

development Method

개발 방법론

Sample Footer Text

7/27/2023

1

소개



방법론 소개

TDD/BDD/ATDD

Extreme programming (XP)

Extreme programming (XP)는 프로그래밍 개발 방법론 중 하나이며, 주요 목적은 소프트웨어 품질을 향상시키는 게 목적이며, 애자일 방법론 중 하나이다. 큰 규모의 프로젝트보다 작은 규모의 프로젝트에서 사용이 적합하다.

XP에 대한 정의한 "**켄트 벡(Kent Beck)**" XP 방법론의 제일 중요한 부분은 가치와 원칙에 중심을 두고 있으며, 이는 개개인에게 역량에 많이 달려 있다. 이러한 부분 때문에 프로토타입의 애플리케이션을 데모 수준으로 이끌어서 빠르게 고객에 시연이 가능하다.

XP개발 방법의 주요 목적은 "**고객이 원하는 수준의 소프트웨어를 빠른 시일내에 전달**".

1. 의사소통
2. 단순성
3. 피드백
4. 용기
5. 존중

Extreme programming (XP)

익스트림 프로그래밍은 1990년도에 많은 개발자들이 도입하였다. 하지만, XP 여러가지 비판 혹은 비평에 직면하게 되었다.

페어 프로그래밍

XP의 기본 중 하나는 동료와 함께 "Pair Review"라는 부분이 전제가 깔려 있다. 하지만, 이 부분은 많은 문화권에서 어려운 부분 중 하나였으며, 이로 인하여 XP의 가치관이 올바르게 적용이 되지 않았다.

연속 설계

연속적인 설계를 하기 위해서는 계속 XP프로세스를 반복해야 되는 문제가 있다. 수정을 위해서 많은 시간과 그리고, 패드 백 및 존중의 시간이 발생이 되며, 이를 통해서 감정적인 문제가 발생하는 경우가 있다.

TDD

Test-driven development (TDD) is a [software development process](#) relying on software requirements being converted to [test cases](#) before software is fully developed, and tracking all software development by repeatedly testing the software against all test cases. This is as opposed to software being developed first and test cases created later.

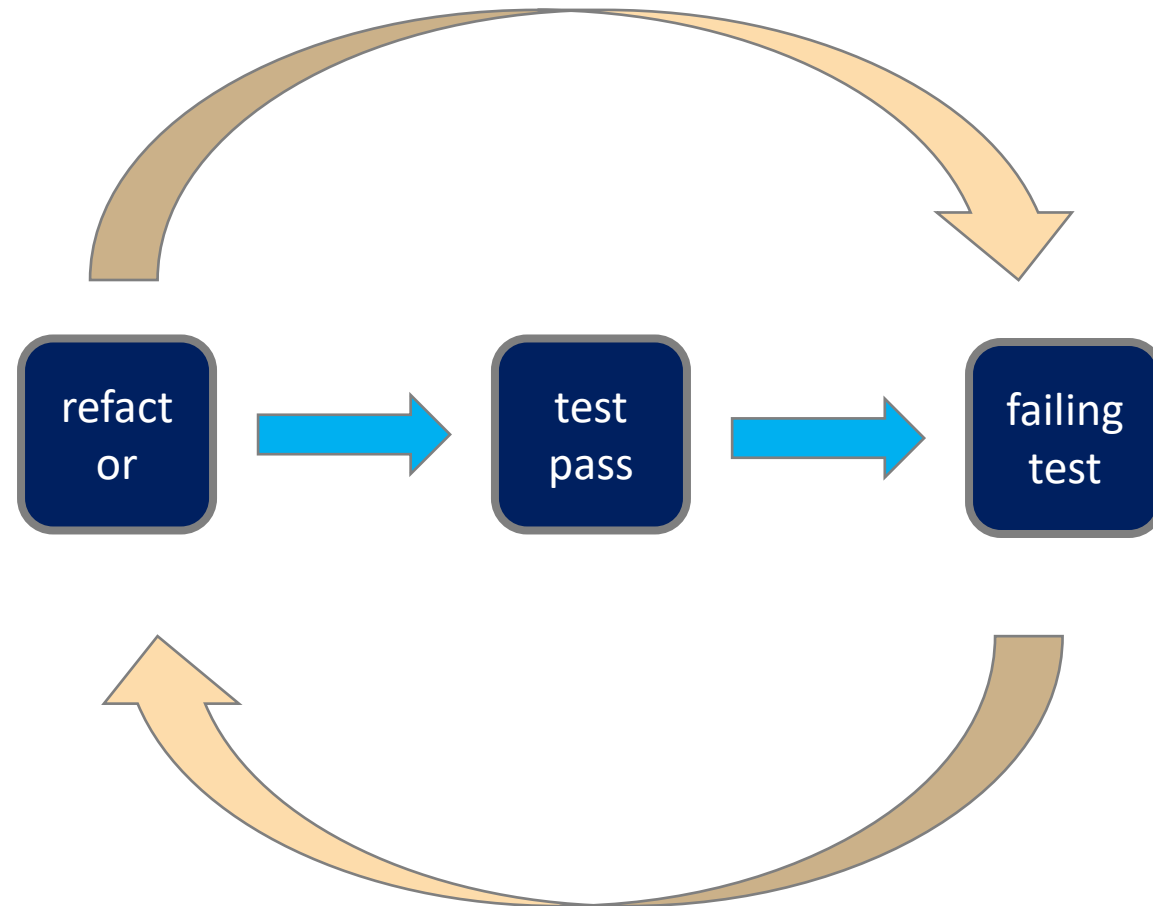
Test-driven development is related to the test-first programming concepts of [extreme programming](#), begun in 1999,^[3] but more recently has created more general interest in its own right.^[4]

