



TDD 장점

우리가 TDD를 해야하는 이유

요구사항의 명확화

테스트 코드를 작성을 위해서는,

무엇을 테스트할지 결정해야 하며, 프로그래밍의 목적을 명확히 한다.

기대 동작을 먼저 테스트로 표현

내부 구현을 먼저 생각하는 것이 아니라, 기대 동작에 대한 **외형만 갖춘 인터페이스**를 먼저 생각하고 구현한다.

동작하는 문서

테스트가 spec 문서의 역할을 한다

무엇을 해야 하는지 명확히 보여주는 **실행 가능한 사양서**의 역할을 한다.
기대하는 동작이 **구체적으로 명시**되어 있다.

코드와의 동기화

spec 변경으로 코드 변경 시 테스트 또한 같이 수정되어야 한다.

리팩토링을 통한 점진적 설계

요구사항 변화에 유연하게 대처

요구사항은 언제든지 변경된다. TDD 에서는 테스트 안전망을 통해 설계를 변화시키기 좋다

작은 단위의 설계 변경

작은 요구사항을 테스트로 작성하고, 이에 맞춰 설계 변경을 점진적으로 진행한다.
요구사항이 추가될 때마다 설계를 다시 고려하며 YAGNI 원칙을 지켜 나갈 수 있다.

테스트하기 쉽도록

테스트가 가능한 구조로 시작한다

Test 를 먼저 고려하여 구현 시 테스트하기 어려운 부분을 확인하며 좋은 설계로 유도한다.

응집도 높은 모듈로 유도

책임이 명확치 않은 테스트를 작성하려면 무엇을 테스트하는지 모호하고 복잡해진다.
테스트 가능성이 하나의 책임에 집중하는 모듈을 설계할 수 있도록 유도한다.

낮은 결합도의 모듈

테스트 작성시 낮은 결합도를 갖는 인터페이스 사용, DI 등을 유도하여
테스트 대상을 의존성과 분리시킬 수 있다.

버그율 감소

버그 조기에 발견

테스트 시점이 구현 시점과 가깝다. 오류를 조기에 발견하고 수정할 수 있도록 해준다.

재현 가능한 버그

작성한 테스트 코드를 통해서, 나중에 같은 버그가 생겨도 즉시 감지 가능하고
버그 경로 추적이 가능해진다.

회귀 방지

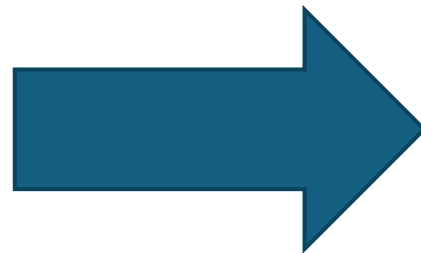
기존의 테스트가 실패한다면, 코드 변경이 기존 동작을 깨뜨렸다는 것을 파악할 수 있게 해준다. 동작이 유지되었음을 신뢰할 수 있게 된다.

Code Coverage 확보

TDD를 통해 약 80% 이상의 코드 커버리지 확보 가능하다

Code Coverage

테스트 프로그램을 돌렸을 때,
코드 라인 별, 몇 % 정도 테스트가
되었는가 ?



```
        lstrcat(arExt, ",");
        if (strstr(arExt, ext+1) != 0) {
            ID=i;
            break;
        }
    }

    Ae.SetParser(ID);
    Option.SetStyleColor(Ae.GetParser());
} = end SelectParser =

void Relayout()
{
    RECT crt;
    int t,s,f,o,w,ch;
    int fh;

    GetClientRect(g_hFrameWnd, &crt);
    GetChildSize(t,s,f,o,w,ch);

    if (Option.bShowToolBar) {
        SendMessage(hToolBar, TB_AUTOSIZE, 0, 0);
        ShowWindow(hToolBar, SW_SHOW);
    } else {
        ShowWindow(hToolBar, SW_HIDE);
    }

    if (Option.bShowStatus) {
        SendMessage(hStatus, WM_SIZE, 0, 0);
        ShowWindow(hStatus, SW_SHOW);
        SetStatusPart();
    } else {
        ShowWindow(hStatus, SW_HIDE);
    }

    if (Option.bShowFileTab) {
        MoveWindow(hTabFrame, 0, t, crt.right, f, TRUE);
        ShowWindow(hTabFrame, SW_SHOW);
    } else {
```

TDD를 하면 프로그래밍이 재미있어진다.

작은 코드를 구현하고, PASS도 눈으로 보고 재미있고 성취감도 느낀다.
그리고 PASS들을 보면, 불안감이 줄어들면서 자신감이 더 생긴다.

코딩이 재밌어지고 심리적 안정감을 얻는다.

Chapter5

TDD 단점

TDD에 부정적인 입장.

