

애자일 방법론

제공된 자료는 훈련생의 수업을 돕기 위한 것으로, 타인과 공유하시면 안 됩니다.

Contents

part.1

애자일 방법론

part.2

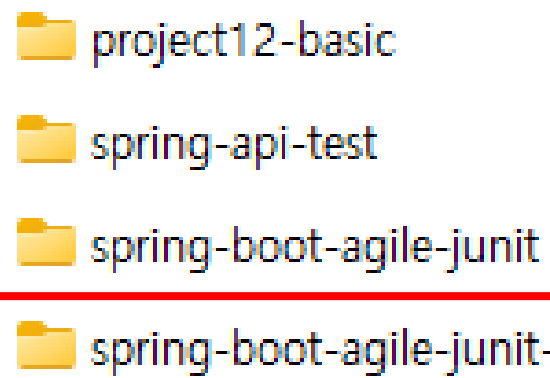
TDD

part.3

단위 테스트

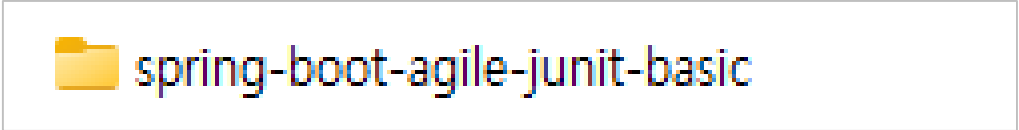
해당 프로젝트는 부분적으로 완성된 구조를 사용할 예정입니다.

spring-study 저장소에서 "spring-boot-agile-junit-basic"을 복사하고, 본인의 저장소로 가져오세요.



project12-basic
spring-api-test
spring-boot-agile-junit
spring-boot-agile-junit-basic

