소프트웨어 디자인 패턴_애자일, 워터폴

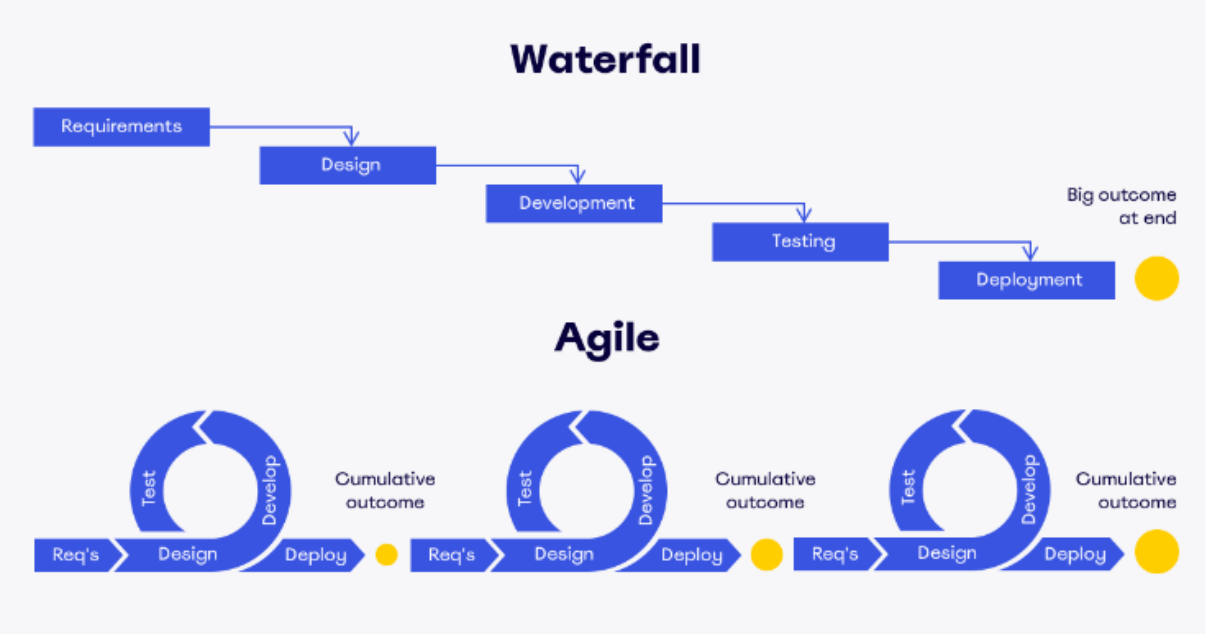
애자일 방법론

> **애자일 소프트웨어 개발 선언**
>
> 공정과 도구보다 개인과 상호작용을
> 포괄적인 문서보다 작동하는 소프트웨어를
> 계약 협상보다 고객과의 협력을
> 계획을 따르기보다 변화에 대응하기를

💡 애자일 방법론

애자일 방법론(Agile Methodology)은 소프트웨어 개발 프로세스에서 유연하고 신속하게 대응하기 위해 만들어진 접근 방식입니다. 이는 전통적인 개발 방법론의 장기 계획과 대규모 문서 작업 대신, 빠르게 변화하는 요구 사항에 유연하게 대응하고, 팀 간의 소통과 협업을 중시하는 것이 특징

워터폴 방식과 애자일 방식 비교



애자일과 워터폴의 장단점 비교

	Agile	Water fall
장점	개발과정이 빠르고 유연	팀 규모에 상관없이 따르기가 쉽다
	짧고 반복적인 스프린트로 구성 -> 빠르게 결함 식별 및 수정가능	개발주기가 정해져 있기 때문에 새로운 프로젝트를 안정적으로 시작 가능
	소규모 팀들이 여러과제를 각각 할당받아 처리 가능	요구사항이 정의되어 있기 때문에 실행하기가 수월하며 목표를 자주 변경하지 않아도 된다.
	개발과정중에 신속하게 제품 변경 가능 (짧은 반복과정 거치기 때문)	필요한 예산과 자원이 초기에 확정되기 때문에 예상결과와, 리스크를 통제하기가 쉽다.
단점	빠른 반복 작업에 익숙한 숙련된 사람이 필요	개발속도가 느리며 유연성이 떨어진다
	수많은 변경사항이 있을 수 있으므로 번거로움 발생 가능	개발 요구사항이 초기에 정해지기 때문에 변경이 자유롭지 못하다
적합한 조직	고품질의 결과물과 지속적 개선에 초점을 맞춘 조직	순차적인 프로젝트 타임라인 사전 확정 예산이 필요한 팀.
	크고 복잡한 회사들이 프로세스를 간소화함으로써 변화에 신속대응 하고자 할때 (ex. IBM, AT&T,마이크로소프트)	개발상의 변경이나 리스크에 덜 민감한 팀
	결과물에 대해 빠른 피드백이 필요한 팀	요구사항이 간단한 팀, 타임라인이 긴 프로젝트

