



잘못된 테스트 사례 학습

Java/Spring 테스트를 추가하고 싶은 개발자들의 오답노트

왜 내가 하는 TDD는 실패하는가?

개요

테스트가 무엇인지 알아보고, 테스트를 하는 개발자의 고민을 같이 살펴봅니다.

제 1장  테스트란?

제 2장  TDD

제 3장  개발자의 고민

1. 테스트란?

01. 테스트

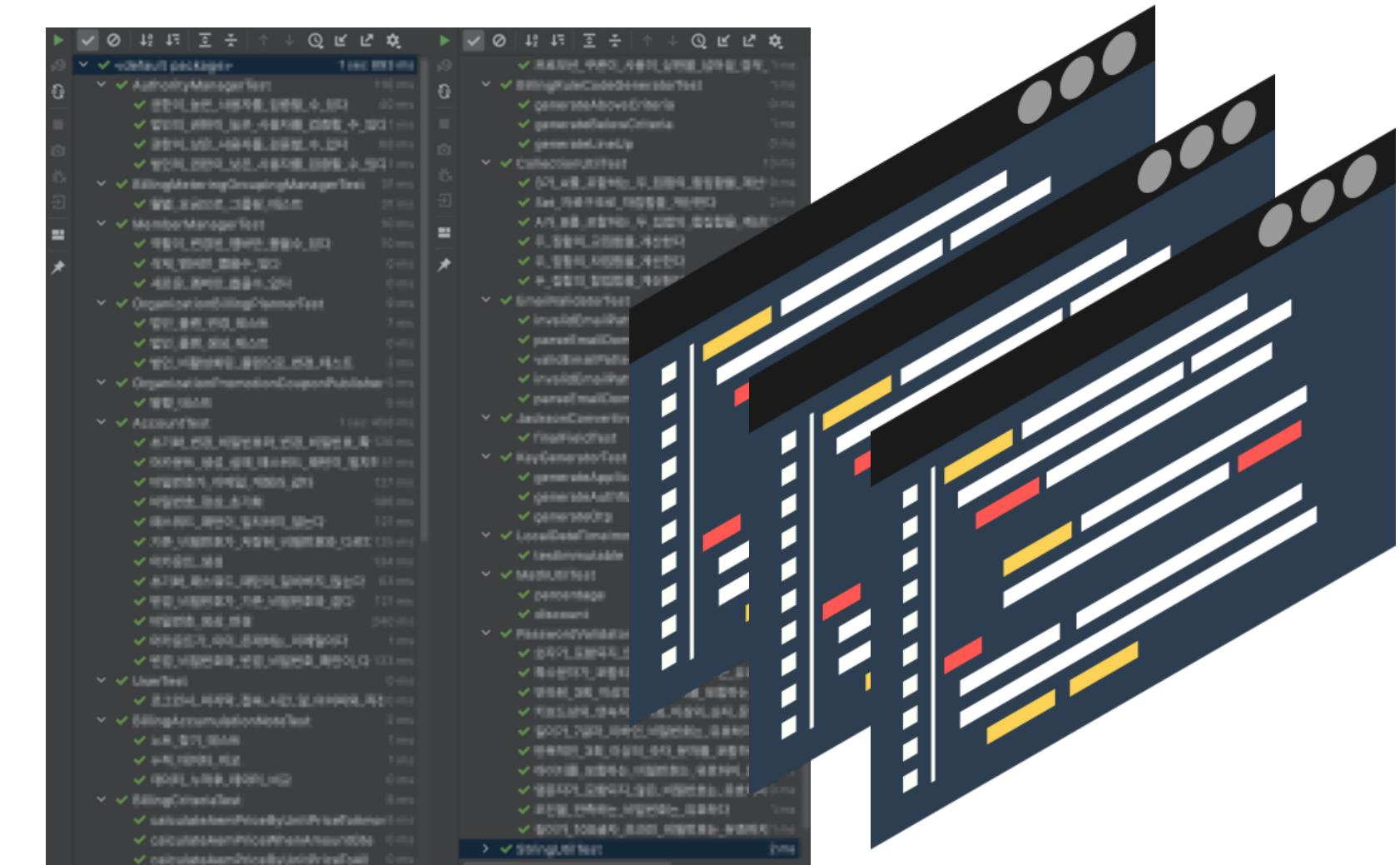
테스트가 무엇인지 살펴봅니다.

1. 테스트란?

테스트란 무엇일까요?



인수 테스트



자동 테스트

1. 테스트란?

테스트란 무엇일까요?



인수 테스트



자동 테스트

1. 테스트란?

테스트란 무엇일까요? 샘플 (1)



```
@Test
public void 이메일_회원가입을_할_수_있다() {
    // given
    UserCreateRequest userCreateRequest = UserCreateRequest.builder()
        .email("foo@localhost.com")
        .password("123456")
        .build();
    FakeRegisterEmailSender registerEmailSender = new FakeRegisterEmailSender();

    // when
    UserService sut = UserService.builder()
        .registerEmailSender(registerEmailSender)
        .userRepository(userRepository)
        .build();
    sut.register(userCreateRequest);

    // then
    User user = userRepository.getByEmail("foo@localhost.com");
    assertThat(user.isPending()).isTrue();
    assertThat(registerEmailSender.findLatestMessage(email: "foo@localhost.com").isPresent()).isTrue();
    assertThat(registerEmailSender.findLatestMessage(email: "foo@localhost.com").get()).isEqualTo("~~");
}
```

1. 테스트란?

테스트란 무엇일까요? 샘플 (2)

The screenshot shows a Java IDE interface with two main panes. The left pane displays a tree view of test run results, with a red arrow pointing to a specific test method named 'Math_는_정수를_더해_정수를_얻을_수_있다()' under the 'MathTest' class. The right pane contains two code files: 'MathTest.java' and 'Math.java'. 'MathTest.java' contains a single test method:

```
class MathTest {  
    @Test  
    void Math_는_정수를_더해_정수를_얻을_수_있다() {  
        // given  
        int a = 10;  
        int b = 20;  
  
        // when  
        int result = Math.add(a, b);  
  
        // then  
        assertThat(result).isEqualTo(30);  
    }  
}
```

'Math.java' contains the implementation of the 'add()' method:

```
class Math {  
    public static int add(int a, int b) {  
        return a + b;  
    }  
}
```

1. 테스트란?