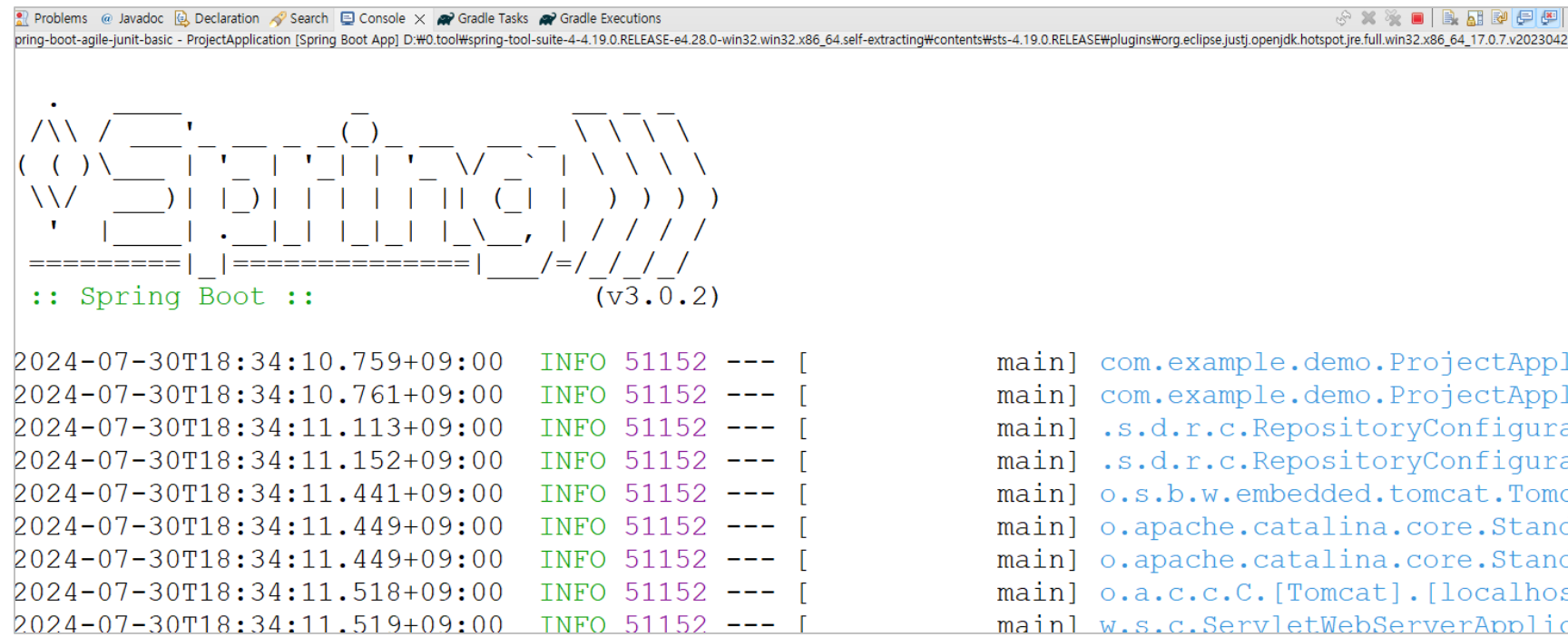
spring-study 폴더



spring-boot-agile-junit-basic

my-workspace 폴더

프로젝트가 정상적으로 실행되는지 확인하세요.



```
Problems @ Javadoc Declaration Search Console x Gradle Tasks Gradle Executions
spring-boot-agile-junit-basic - ProjectApplication [Spring Boot App] D:\#0.tool\spring-tool-suite-4-4.19.0.RELEASE-e4.28.0-win32.win32.x86_64.self-extracting\contents\sts-4.19.0.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.7.v2023042

:: Spring Boot :: (v3.0.2)

2024-07-30T18:34:10.759+09:00 INFO 51152 --- [main] com.example.demo.ProjectApp
2024-07-30T18:34:10.761+09:00 INFO 51152 --- [main] com.example.demo.ProjectApp
2024-07-30T18:34:11.113+09:00 INFO 51152 --- [main] .s.d.r.c.RepositoryConfigura
2024-07-30T18:34:11.152+09:00 INFO 51152 --- [main] .s.d.r.c.RepositoryConfigura
2024-07-30T18:34:11.441+09:00 INFO 51152 --- [main] o.s.b.w.embedded.tomcat.Tomc
2024-07-30T18:34:11.449+09:00 INFO 51152 --- [main] o.apache.catalina.core.Stand
2024-07-30T18:34:11.449+09:00 INFO 51152 --- [main] o.apache.catalina.core.Stand
2024-07-30T18:34:11.518+09:00 INFO 51152 --- [main] o.a.c.c.C.[Tomcat].[localhos
2024-07-30T18:34:11.519+09:00 INFO 51152 --- [main] w.s.c.ServletWebServerApplic
```

애자일 방법론

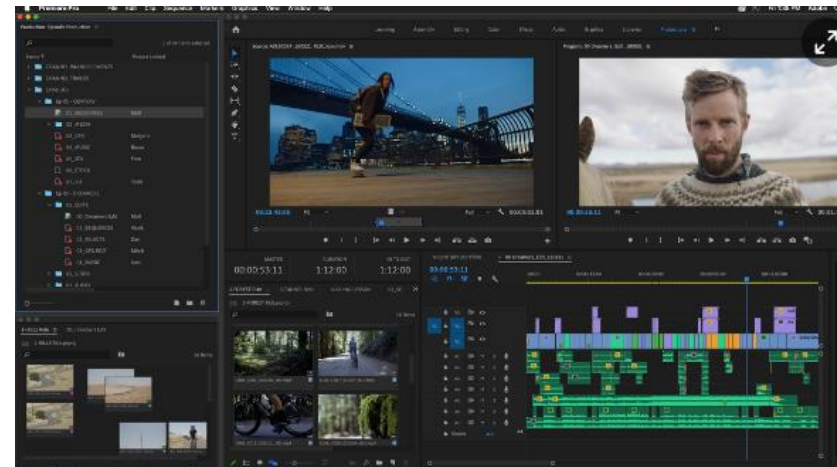
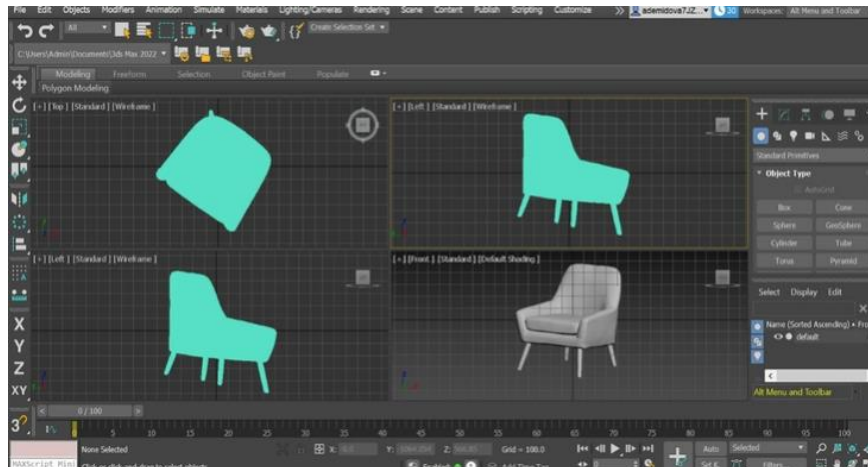
개발 방법론

개발 방법론은 개발자가 클라이언트의 요구사항을 반영하여 프로그램을 구현하기 위한 방식이다.
이를 통해 개발자는 클라이언트의 요구사항을 정확하게 파악하고, 빠르게 구현할 수 있다.
대표적인 개발 방법론은 다음과 같은 것들이 있다.