TDD

Test-driven development (TDD)는 소프트웨어 개발론의 하나이며, 개발하는 과정에서 테스트를 통해서 빠르게 문제점과 개발 진행 및 통합이 주요 목적이다. 이러한 방법론 기반으로 진행하였을 때 최대 장점은, 개발자가 별도로 품질 및 테스트 검증을 진행하지 않아도 되며, 빠르게 소프트웨어 배포가 가능하다.

TDD는 1999년도에 컨셉이 나왔으며, 실제로 바로 개발환경에 적용되지는 않았다. 오픈소스가 기업이 및 커뮤니티에서 활성화가 되면서, 많은 회사들이 빠르게 TDD기반으로 개발 과정을 도입을 시도 하였다. 최근, 기존 모노리크 서비스 아키텍처에서 하이브리드 및 마이크로 서비스(기능기반)으로 변환이 되면서 도입이 가속화 되었다.

TDD



BDD

Behavior-driven development(BDD)는 **TDD**에서 확장된 개념이다. 기존 개발 프로세스에 DSL를 사용하여 조금 더 유닛 단위의 테스트를 쉽게 할 수 있도록 하였다. DSL(Domain Specific Language)기반으로 문법을 작성하여, 각각 문법을 실행하면서 기능을 테스트 및 실행 할 수 있도록 한다. 이를 통해서 자연스럽게 TDD에 확장하여 개발과정에 생태계를 구축 할 수 있다.

BDD 프레임 워크(BDD Framework)는 많은 개발자들이 프레임워크 기반으로 오랫동안 사용하였으나, 정의에 대해서 완벽하지 않았다. 품질담당/비 기술적 영역(라이프사이클)/비즈니스는 소프트웨어 개발자와 협력하는 프레임워크로 구성하였다. 이러한 내용을 기반으로 "**애자일 사양(Agile specifications)**"를 구성하게 되었다.

BDD는 다음과 같은 목적 및 주제를 가지고 있다.

- Where to start in the process
- What to test and what not to test
- How much to test in one go
- What to call the tests
- How to understand why a test fails

BDD

이 테스트 방식은 최대한 사람이 사용하는 일반 대화형 언어 형태로 작성해야 한다.

Given

테스트를 위해 주어진 상태/조건/환경이다.