테스트 자체는 중요하지만 무리한 Unit Test 작성에 걸리는 비용이 클 수 있다

이는 두 가지로 나눌 수 있다.

1. Unit Test 만드는데 어려운 경우
2. 작은 Unit Test 가 너무 많은 경우

Unit Test 만들기 어려운 모듈이 있다.

외부 의존성들로 인해 TDD 가 어려운 경우가 있다

GUI, 병렬 프로그래밍, 네트워크 관련 프로그래밍, 외부 API 등등

TDD를 위해 Unit Test를 억지로 만드는데, 그 비용이 너무 크다.

반론의견

Unit Test 노하우가 쌓이면,
그 비용은 점차 줄어들 수 있다.

Unit Test가 너무 많다.

Baby Step 으로 이뤄진 Unit Test가 너무 많다.

Unit Test 만드는 시간이 아깝다고 느껴진다.

변경으로 인한, Unit Test 유지보수 시간이 아깝다.

3,000개 Unit Test 중, Production Code가 크게 변경되어
500개 Unit Test에서 Fail이 발생할 경우, 500개 Unit Test 유지보수 해주어야 한다.

반론의견

한 프로젝트가 3,000 개의 Unit Test를 보유했다고 한 경우,
객관적으로 품질을 위한 안전 장치를 가졌다고 할 수 있다.
→ 따라서 유지보수 시간을 들일만하다고 생각할 수 있다.

TDD 단점이면서, 동시에 장점

TDD 결과로 수 많은 Unit Test가 만들어진다

개발 비용이 많이 더 투입이 된다.

대신, 품질에 대한 안정성을 더 확보할 수 있다.

매우 높은 품질 & 신뢰성이 요구되는 대규모 프로젝트에는
TDD의 단점보다 장점이 더 돋보인다.

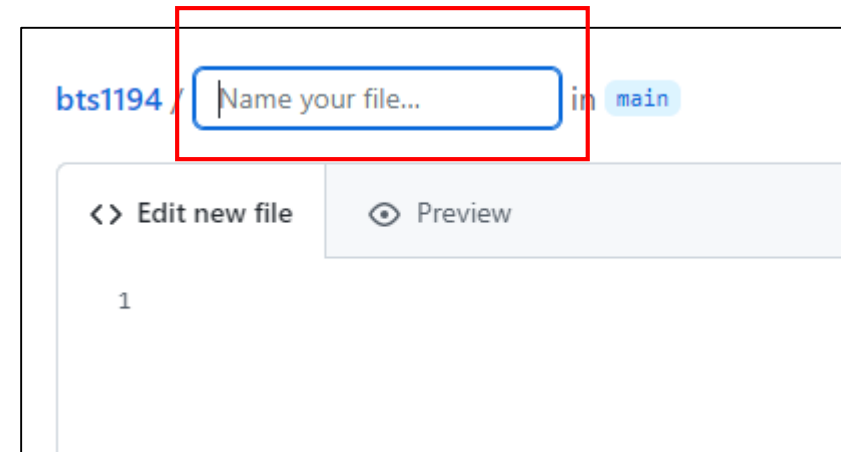
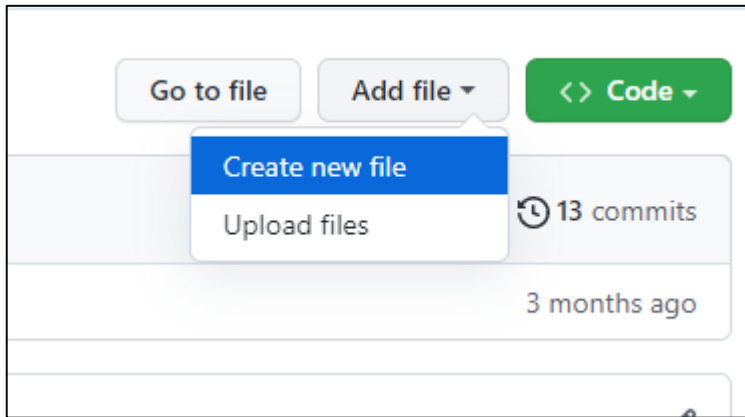