폭포수 모델 (Waterfall Model) : 각 단계를 순차적으로 진행하는 방식

애자일 방법론 (Agile Methodology) : 고객의 요구가 변경될 때마다 빠르게 대응하는 방식

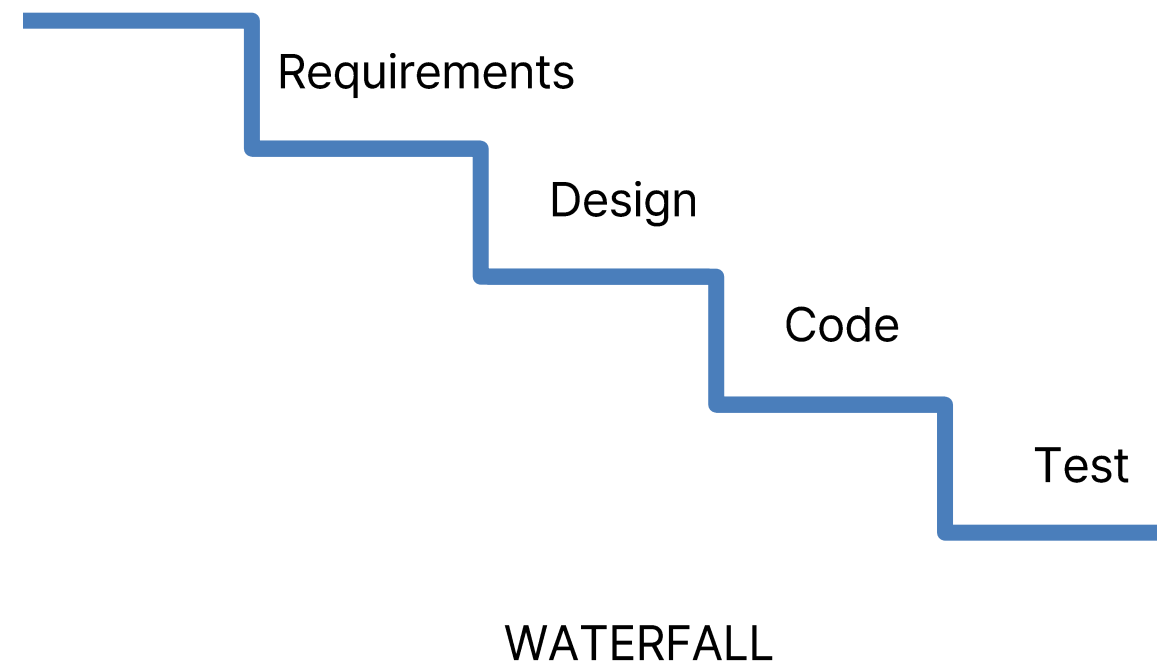
스프린트 모델 (Scrum) : 애자일 방법론 중 하나로, 특정기간(스프린트)동안 목표를 달성하는 방식



다양한 소프트웨어들

폭포수(WATERFALL) 모델은 단계를 순차적으로 진행하는 방식으로 요구사항 분석, 설계, 구현, 테스트, 배포 순으로 이루어진다.

1. 요구사항 분석: 사용자의 요구사항을 분석하여 문서화
2. 설계: 요구사항에 따라 소프트웨어 구조 정의
3. 구현: 설계에 따라 소프트웨어 개발
4. 테스트: 소프트웨어가 요구사항을 충족하고 버그가 없는지 확인
5. 배포: 실제 서버에 소프트웨어 설치



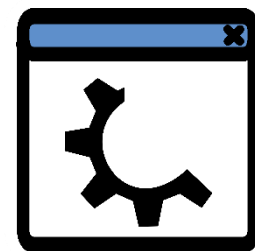
폭포수(WATERFALL) 모델의 문제점

폭포수 모델은 프로젝트 초기 단계에서 클라이언트의 요구사항을 정확히 파악하지 못할 경우 문제가 발생할 수 있다. 이 모델에서는 한번 진행된 단계를 되돌아가 수정하는 것이 어렵기 때문에, 요구사항을 변경하면 추가비용이 크게 증가한다. 그래서 개발자가 클라이언트의 요구사항을 유동적으로 대응하기가 어렵다.

프로그램 구현이
완료되었습니다!



개발자



클라이언트

내가 원하는건
이게 아닌데..?