테스트란 무엇일까요? 샘플 (3) h2

The screenshot shows a Java IDE with two files open. On the left is a test class named `UserRepositoryTest`, which contains a single test method. On the right is the corresponding `UserRepository` interface. A red arrow points from the test class to the `@Test` annotation, and another red arrow points from the test method to the `assertThat` assertion.

```
class UserRepositoryTest {  
    @Autowired  
    private UserRepository userRepository;  
  
    @Test  
    void 사용자_정보를_save로_저장_할_수_있다() {  
        // given  
        UserEntity userEntity = new UserEntity();  
        userEntity.setEmail("kok202@naver.com");  
        userEntity.setStatus(UserStatus.PENDING);  
        userEntity.setCertificationCode("hash1234567890");  
        userEntity.setAddress("Seoul");  
        userEntity.setLastLoginAt(Clock.systemUTC().millis());  
  
        // when  
        userEntity = userRepository.save(userEntity);  
  
        // then  
        assertThat(userEntity.getId()).isNotNull();  
    }  
}  
  
public interface UserRepository extends JpaRepository<UserEntity, Long> {  
}
```

2. TDD

01. TDD

TDD가 무엇인지 살펴봅니다.

02. TDD의 장단

TDD의 장단점을 살펴봅니다.

2. TDD

테스트 주도 개발

RED 1. 깨지는 테스트를 먼저 작성한다.

GREEN 2. 깨지는 테스트를 성공시킨다.

BLUE 3. 리팩토링한다.

2. TDD

테스트 주도 개발

RED 1. 깨지는 테스트를 먼저 작성한다.

GREEN 2. 깨지는 테스트를 성공시킨다.

BLUE 3. 리팩토링한다.

```
@Service  
@RequiredArgsConstructor  
public class PostService {  
  
    public PostEntity getPostById(Long id) {  
        throw new RuntimeException("Method is not implemented.");  
    }  
}
```

```
@SpringBootTest  
@TestPropertySource("classpath:test-application.properties")  
class PostServiceTest {  
  
    @Autowired  
    private PostService postService;  
  
    @Autowired  
    private UserRepository userRepository;  
  
    @Autowired  
    private PostRepository postRepository;  
  
    @Test  
    void 게시물을_아이디로_조회_할_수_있다(){  
        // given  
        UserEntity userEntity = userRepository.save(new UserEntity());  
        PostEntity postEntity = new PostEntity();  
        postEntity.setWriter(userEntity);  
        postEntity.setContent("helloworld");  
        postEntity.setCreatedAt(Clock.systemUTC().millis());  
        postEntity.setModifiedAt(Clock.systemUTC().millis());  
        postEntity = postRepository.save(postEntity);  
  
        // when  
        PostEntity result = postService.getPostById(postEntity.getId());  
  
        // then  
        assertThat(result.getContent()).isEqualTo("helloworld");  
    }  
}
```

2. TDD

테스트 주도 개발

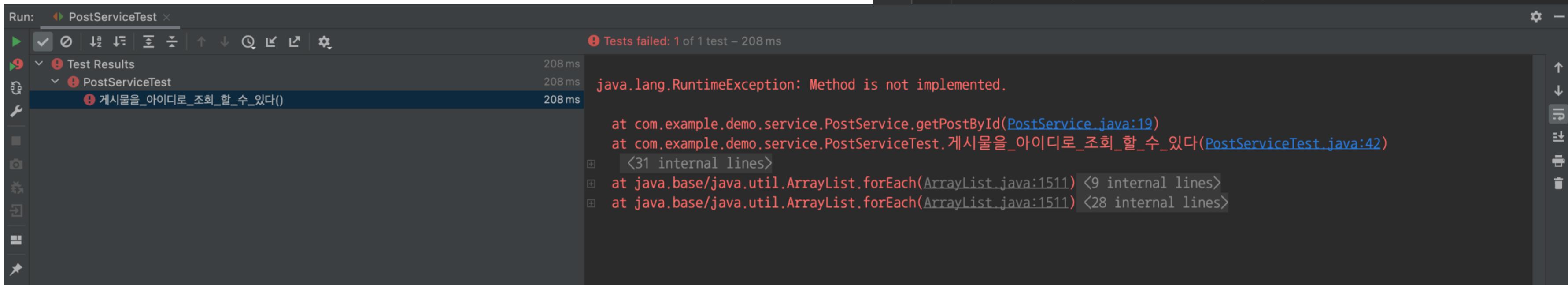
RED 1. 깨지는 테스트를 먼저 작성한다.

GREEN 2. 깨지는 테스트를 성공시킨다.

BLUE 3. 리팩토링한다.

```
@Service  
@RequiredArgsConstructor  
public class PostService {  
  
    public PostEntity getPostById(Long id) {  
        throw new RuntimeException("Method is not implemented.");  
    }  
}
```

```
@SpringBootTest  
@TestPropertySource("classpath:test-application.properties")  
class PostServiceTest {  
  
    @Autowired  
    private PostService postService;  
  
    @Autowired  
    private UserRepository userRepository;  
  
    @Autowired  
    private PostRepository postRepository;  
  
    @Test  
    void 게시물을_아이디로_조회_할_수_있다(){  
        // given  
        UserEntity userEntity = userRepository.save(new UserEntity());  
        PostEntity postEntity = new PostEntity();  
        postEntity.setWriter(userEntity);  
    }  
}
```



2. TDD

테스트 주도 개발

RED 1. 깨지는 테스트를 먼저 작성한다.

GREEN 2. 깨지는 테스트를 성공시킨다.