PR Template

PR 작성자를 위한 Check List 만들기

PR Template 작성해보기

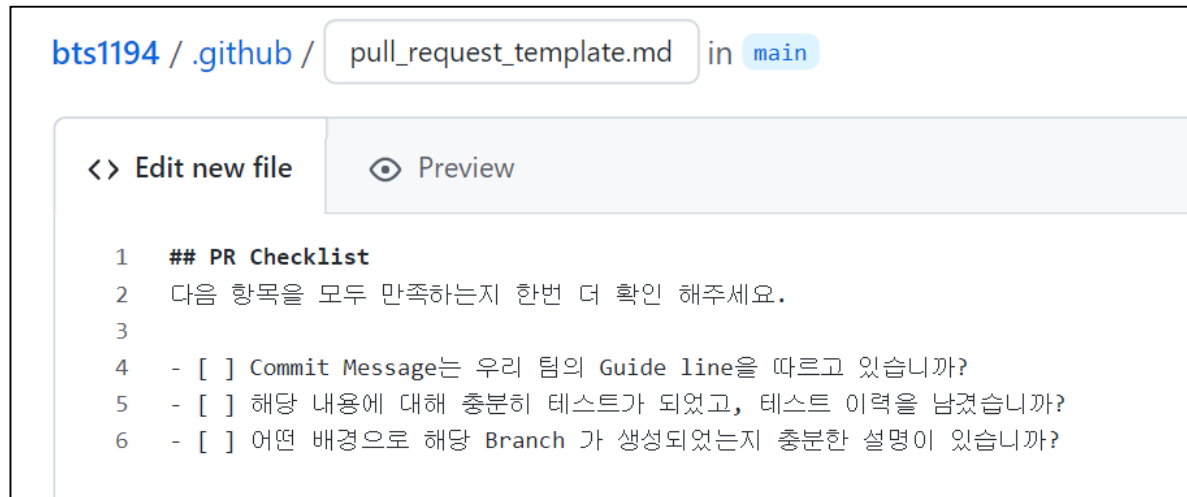
- PR을 작성할 때,



`.github/pull_request_template.md`
이름으로 파일명 입력

PR Template 내용 기입

- 체크리스트 항목 내용 기입



The screenshot shows a GitHub interface for editing a file named `pull_request_template.md` in the `main` branch of the repository `bts1194 / .github /`. The interface has two tabs: `<> Edit new file` and `Preview`. The `Edit new file` tab is active, displaying the following content:


```
1  ## PR Checklist
2  다음 항목을 모두 만족하는지 한번 더 확인 해주세요.
3
4  - [ ] Commit Message는 우리 팀의 Guide line을 따르고 있습니까?
5  - [ ] 해당 내용에 대해 충분히 테스트가 되었고, 테스트 이력을 남겼습니까?
6  - [ ] 어떤 배경으로 해당 Branch 가 생성되었는지 충분한 설명이 있습니까?
```



PR 작성시

- Template 내용을 기반으로 내용을 추가하여 PR을 작성한다.
- 콤보박스 내용은
[] 부분에서
[x] 로 수정하면 된다.

Open a pull request







Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

 base: main ← compare: test ✓ Able to merge. These branches can be automatically merged.

 Update README.md

Write

Preview

H B I      @  ↶

PR 요청
이런 내용으로 PR 요청합니다.

PR 세부 사항
이런 세부 사항이 있습니다.

PR Checklist
다음 항목을 모두 만족하는지 한번 더 확인 해주세요.

☒ [x] Commit Message는 우리 팀의 Guide line을 따르고 있습니까?

☒ [x] 해당 내용에 대해 충분히 테스트가 되었고, 테스트 이력을 남겼습니까?

☒ [x] 어떤 배경으로 해당 Branch 가 생성되었는지 충분한 설명이 있습니까?

Attach files by dragging & dropping, selecting or pasting them.

Create pull request



코드리뷰 : 마음가짐

더 나은 코드리뷰 문화를 위한 마음가짐

코드 리뷰이 (PR 요청자)

상대방에게는 추가 업무가 될 수 있기에
추가적인 업무 부하를 주지 않도록, 준비를 철저하게 하고 PR을 요청한다.

리뷰를 보며, 마음의 상처를 덜 받는다.

코드 리뷰어

상대가 마음의 상처를 입지 않도록, 정중한 표현을 사용하도록 노력한다.
피드백을 통해 학습 기회를 제공하고 코드의 품질을 향상시킨다

팀원의 코멘트에 상처를 덜 받자

극단적인 마인드를 제어하자

예시 1) 감히 (?) : 직급도, 나이도 어린 것이.. 감히 나에게 이런 지적을?

예시 2) 코멘트 주셔서 감사합니다. 아이고, 네네.. 무조건 맞습니다.

코멘트는 커뮤니케이션의 시작이다

코멘트에 의문이 있다면, 충분히 분석 / 조사를 해본 후 구체적인 내용에 대해 논의를 한다.

코멘트 내용을 받아들이지 않아도 된다

받아들이지 않고 진행하더라도 해당 내용에 대해 충분히 검토한 것이기에, 품질에 도움이 된다.

리뷰어의 부담을 고려

리뷰어들이 이해하기 쉽게 써야 한다. 명확하고 읽기 편한 문체와 전문 용어 사용한다. 맥락을 제공하기 위한 설명을 적는다.