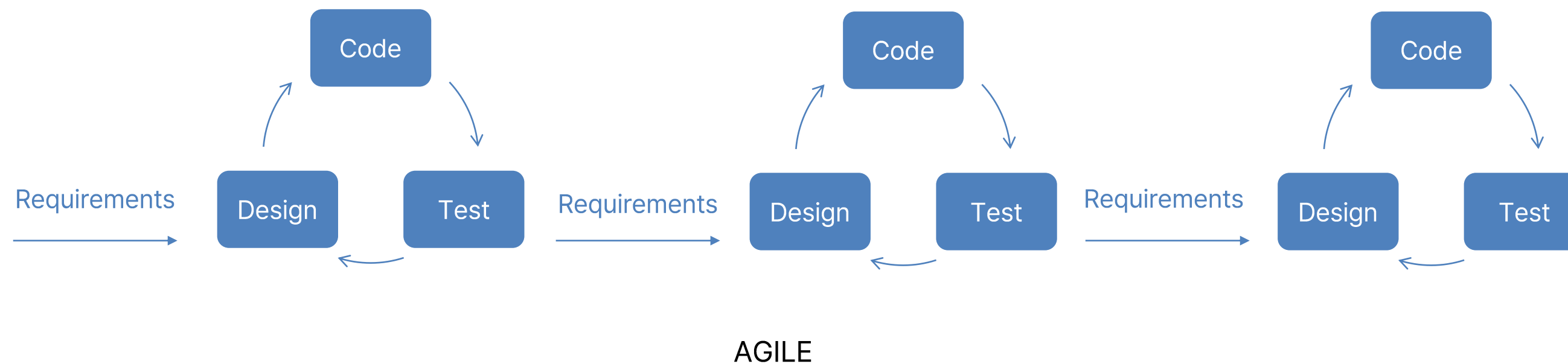
애자일 방식이란?

1990년대 후반까지는 폭포수 모델을 주로 사용했다.

하지만 소프트웨어와 시장이 빠르게 변화하면서 어려움이 생겼다.

폭포수 모델은 서비스가 고도화될수록 사용자의 요구사항에 빠르게 대응하기에는 한계가 있었다.

이에 요구사항 변화에 유연하게 대응할 수 있는 '애자일 방법론'이 등장하게 되었다.



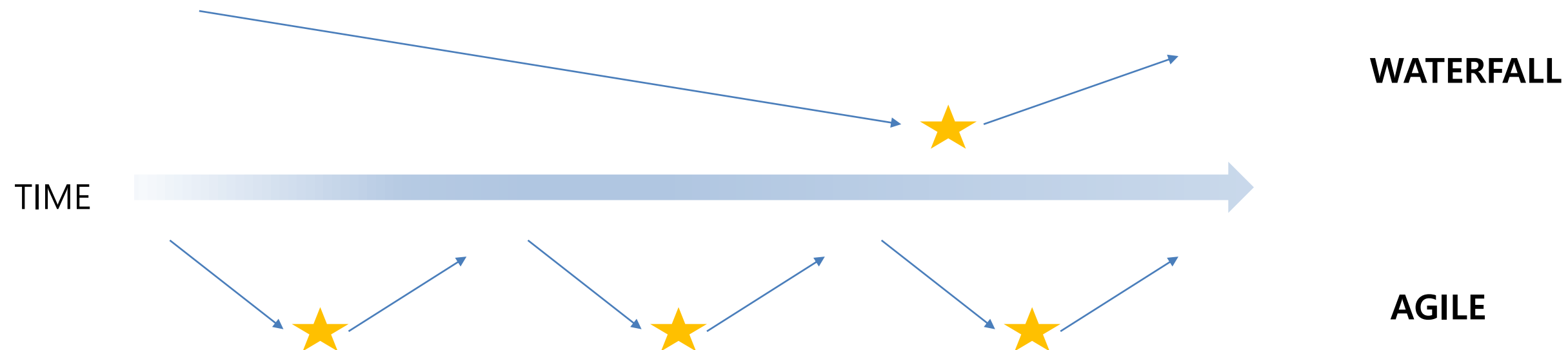
애자일 방식의 장점

애자일 방식의 장점은 클라이언트의 요구사항을 지속적으로 확인하고 수정할 수 있다는 점이다.

워터폴 방식은 긴 시간 동안 개발을 진행하기 때문에 요구사항이 자주 점검하기 어렵다.

반면, 애자일은 짧은 주기로 데모를 만들기 때문에, 요구사항이 맞는지 자주 확인할 수 있다.

따라서 개발 도중 문제가 생기더라도 빨리 조치할 수 있다.



애자일 방법론

애자일 방법론

애자일 방식의 단점

프로젝트 초기 단계에서 WBS 또는 스프린트 같은 개발 일정을 작성해야 한다.

개발자는 이 일정에 따라 작업을 진행하고, 계획이 제대로 진행되고 있는지 점검해야 한다.