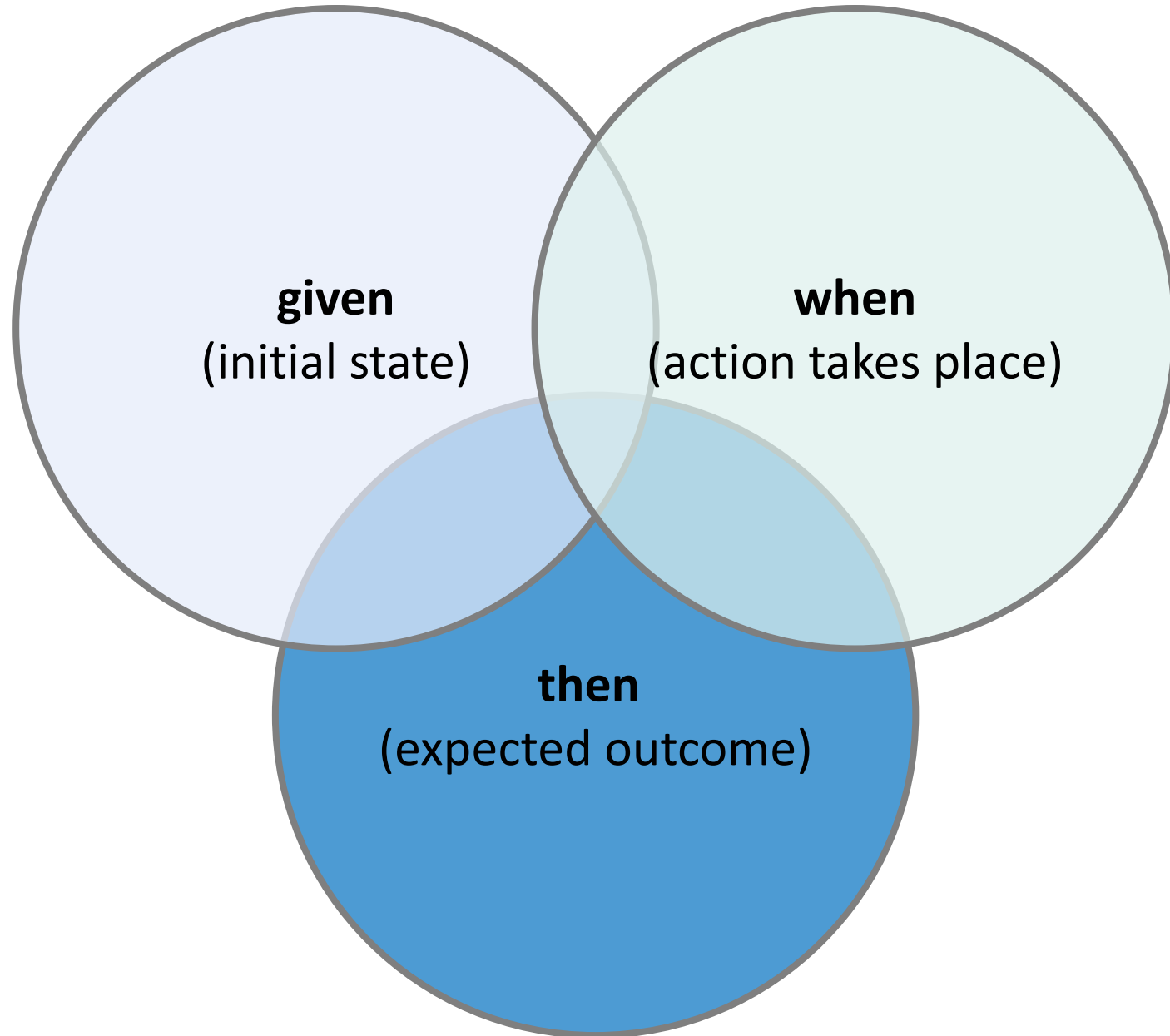
When

테스트를 위해 가해진 상태/조건/환경. 이 부분은 보통 상태 -> 조건 -> 환경 이와 같은 상태를 구성한다.

Then

위의 조건으로 테스트가 완료 후 나온 결과를 처리하는 부분.

