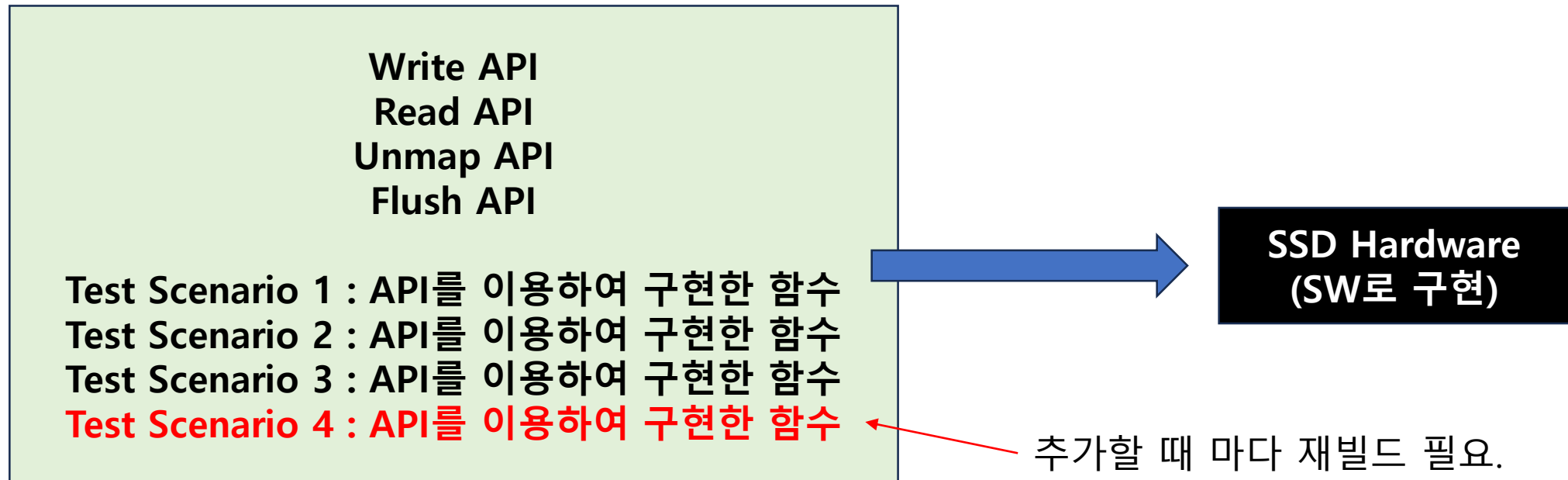
- Read를 수행할 때, Write Buffer 부터 확인을 한다.
- Write Buffer에 Read할 값이 적혀 있다면, 실제 NAND를 읽지 않고도 값을 읽어 리턴한다.

시나리오 추가시, 재빌드 이슈 해결

난이도 HARD, Test Shell의 Script 관리 방법을 변경하자.

Test Scenario를 추가할 때마다 전체 재빌드 필요.

매번 시나리오를 추가할 때 마다,
셀을 재빌드하는 문제가 있다.



재빌드하는데 걸리는 시간이 너무 크다.

Test Scenario 함수들은
매우 많으며,
소스파일은 300개가 넘는다.

이 때문에
한번 빌드하는데 걸리는 시간은
5분 걸린다고 가정하자.

- 사소한 버그 날때마다 재빌드 (5분 소요)
- 한번 테스트할때마다 재빌드 (5분 소요)

이로 인해 개발이 힘들다.

Write API
Read API
Unmap API
Flush API

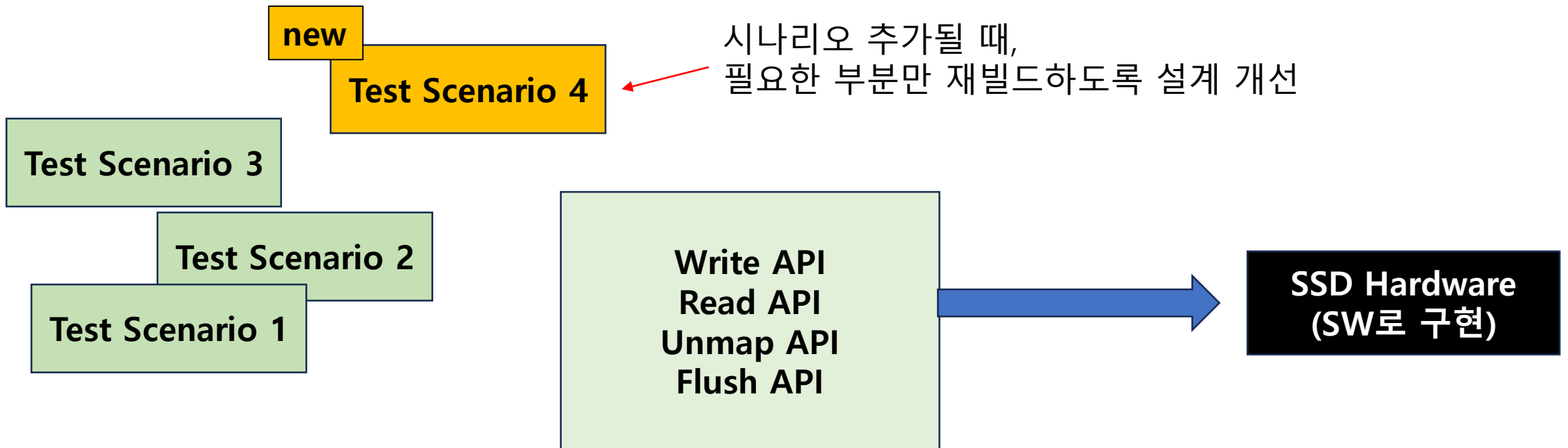
Test Scenario 1 : API를 이용하여 구현한 함수
Test Scenario 2 : API를 이용하여 구현한 함수
Test Scenario 3 : API를 이용하여 구현한 함수
Test Scenario 4 : API를 이용하여 구현한 함수

...

Test Scenario 999 : API를 이용하여 구현한 함수
Test Scenario 1000 : API를 이용하여 구현한 함수

Test Scenario를 추가할 때마다 전체 재빌드 필요.

재빌드 해야하는 규모를 최소화할 수 있는
설계로 개선하는 리팩토링을 수행한다.



기능 추가 미션, 유의사항

미션 수행시, 유의사항

유의사항 1

1. 협업이 잘 되도록 역할분담을 해주세요.
2. 팀장님을 제외한 코드리뷰어를 1명 이상 꼭 지정해주세요.
팀원들이 개발한 코드는 코드리뷰어가 리뷰해주시면 됩니다.
코드리뷰어가 개발한 코드는 팀장님이 리뷰해주세요.
3. 시간 간격을 협의하셔서, 정해진 시간별로 코드리뷰 담당자를 변경해주세요.

유의사항 2

4. 3일차 부터는 TDD로 개발하지 않아도 됩니다.

5. 일부러 지저분하게 작성 후 Commit을 해주어도 좋습니다.
개발과 리팩토링을 동시에 하는 것을 자제해주세요.

6. 개발할 때는 마음 편하게
변수명, 함수명 네이밍 등을 사용해도 좋습니다

7. 개발 후 동작이 잘 되면 Commit을 하고,
클린코드가 되게끔 리팩토링을 하여 Commit을 해주세요.
전 후 비교되게끔 해주시면, 발표자료를 만들 때 사용하기 좋습니다.