그런데 애자일 방식은 주기적으로 변화하는 요구사항에 맞춰 일정을 수정해야 한다.

이로 인해 실무에서는 개발 일정 관리가 복잡해지고, 개발 방법론을 적용하는데 어려움이 있을 수 있다.

단계	핵심배치	가능	일일작	산출물	단위별 공정률	잔량	잔량 비율	작업일정	시작일	종료일	지연	2/19	2/26	3/5	3/12
설계	개요정의	개요 구성		개요 구성도	0%	21.00	0.01	0.00	2022-01-20	2022-02-09					
	컨텐츠 / PSD	컨텐츠 정의		컨텐츠 정의서	0%	23.00	0.01	0.00	2022-01-20	2022-02-11					
	Front Main			화면설계서	0%	27.00	0.01	0.00	2022-01-20	2022-02-15					
	스토리보드 작성	UI		화면설계서	0%	31.00	0.02	0.00	2022-01-20	2022-02-19					
	Admin			화면설계서	0%	27.00	0.01	0.00	2022-01-20	2022-02-15					
	기타 Page			화면설계서	0%	22.00	0.01	0.00	2022-01-20	2022-02-10					
	디자인 시안	Front Main		PSD	0%	27.00	0.01	0.00	2022-01-20	2022-02-15					
	디자인 스킴을 기반으로			PSD	0%	27.00	0.01	0.00	2022-01-20	2022-02-15					
	수정 사항	기밀설계		수정사항 명세서	0%	33.00	0.02	0.00	2022-01-20	2022-02-21					
	개발준비	개발준비							2022-01-20	2022-02-20					
구현(개발)	프론트입력	프론트입력		프론트입력	0%	32.00	0.02	0.00	2022-01-20	2022-02-20					
	백엔드입력	백엔드입력		백엔드입력					2022-01-20	2022-02-20					
	데이터베이스	데이터베이스		데이터베이스					2022-01-20	2022-02-20					
	테스트	테스트		테스트					2022-01-20	2022-02-20					
	배포	배포		배포					2022-01-20	2022-02-20					
	로그인 Topbar			로그인 Topbar					2022-02-20	2022-03-20					
	회원가입			회원가입					2022-02-20	2022-03-20					
	회원탈퇴			회원탈퇴					2022-02-20	2022-03-20					
	회원관리			회원관리					2022-02-20	2022-03-20					
	회원관리			회원관리					2022-02-20	2022-03-20					

WBS

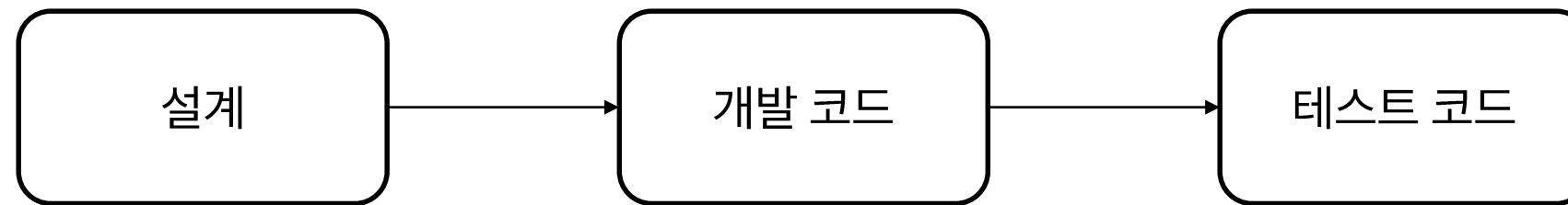
2월 4주차 목표		
1) 광고 전환 수 100개 이상 만들기 2) CPA 10,000원 이하 만들기		
목표	비고	담당자
광고 전환 100개 이상 만들기	CPA 10,000원 이하 만들기	이안, 리오, 세라, 올리버
2월 4주차 스프린트		
할 일 1	진행 중 3	완료 1
광고 라이브 실행 2024년 2월 22일 → 2024년 2월 28일	광고 소재 기획안 작성 2024년 2월 19일 → 2024년 2월 20일	데이터 세팅 2024년 2월 19일 → 2024년 2월 20일
+ 새로 만들기	광고 소재 디자인	+ 새로 만들기