BDD



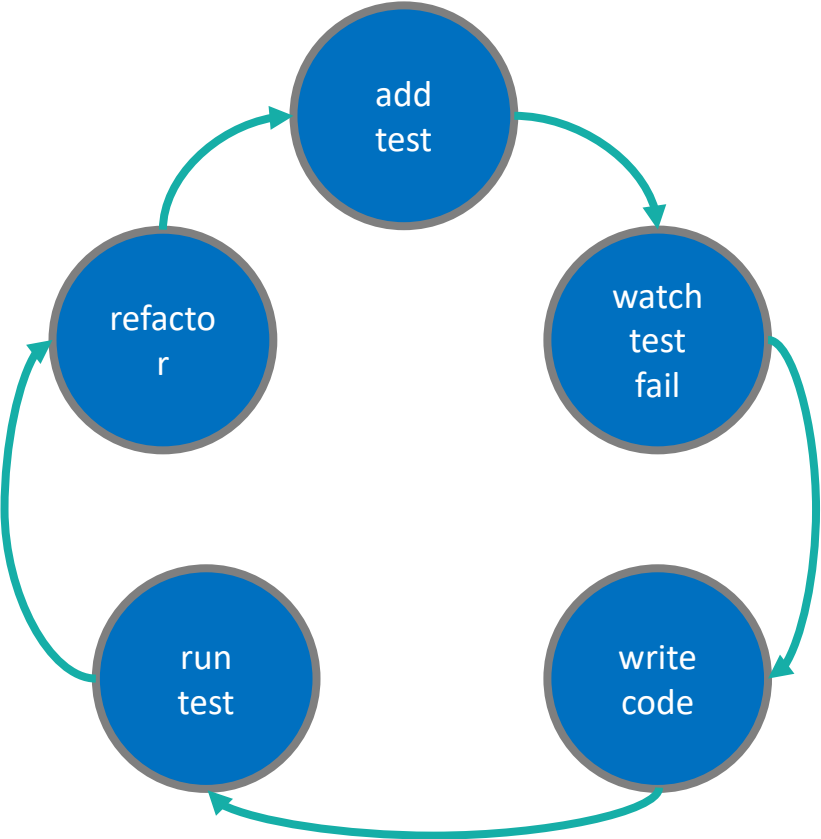
ATDD

Acceptance test-driven development (ATDD)는 BDD+TDD가 통합된 개발 방법론이다. 이 방법론은 고객(사용자), 개발자 그리고 테스터(QA)가 함께 묶여진다. ATDD는 SBE, BDD, EDD 그리고, SDD(Story Test Driven Development)가 같이 어울려진다.

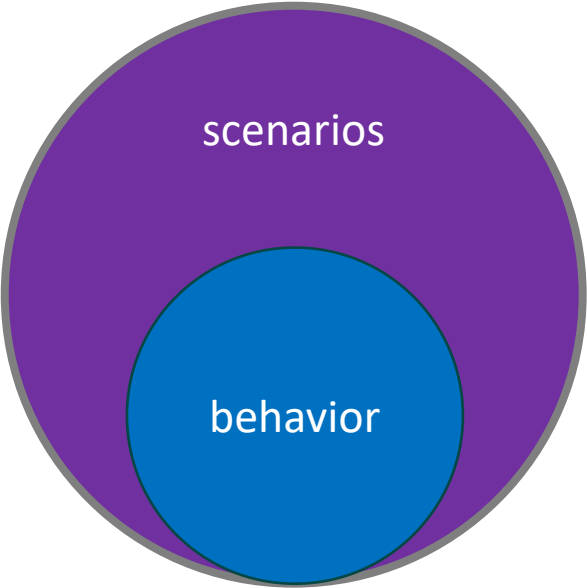
모든 프로세스에 개발자 및 품질 관리자들은 고객의 요구사항(needs)를 파악하고, 여기에 대해서 **Domain Language(DL)**를 통해서 요구사항을 재구성 한다.

ATDD는 TDD와 매우 가까운 관계이다. 이들의 중요한 차이점은 **개발자/품질 관리자/고객의 협업**이 중요하다. ATDD는 개발자가 코딩을 진행하기 전에, 어떠한 방식으로 테스트를 할지 **DL(Domain Lanage)**를 통해서 미리 작성 및 구성한다.

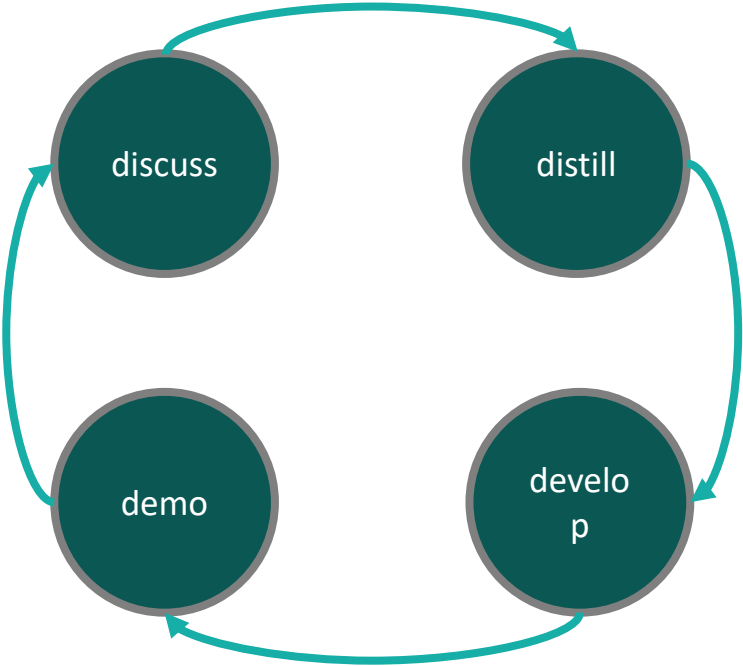
ATDD



TDD



BDD



ATDD

agile

애자일 소프트웨어 방법론은 대략 2001년도, 즉 XP가 정점에 도달하였을 때, 애자일 방법론이 부각되기 시작하였다. 단일 혹은 여러 팀들이 기능적으로 협력하여 고객 혹은 최종 사용자의 요구사항을 찾고 발전 시키는 게 주요 목적이다. 애자일 방법론에 대한 구성 및 방법론은 **스크럼(Scrum)** 및 **칸반(Kanban)**와 같은 소프트웨어 도구 및 방법론이 나오면서 강화 및 발전이 되기 시작하였다.