BLUE 3. 리팩토링한다.

```
@Service  
@RequiredArgsConstructor  
public class PostService {  
  
    private final PostRepository postRepository;  
  
    public PostEntity getPostById(Long id) {  
        return postRepository.findById(id)  
            .orElseThrow(() -> new ResourceNotFoundException("Post", id));  
    }  
}
```

```
@Autowired  
private PostService postService;  
  
@Autowired  
private UserRepository userRepository;  
  
@Autowired  
private PostRepository postRepository;  
  
@Test  
void 게시물을_아이디로_조회_할_수_있다(){  
    // given  
    UserEntity userEntity = userRepository.save(new UserEntity());  
    PostEntity postEntity = new PostEntity();  
    postEntity.setWriter(userEntity);  
    postEntity.setContent("helloworld");  
    postEntity.setCreatedAt(Clock.systemUTC().millis());  
    postEntity.setModifiedAt(Clock.systemUTC().millis());  
    postEntity = postRepository.save(postEntity);  
  
    // when  
    PostEntity result = postService.getPostById(postEntity.getId());  
  
    // then  
    assertThat(result.getContent()).isEqualTo("helloworld");  
}
```

2. TDD

테스트 주도 개발

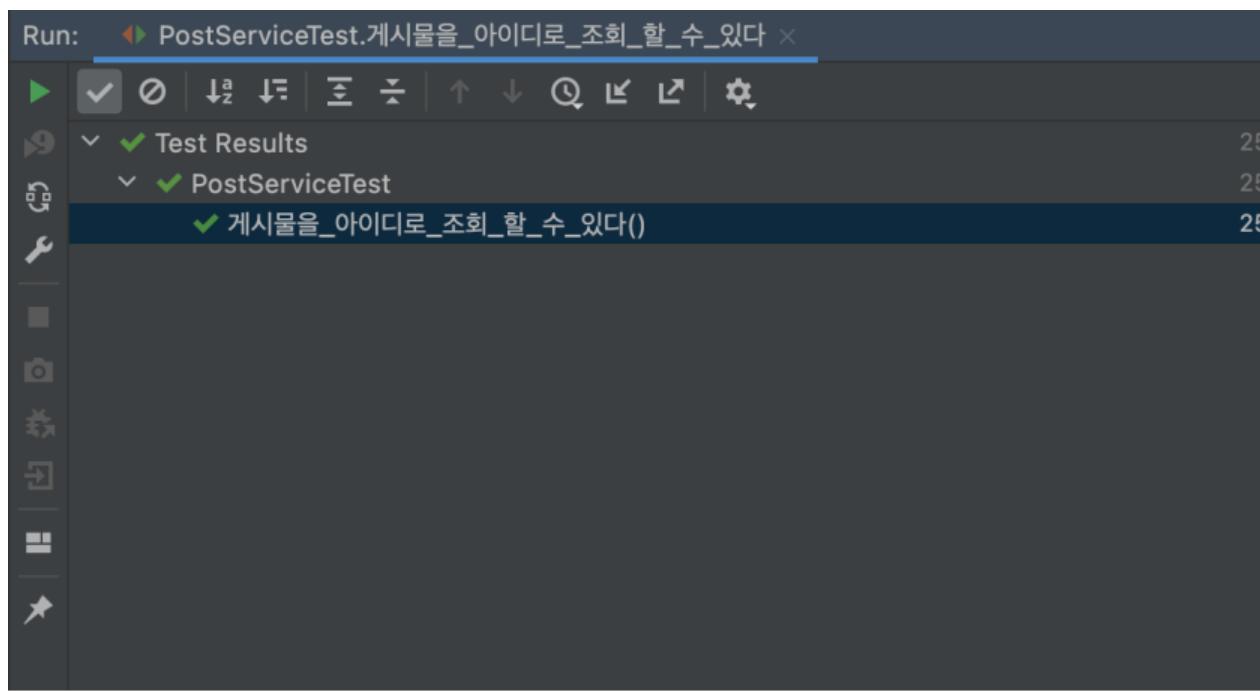
RED 1. 깨지는 테스트를 먼저 작성한다.

GREEN 2. 깨지는 테스트를 성공시킨다.

BLUE 3. 리팩토링한다.

```
@Service  
@RequiredArgsConstructor  
public class PostService {  
  
    private final PostRepository postRepository;  
  
    public PostEntity getPostById(Long id) {  
        return postRepository.findById(id)  
            .orElseThrow(() -> new ResourceNotFoundException("Post", id));  
    }  
}
```

```
@Autowired  
private PostService postService;  
  
@Autowired  
private UserRepository userRepository;  
  
@Autowired  
private PostRepository postRepository;  
  
@Test  
void 게시물을_아이디로_조회_할_수_있다(){  
    // given  
    UserEntity userEntity = userRepository.save(new UserEntity());  
    PostEntity postEntity = new PostEntity();  
    postEntity.setWriter(userEntity);  
}
```



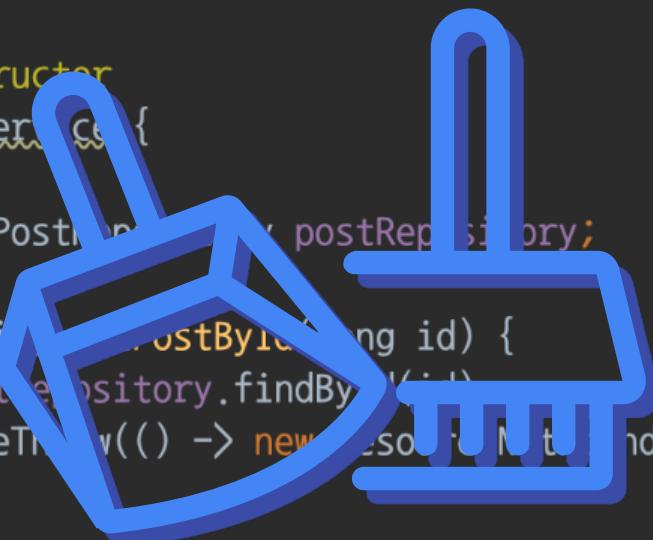
2. TDD

테스트 주도 개발

RED 1. 깨지는 테스트를 먼저 작성한다.