스프린트

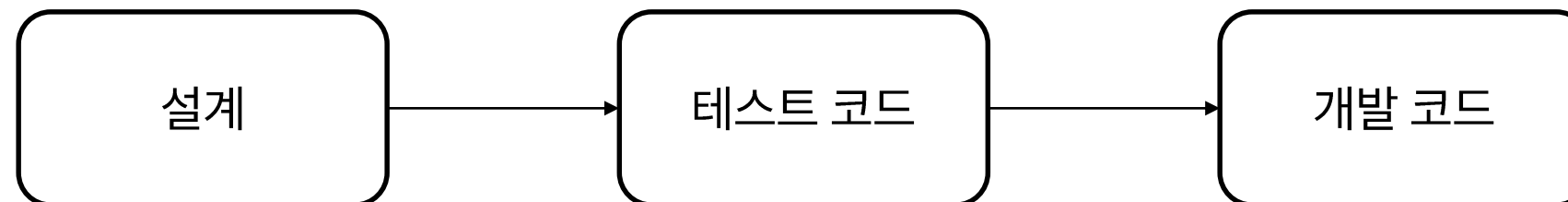
TDD란?

TDD(테스트 주도 개발 방식, Test-driven development)는 개발 방법론 중 하나로, 테스트 코드 기반으로 개발하는 방식이다. 먼저 요구사항을 검증할 수 있는 테스트 코드를 작성하고, 실제 구현 코드를 작성한다. 이를 통해 요구사항에 맞는 기능을 정확하게 구현할 수 있다.

기존 방식



TDD 방식



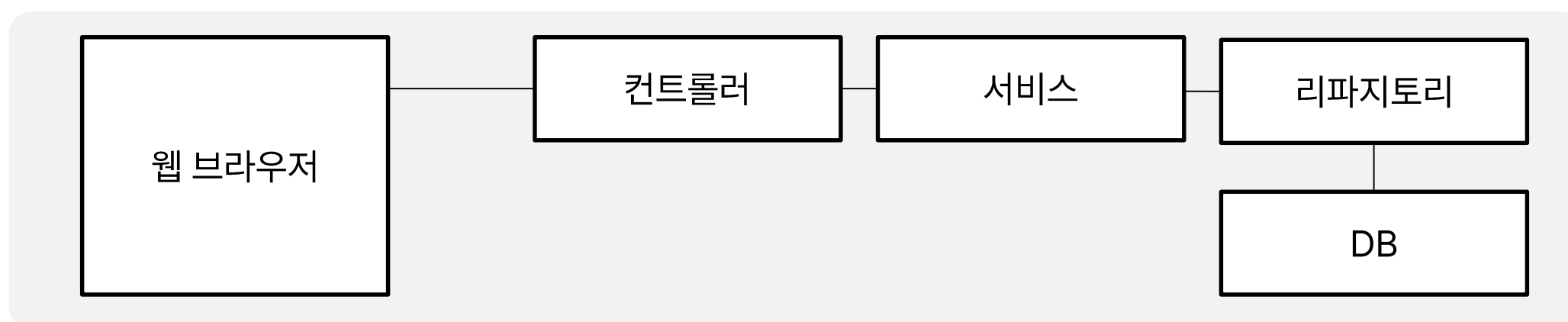
TDD 방식은 테스트 코드를 작성하여 기능이 정확하게 동작하는지 검증해야 한다.

통합 테스트

여러 모듈이 통합되어 전체 시스템이 올바르게 동작하는지 확인한다.

통합테스트는 단위테스트를 먼저 수행한 후에 진행한다.

통합 테스트 과정



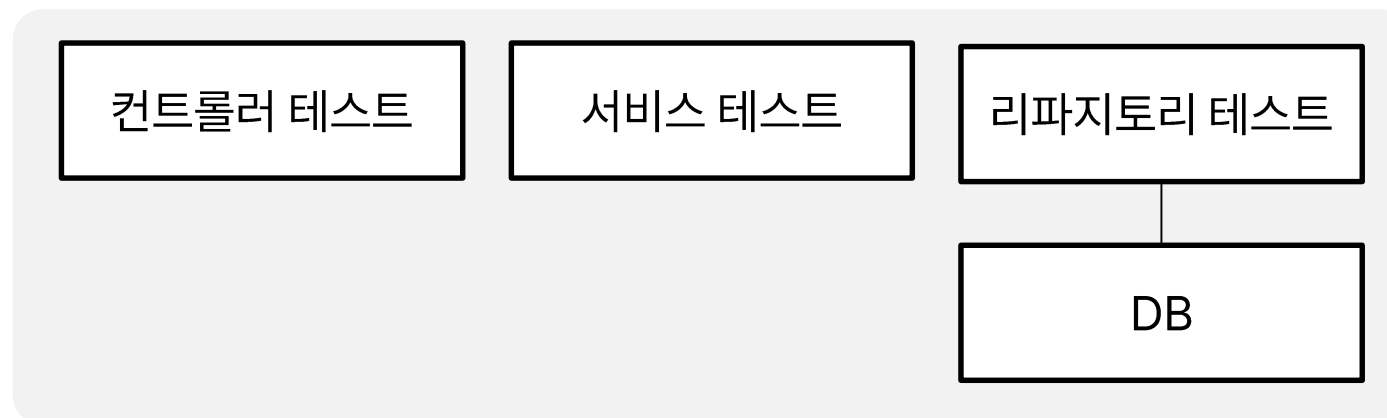
단위 테스트

단위 테스트는 코드의 작은 단위(메소드나 클래스)가 잘 동작하는지 확인하는 절차이다.