작고 이해하기 쉬운 PR 단위

PR이 너무 크면 리뷰어가 맥락을 파악하기 어렵고, 지연되며 리뷰 품질도 떨어진다

단일 목적을 가지기

- 하나의 PR에는 하나의 목적만 갖도록 한다.
- 리팩토링과 기능 추가는 각각 다른 PR 로 나누는 것이 좋다.

셀프 리뷰를 먼저 하고, 코드 리뷰를 요청한다

이 PR을 보면서 내가 리뷰어라면 어떤 질문을 할지를 예측하고 이것에 대해 개선을 먼저 한다.

그리고 나서 다시 PR을 준비한다.

나에게 추가적인 일인 만큼 상대방에도 추가적인 일이 되므로, PR은 꼼꼼한 대비가 필요하다.

테스트 정보를 제공한다

변경 사항이 실제로 작동하는지 확인할 수 있는 정보를 제공한다.
구체적인 **테스트 방식**과 **테스트 결과**를 명시한다.

코드 리뷰어가 품질에 대한 의심이 없도록 해주어야 한다

Unit Test 결과, 자동화 테스트 통과여부
가능하다면, 더 높은 Level의 테스트 까지
가능하다면, 성능 또는 회귀 위험 체크까지
Self 코드리뷰 결과까지

코드 리뷰어가, 더 Deep한 숨은 버그 발생 요소 &
코드 개선점에 대해 논의할 수 있도록 테스트 정보를 제공한다

배경 설명

어떤 History에 의해서, 코드를 왜 바꿨는지
코드만 봐서는 알 수 없는 배경, 맥락을 명시한다.

변경 내용 요약 / 설명

변경 내용에 대해 내용을 **명시적**으로 적는다.
코드 보면 알 수 있어도, 이해에 도움이 되도록 적는다.

테스트 이력

어떠한 테스트를 했고, 성능 / 품질에 이상이 없음을 알린다.
정적 분석도구 결과 / 코드 커버리지 결과 등

코드리뷰어 – 도움을 위한 Review

코드 리뷰어와 다른 코드 리뷰어까지 모두가 도움이 될 수 있는 Review를 남긴다.

안 좋은 예시)

이 부분을 수정했으면 좋겠습니다.
요즘 이런 식으로 개발 안합니다.(X)

좋은 예시)

이 부분을 수정했으면 좋겠습니다.
이유는 이렇습니다,
고치는 방법은
1. 어떤 방법
2. 어떤 방법
인데 제 개인적으로는 1번을 더 추천합니다. 이러하기 때문입니다.
관련 자료는 이렇고, 검토를 부탁드립니다.

모호한 comment

```
10
11
12 def get_total_cars():
13     total_cars = inventory.filter(lambda x: x.type == Types.CAR)
14     return total_cars
15
```

“comment > 이름을 좀 더 좋게 변경할 수 있을 거 같아요”

무슨 이름이 문제인지 명확하지 않다

- get_total_cars()? total_cars? Types.CAR?

왜 더 좋아야 하는지 설명이 없다

- 네이밍이 어떤 기준에서 어색한 건지 알 수 없음

대안이 없다

- 이게 더 나아 보인다는 제안이 없음

명확하고 건설적인 피드백

```
10
11
12 def get_total_cars():
13     total_cars = inventory.filter(lambda x: x.type == Types.CAR)
14     return total_cars
15
```

“comment >

get_total_cars()는 함수가 자동차의 개수를 리턴하는 것으로 오해될 수 있어요.
하지만 실제로는 필터링된 자동차 목록(list)을 반환하네요.

filter_cars()나 get_car_items() 같은 이름을 고려해보시는 건 어떨까요?”

리뷰를 남길 때, 다음 항목을 재검토 해보자

- 코드 리뷰어에게, 도움이 되는 내용인지 검토해본다
- 부정적인 지적 / 비난으로 들릴 수 있을지 생각해본다
- 모든 팀원들이 내가 작성한 Review처럼 남길 때, 긍정적인 코드 리뷰 문화가 만들어질 수 있을지 생각해본다
- 변경 코드가 팀의 표준에 맞는지, 기술 부채가 되지는 않을지 검토한다

[참고자료]

- 구글의 코드리뷰 문화

- <https://m.post.naver.com/viewer/postView.naver?volumeNo=30978428&memberNo=36733075>
- https://www.hanbit.co.kr/channel/view.html?cate_cd=&cmscode=CMS3858769941

- 카카오 코드리뷰

- <https://tech.kakao.com/2022/03/17/2022-newkrew-onboarding-codereview/>

- 구글 코드 리뷰 가이드

- 영문 : <https://google.github.io/eng-practices/>
- 한글번역 : <https://soojin.ro/review/>