GREEN 2. 깨지는 테스트를 성공시킨다.

BLUE 3. 리팩토링한다.



```
@Service
@RequiredArgsConstructor
public class PostService {
    private final PostRepository postRepository;

    public PostEntity getPostById(Long id) {
        return postRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Post", id));
    }
}
```

```
@Autowired
private PostService postService;

@Autowired
private UserRepository userRepository;

@Autowired
private PostRepository postRepository;

@Test
void 게시물을_아이디로_조회_할_수_있다(){
    // given
    UserEntity userEntity = userRepository.save(new UserEntity());
    PostEntity postEntity = new PostEntity();
    postEntity.setWriter(userEntity);
    postEntity.setContent("helloworld");
    postEntity.setCreatedAt(Clock.systemUTC().millis());
    postEntity.setModifiedAt(Clock.systemUTC().millis());
    postEntity = postRepository.save(postEntity);

    // when
    PostEntity result = postService.getPostById(postEntity.getId());

    // then
    assertThat(result.getContent()).isEqualTo("helloworld");
}
```

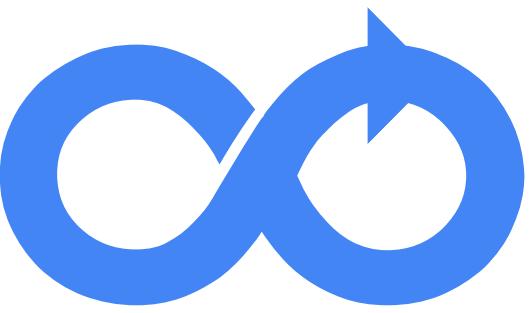
2. TDD

테스트 주도 개발

RED 1. 깨지는 테스트를 먼저 작성한다.

GREEN 2. 깨지는 테스트를 성공시킨다.

BLUE 3. 리팩토링한다.



2. TDD

테스트 주도 개발의 장단점

장점 1. 깨지는 테스트를 먼저 작성해야하기 때문에, 인터페이스를 먼저 만드는 것이 강제된다.

2. TDD

테스트 주도 개발의 장단점

장점 1. 깨지는 테스트를 먼저 작성해야하기 때문에, 인터페이스를 먼저 만드는 것이 강제된다.



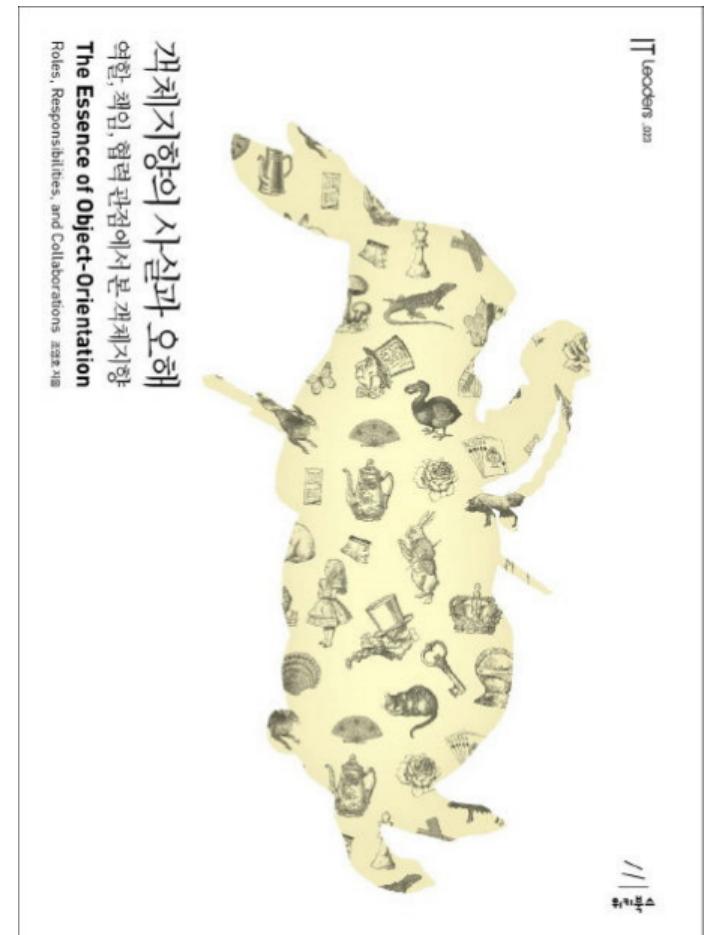
2. TDD

테스트 주도 개발의 장단점

What/Who 사이클

“ What/Who 사이클이라는 용어가 의미하는 것은 객체 사이의 협력 관계를 설계하기 위해서는 먼저 '어떤 행위(waht)'를 수행할 것인지를 결정한 후에 '누가(who)' 그 행위를 수행할 것인지를 결정해야 한다는 것이다. 여기서 '어떤 행위'가 바로 메시지다.