이 테스트는 특정 모듈이 정상적으로 동작하는지 확인하는데 중점을 둔다.

테스트 대상은 컨트롤러, 서비스, 리파지토리 등 컴포넌트 단위로 구분한다.

단위 테스트 과정

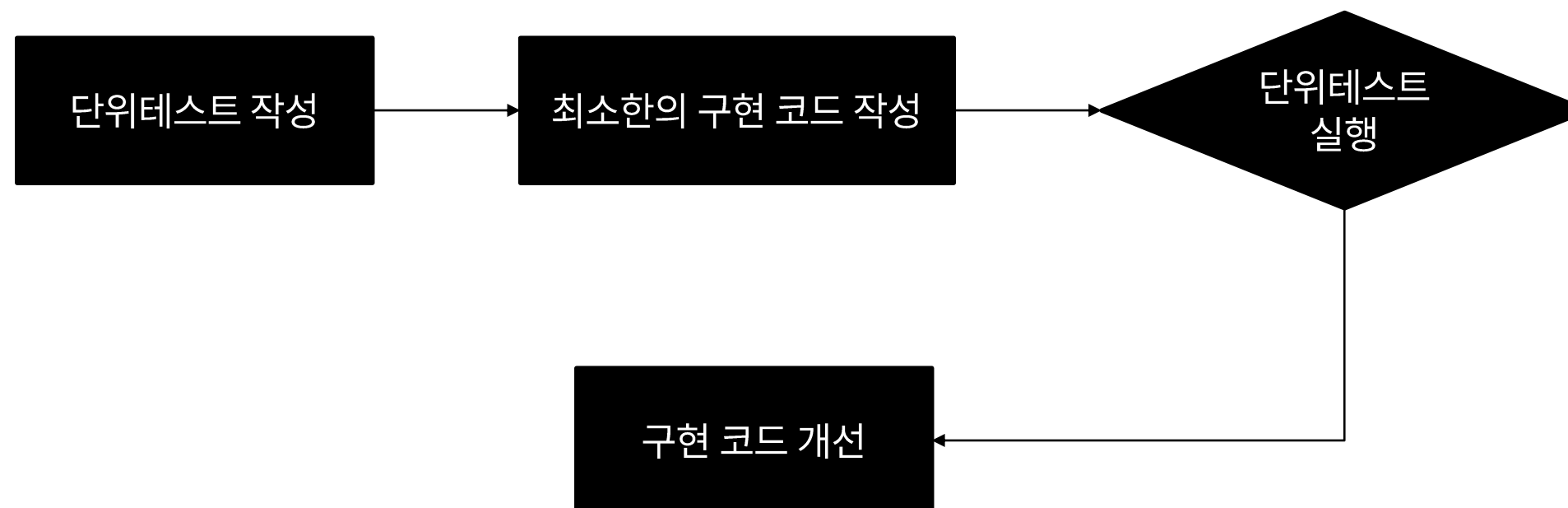


테스트 코드 예시

```
@Test
public void 게시물등록() {
    Member member1 = Member.builder().id("user1").build();
    List<Board> list = new ArrayList<>();
    list.add(new Board(0, "1번글", "내용입니다", member1));
    list.add(new Board(0, "2번글", "내용입니다", member1));
    list.add(new Board(0, "3번글", "내용입니다", member1));
    repository.saveAll(list);
}
```

TDD 테스트 순서

1. 새 기능을 추가할 때 요구 사항을 충족하는 테스트를 먼저 작성한다.
2. 이 테스트를 통과하기 위한 "최소한의 코드"를 작성한다.
3. 작성한 코드가 테스트를 통과하는지 확인한다.
4. 기능을 보완한 후, 다시 테스트가 통과하는지 확인한다.