▼ 특징

- 반복적이고 점진적인 개발 :
  반복적이고 점진적인 개발 : 보통 1~4주 정도의 기간으로 설정되며, 이 기간 동안 사용할 수 있는 제품의 일부를 완성 후 피드백을 주고 받는다.
- 고객과의 지속적인 소통 :
  고객의 요구 사항을 지속적으로 수집하고 반영하여 수용하기 위해 개발자들은 정기적인 회의와 리뷰를 통해 과업을 완수한다.
- 팀의 자율성과 협력 :
  팀원들은 자율적으로 작업을 계획하고 진행하며, 이를 통해 높은 책임감과 동기부여를 유지한다.
  팀 내의 역할은 유연하게 나누어져 있으며, 모든 팀원이 서로 협력하여 문제를 해결

한다.

- 적응성과 유연성 :
  초기 계획에 얽매이지 않고, 프로젝트 진행 중 발생하는 변화를 유연하게 수용한다.
  우선순위를 지속적으로 조정하여 가장 중요한 작업에 집중한다.

- 지속적인 개선 :
  각 스프린트가 끝날 때마다 회고를 통해 팀의 작업 방식과 프로세스를 개선해간다.
  이를 통해 팀의 효율성과 제품의 품질을 지속적으로 향상시킨다.

▼ 예시

- **Scrum**

제품의 우선순위에서 어디까지 작업할지 팀과 조율하여 단기 목표(스프린트)를 세운다.

스프린트를 진행하는 동안 매일 모든 개발 팀원이 참여하는 일일 Scrum 회의를 가진다.

스프린트가 종료할 때마다, 스프린트 리뷰 미팅을 통해 만들어진 제품을 학습하고 이해한다.

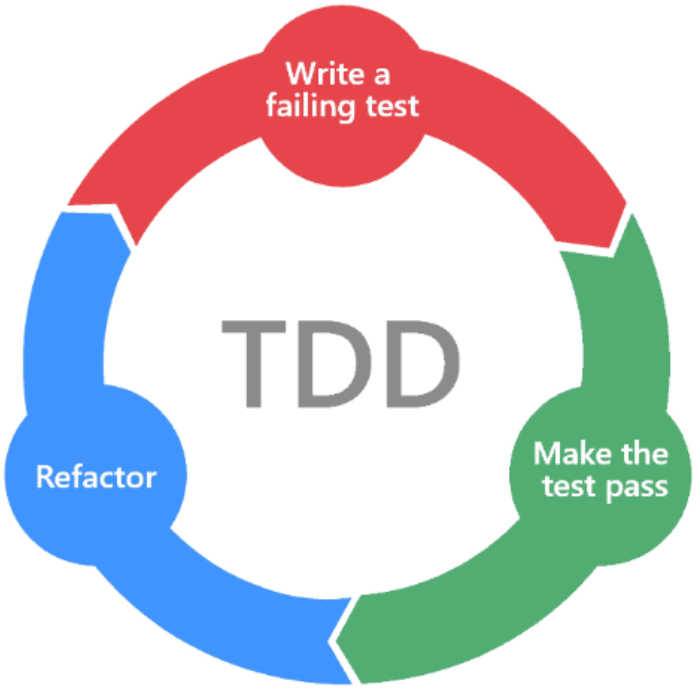
제품의 학습과 이해가 끝나면, 스프린트 회고를 통해 팀의 개발 프로세스에 대한 개선의 시간을 갖는다.

스프린트 기간 중 다음 스프린트를 준비하기 위해 PO와 필요 인원이 모여 백로그를 준비하는 시간을 갖는다.

- **Test Driven Development (TDD)**

  작은 단위의 테스트 케이스를 작성하고 이를 통과하는 코드를 추가하는 단계를 반복하여 구현하는 디자인 패턴

TDD 프로세스



TDD 의 장단점

장점

- 재설계 시간 단축

  → 높은 품질의 소프트웨어를 보장
- 철저한 기능별 모듈화

  → 종속성이 낮은 코드
- 추가 요구사항 반영이 쉬움
- 디버깅 시간 단축

  → 에러 및 버그가 적음

단점

- 낮은 생산성

  → 테스트코드를 추가적으로 개발해야해서
- 초기 비용 증가

  → 초기 세팅 비용이 들고 숙련까지 오랜 시간이 걸림
- 기한 준수가 까다로움

  → 생산성이 낮기 때문에