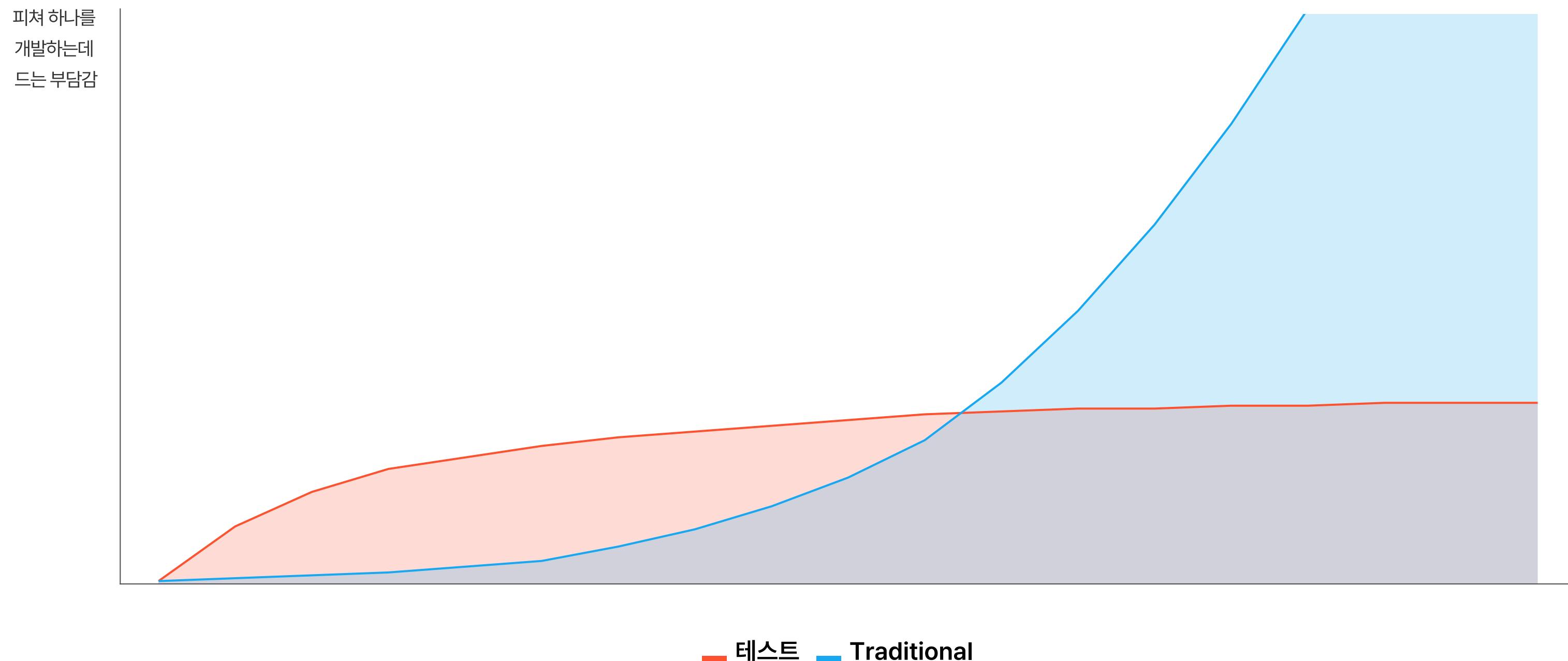
조영호 저, 객체지향의 사실과 오해 역할, 책임, 협력 관점에서 본 객체지향, , (위키북스, 2018-07-31), 158p