자바에서는 JUNIT을 사용하여 단위 테스트 코드를 작성할 수 있다.
JUNIT은 어노테이션을 통해 테스트 환경을 설정하여 실제 어플리케이션과 유사한 환경을 만든다.
이를 통해 리파지토리나 서비스 컴포넌트의 기능을 확인할 수 있다.

```
@SpringBootTest
public class BoardRepositoryTest {

    @Autowired
    BoardRepository repository;

    @Test
    void 게시물등록 () {
        Board board = Board.builder()
            .title("1번글")
            .content("내용입니다")
            .writer("둘리")
            .build();

        repository.save(board);
    }
}
```

리파지토리 단위테스트

```
@SpringBootTest
public class BoardServiceTest {

    @Autowired
    BoardService service;

    @Test
    public void 게시물등록 () {
        BoardDTO dto = BoardDTO.builder()
            .title("2번글")
            .content("내용입니다")
            .writer("또치")
            .build();

        int no = service.register(dto);
        System.out.println("새로운 게시물 번호: " + no);
    }
}
```

서비스 단위테스트

TDD방식으로 단위 테스트를 실행할 때는 given-when-then 패턴을 자주 사용한다.

이 패턴은 테스트의 각 단계를 구분하여 테스트 시나리오를 체계적으로 작성할 수 있다.

- given: 테스트 데이터 설정
- when: 테스트 하려는 동작 수행
- then: given-when절을 통해 나온 결과가 원하는 결과와 일치하는 지 확인

Spring 프로젝트에서는 JUnit 과 AssertJ를 사용하여 given-when-then 패턴으로 테스트를 진행할 수 있다.

41.1 Test scope dependencies

If you use the `spring-boot-starter-test` 'Starter' (in the

- [JUnit](#) — The de-facto standard for unit testing Java appl
- [Spring Test](#) & Spring Boot Test — Utilities and integratio
- [AssertJ](#) — A fluent assertion library.
- [Hamcrest](#) — A library of matcher objects (also known as
- [Mockito](#) — A Java mocking framework.
- [JSONassert](#) — An assertion library for JSON.
- [JsonPath](#) — XPath for JSON.

게시물 서비스의 기능을 확인하기 위해 given-when-then 방식을 사용하여 테스트 코드를 작성한다.

기존 방식은 단순히 결과를 출력하고 끝나지만, `assertThat`를 사용하면 예상 결과와 다를 경우

테스트가 실패 처리되어 결과를 더 정확하게 확인할 수 있다

이를 위해 AssertJ 라이브러리의 `assertThat`를 사용하여 테스트를 진행한다.

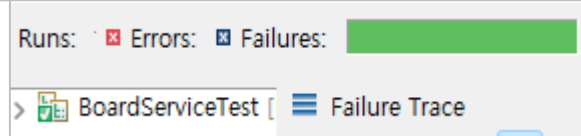
테스트 시나리오가 의도한 대로 동작하는지 확인하며 예상 결과와 일치하면 단위 테스트가 성공, 그렇지 않으면 실패한다

```
@Test
void 게시물등록 () {

    // Given: 게시물 생성
    BoardDTO dto = BoardDTO.builder().title("1번글")

    // When: 저장
    int no = service.register(dto);

    // Then: 새로 생성된 게시물의 번호가 1번이 맞는지 확인
    assertThat(no).isEqualTo(1);
}
```



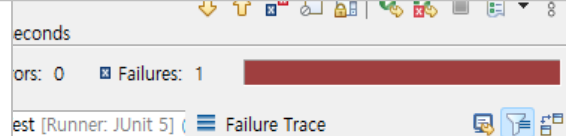
예상 결과가 맞으면 성공

```
@Test
void 게시물등록 () {

    // Given: 게시물 생성
    BoardDTO dto = BoardDTO.builder().title("1번글")

    // When: 저장
    int no = service.register(dto);

    // Then: 새로 생성된 게시물의 번호가 2번이 맞는지 확인
    assertThat(no).isEqualTo(2);
}
```



예상 결과가 틀리면 실패

처음에는 단위 테스트를 통과할 수 있도록 최소한의 코드만 작성한다.

이 후, 코드를 리팩토링을 하여 기능을 개선한다.

게시물 서비스의 삭제 메소드를 보완하여, 게시물이 존재하는지 확인하고 삭제하도록 수정한다.

그리고 단위 테스트를 다시 실행하여 기능이 정상적으로 동작하는지 확인한다.

이렇게 하면 테스트 작성 → 최소한의 코드 작성 → 리팩토링이라는 TDD의 순서를 따라갈 수 있다.

```
@Override
public void remove(int no) {

    // 기존 게시물이 존재하는지 확인하고 삭제
    Optional<Board> result = repository.findById(no);
    if (result.isPresent()) {
        repository.deleteById(no);
    }
}
```

삭제 메소드

```
@Test
public void 게시물삭제 () {
    // Given: 게시물번호
    int boardNo = 1;

    // When: 게시물 삭제
    service.remove(boardNo);

    // Then: 게시물이 삭제되었는지 확인
    BoardDTO dto = service.read(boardNo);
    assertThat(dto).isNull();
}
```

단위 테스트