2. TDD

테스트 주도 개발의 장단점

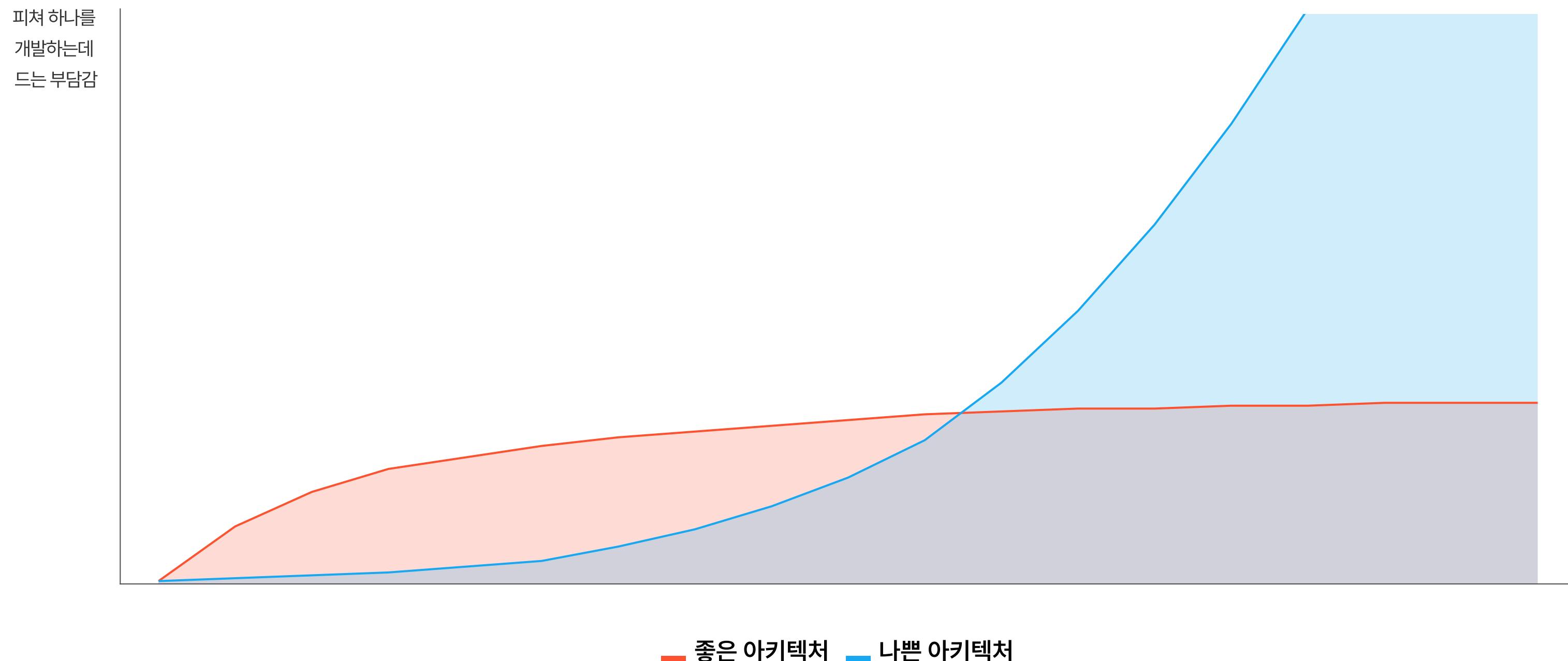
장점 2. 장기적인 관점에서 개발 비용 감소



2. TDD

테스트 주도 개발의 장단점

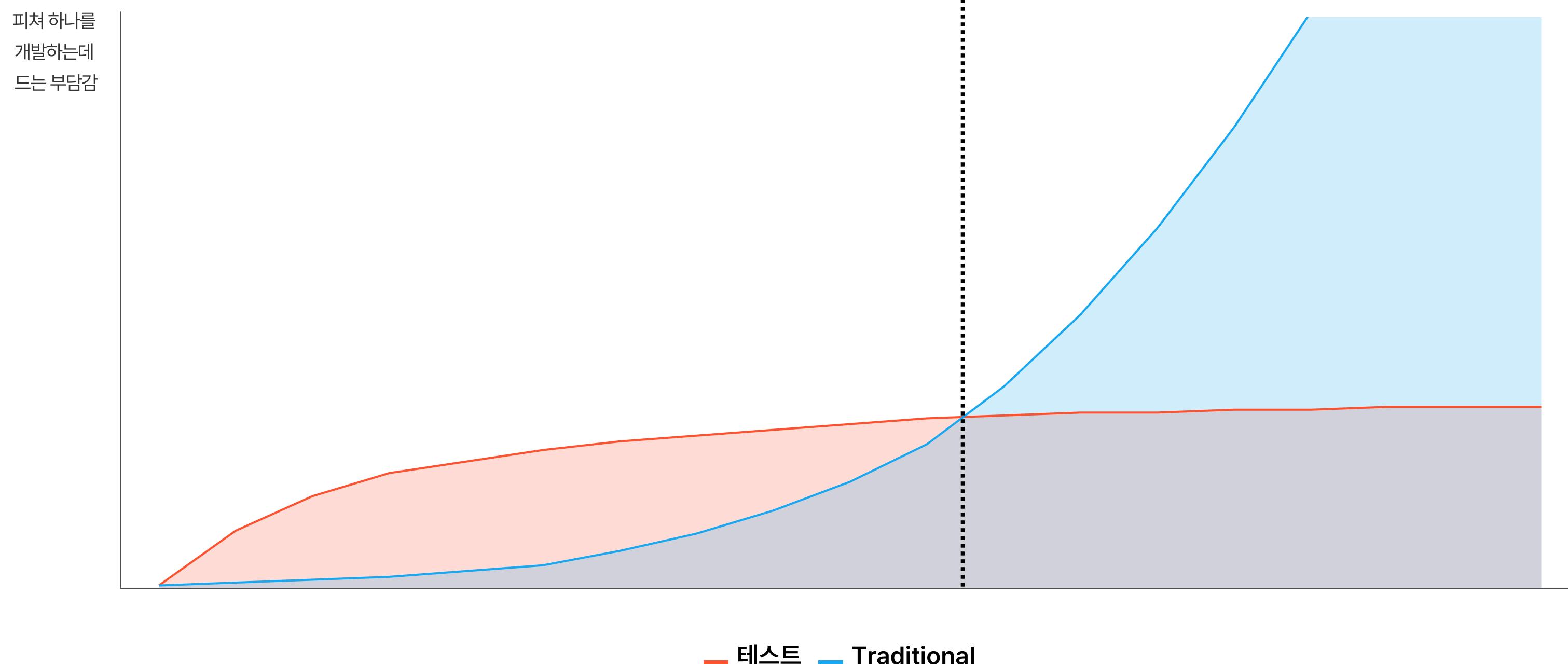
장점 2. 장기적인 관점에서 개발 비용 감소



2. TDD

테스트 주도 개발의 장단점

단점 1. 초기 개발 비용



2. TDD

테스트 주도 개발의 장단점

단점 2. 난이도

3. 개발자의 고민

01. 고민 1

무의미한 테스트

02. 고민 2

테스트가 불가한 코드

03. 고민 3

쉽게 깨지는 테스트

3. 개발자의 고민

고민 1. 무의미한 테스트 (Recap)

RED 1. 깨지는 테스트를 먼저 작성한다.

GREEN 2. 깨지는 테스트를 성공시킨다.

BLUE 3. 리팩토링한다.

```
@Service  
@RequiredArgsConstructor  
public class PostService {  
  
    private final PostRepository postRepository;  
  
    public PostEntity getPostById(Long id) {  
        return postRepository.findById(id)  
            .orElseThrow(() -> new ResourceNotFoundException("Post", id));  
    }  
}
```

```
@Autowired  
private PostService postService;  
  
@Autowired  
private UserRepository userRepository;  
  
@Autowired  
private PostRepository postRepository;  
  
@Test  
void 게시물을_아이디로_조회_할_수_있다(){  
    // given  
    UserEntity userEntity = userRepository.save(new UserEntity());  
    PostEntity postEntity = new PostEntity();  
    postEntity.setWriter(userEntity);  
    postEntity.setContent("helloworld");  
    postEntity.setCreatedAt(Clock.systemUTC().millis());  
    postEntity.setModifiedAt(Clock.systemUTC().millis());  
    postEntity = postRepository.save(postEntity);  
  
    // when  
    PostEntity result = postService.getPostById(postEntity.getId());  
  
    // then  
    assertThat(result.getContent()).isEqualTo("helloworld");  
}
```

3. 개발자의 고민

고민 1. 무의미한 테스트 (Recap)

솔직히 이런 건 Jpa 구현체측에서 알아서 잘 테스트 해줬을겁니다.

```
@Service  
@RequiredArgsConstructor  
public class PostService {  
  
    private final PostRepository postRepository;  
  
    public PostEntity getPostById(Long id) {  
        return postRepository.findById(id)  
            .orElseThrow(() -> new ResourceNotFoundException("Post", id));  
    }  
}
```

```
@Autowired  
private PostService postService;  
  
@Autowired  
private UserRepository userRepository;  
  
@Autowired  
private PostRepository postRepository;  
  
@Test  
void 게시물을_아이디로_조회_할_수_있다(){  
    // given  
    UserEntity userEntity = userRepository.save(new UserEntity());  
    PostEntity postEntity = new PostEntity();  
    postEntity.setWriter(userEntity);  
    postEntity.setContent("helloworld");  
    postEntity.setCreatedAt(Clock.systemUTC().millis());  
    postEntity.setModifiedAt(Clock.systemUTC().millis());  
    postEntity = postRepository.save(postEntity);  
  
    // when  
    PostEntity result = postService.getPostById(postEntity.getId());  
  
    // then  
    assertThat(result.getContent()).isEqualTo("helloworld");  
}
```

3. 개발자의 고민

고민 1. 무의미한 테스트 (Recap)

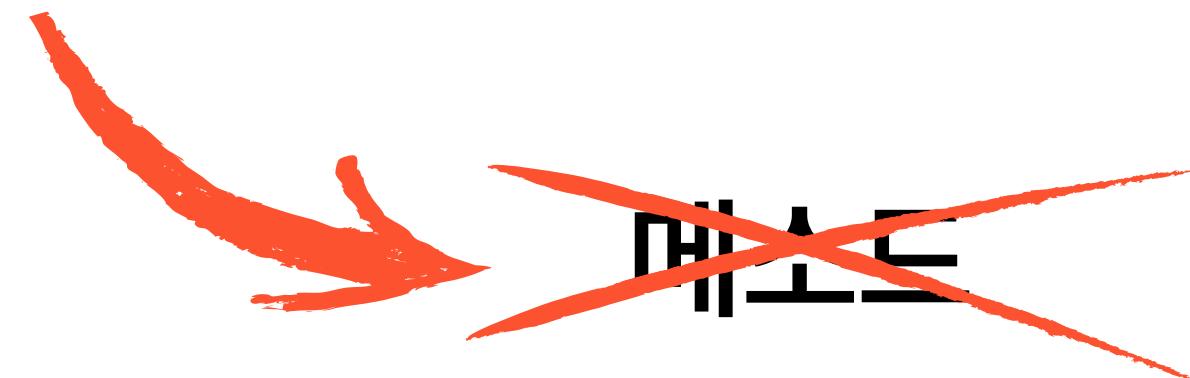
장점 1. 깨지는 테스트를 먼저 작성해야하기 때문에, 인터페이스를 먼저 만드는 것이 강제된다.



3. 개발자의 고민

고민 1. 무의미한 테스트 (Recap)

장점 1. 깨지는 테스트를 먼저 작성해야하기 때문에, 인터페이스를 먼저 만드는 것이 강제된다.



3. 개발자의 고민

고민 2. 느리고 쉽게 깨지는 테스트

이 테스트는 괜찮은 테스트일까요?

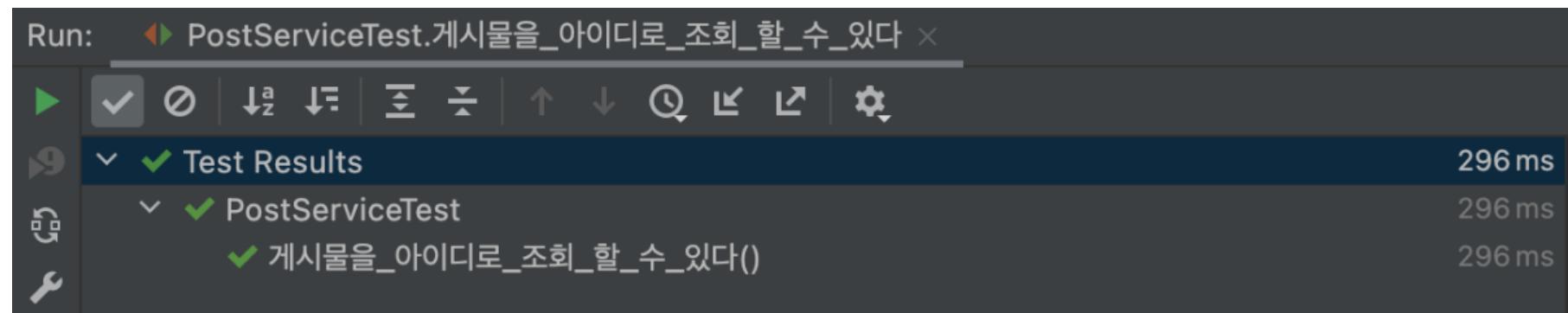
```
@Service  
@RequiredArgsConstructor  
public class PostService {  
  
    private final PostRepository postRepository;  
  
    public PostEntity getPostById(Long id) {  
        return postRepository.findById(id)  
            .orElseThrow(() -> new ResourceNotFoundException("Post", id));  
    }  
}
```

```
@Autowired  
private PostService postService;  
  
@Autowired  
private UserRepository userRepository;  
  
@Autowired  
private PostRepository postRepository;  
  
@Test  
void 게시물을_아이디로_조회_할_수_있다(){  
    // given  
    UserEntity userEntity = userRepository.save(new UserEntity());  
    PostEntity postEntity = new PostEntity();  
    postEntity.setWriter(userEntity);  
    postEntity.setContent("helloworld");  
    postEntity.setCreatedAt(Clock.systemUTC().millis());  
    postEntity.setModifiedAt(Clock.systemUTC().millis());  
    postEntity = postRepository.save(postEntity);  
  
    // when  
    PostEntity result = postService.getPostById(postEntity.getId());  
  
    // then  
    assertThat(result.getContent()).isEqualTo("helloworld");  
}
```

3. 개발자의 고민

고민 2. 느리고 쉽게 깨지는 테스트

이 테스트는 괜찮은 테스트일까요?



```
@Service
@RequiredArgsConstructor
public class PostService {

    private final PostRepository postRepository;

    public PostEntity getPostById(Long id) {
        return postRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Post", id));
    }
}
```

```
@Autowired
private PostService postService;

@Autowired
private UserRepository userRepository;

@Autowired
private PostRepository postRepository;

@Test
void 게시물을_아이디로_조회_할_수_있다(){
    // given
    UserEntity userEntity = userRepository.save(new UserEntity());
    PostEntity postEntity = new PostEntity();
    postEntity.setWriter(userEntity);
    postEntity.setContent("helloworld");
    postEntity.setCreatedAt(Clock.systemUTC().millis());
    postEntity.setModifiedAt(Clock.systemUTC().millis());
    postEntity = postRepository.save(postEntity);

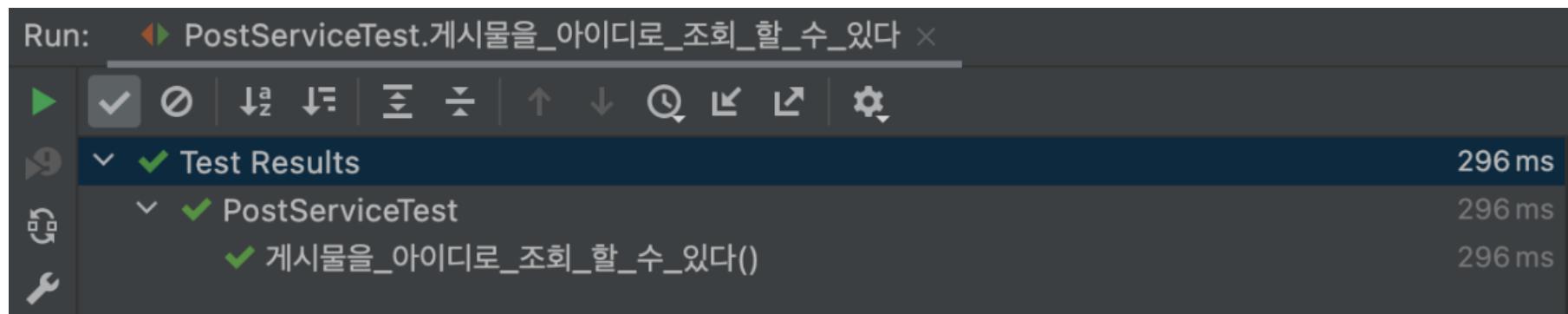
    // when
    PostEntity result = postService.getPostById(postEntity.getId());

    // then
    assertThat(result.getContent()).isEqualTo("helloworld");
}
```

3. 개발자의 고민

고민 2. 느리고 쉽게 깨지는 테스트

이 테스트는 괜찮은 테스트일까요?



```
@Service  
@RequiredArgsConstructor  
public class PostService {  
  
    private final PostRepository postRepository;  
  
    public PostEntity getPostById(Long id) {  
        return postRepository.findById(id)  
            .orElseThrow(() -> new ResourceNotFoundException("Post", id));  
    }  
}
```

```
@Autowired  
private PostService postService;  
  
@Autowired  
private UserRepository userRepository;  
  
@Autowired  
private PostRepository postRepository;  
  
@Test  
void 게시물을_아이디로_조회_할_수_있다(){  
    // given  
    UserEntity userEntity = userRepository.save(new UserEntity());  
    PostEntity postEntity = new PostEntity();  
    postEntity.setWriter(userEntity);  
    postEntity.setContent("helloworld");  
    postEntity.setCreatedAt(Clock.systemUTC().millis());  
    postEntity.setModifiedAt(Clock.systemUTC().millis());  
    postEntity = postRepository.save(postEntity);  
  
    // when  
    PostEntity result = postService.getPostById(postEntity.getId());  
  
    // then  
    assertThat(result.getContent()).isEqualTo("helloworld");  
}
```

3. 개발자의 고민

고민 3. 테스트가 불가한 코드

```
@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;

    @Transactional
    public void login(long id) {
        UserEntity userEntity = userRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("User", id));
        userEntity.setLastLoginAt(Clock.systemUTC().millis());
    }
}
```

3. 개발자의 고민

고민 3. 테스트가 불가한 코드

```
└─ @SpringBootTest
   └─ @TestPropertySource("classpath:test-application.properties")
    ► class UserServiceTest {

      @Autowired
      private UserService userService;

      @Autowired
      private UserRepository userRepository;

      @Test
      void 사용자는_login_하면_마지막_로그인_시간이_현재_시간으로_기록된다() {
        // given
        long userId = 1;

        // when
        userService.login(userId);

        // then
        UserEntity userEntity = userRepository.findById(userId).get();
        assertThat(userEntity.getLastLoginAt()).isEqualTo(현재시간);
      }
    }
```

3. 개발자의 고민

고민 3. 테스트가 불가한 코드

```
└─ @SpringBootTest
   └─ @TestPropertySource("classpath:test-application.properties")
    ► class UserServiceTest {

      @Autowired
      private UserService userService;

      @Autowired
      private UserRepository userRepository;

      @Test
      void 사용자는_login_하면_마지막_로그인_시간이_현재_시간으로_기록된다() {
        // given
        long userId = 1;

        // when
        userService.login(userId);

        // then
        UserEntity userEntity = userRepository.findById(userId).get();
        assertThat(userEntity.getLastLoginAt()).isEqualTo(현재시간);
      }
    }
```



3. 개발자의 고민

고민을 통한 개선

1. 무의미한 테스트
2. 느리고 쉽게 깨지는 테스트
3. 테스트가 불가한 코드

→ 테스트는 좋은 설계를 유도합니다.

정리

- 테스트가 무엇인지 살펴봤습니다.
- TDD가 무엇인지 살펴봤습니다.
- 테스트를 작성하는 개발자들의 고민을 살펴봤습니다
- 그리고 그런 고민 중 일부는, 사실 테스트가 보내는 신호임을 같이 살펴봤습니다

RETURN;