애자일 방법론은 1956년도에 프로젝트 관리 및 소프트웨어 개발 방법론에 대해서 구상이 되었다. 실제로 구현 및 사용은 1970년도부터 시작이 되었다.

agile RULES

1. 프로세스 및 도구를 통해서 개발자들은 소통을 한다.
2. 문서 작업을 통해서 소프트웨어 개발을 진행 및 수행한다.
3. 계약을 통해서 고객과 협업을 만든다.
4. 계획적으로 변경 내용이 발생하면, 해당 변경 사항에 대해서 대응한다.

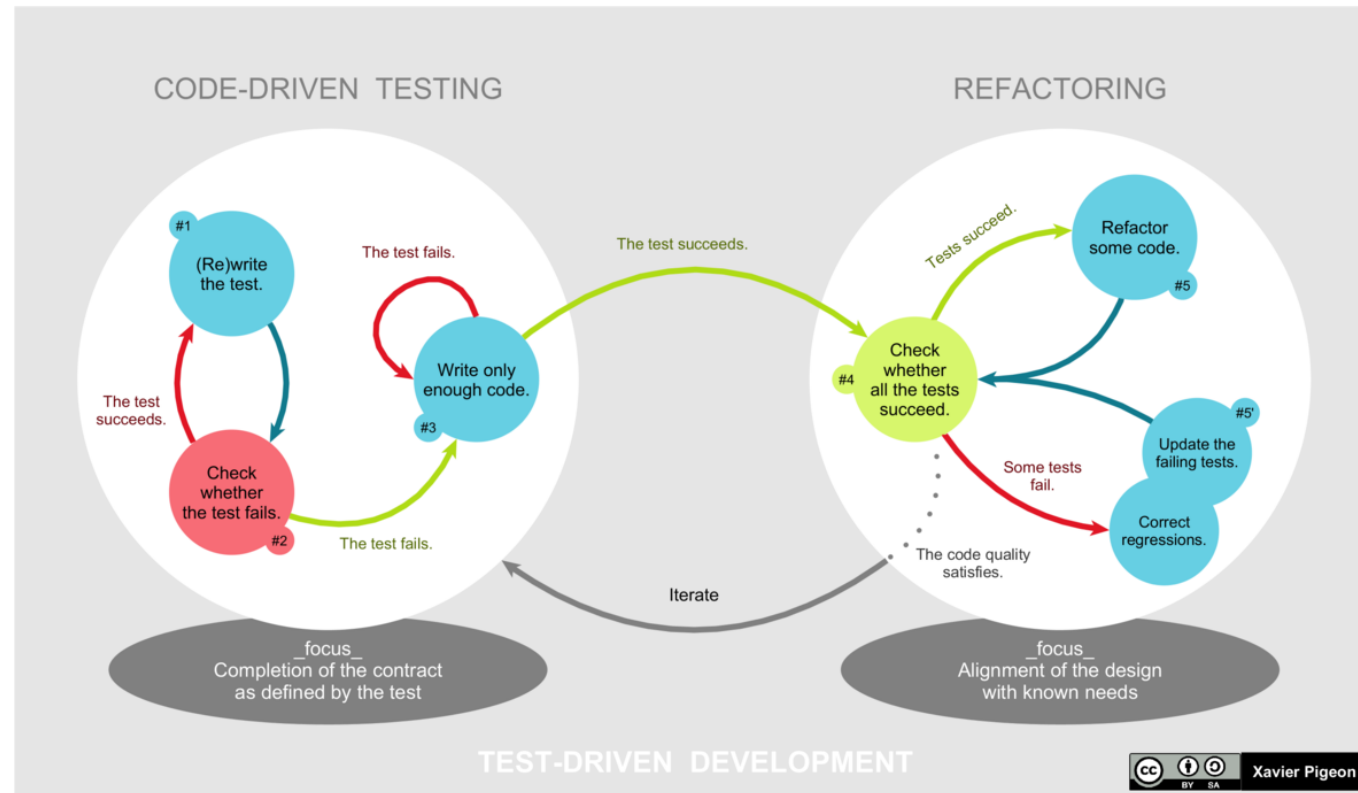
AGILE

- 고객 혹은 사용자가 원하는 요구사항을 지속적으로 제공(가치) 한다.
- 개발 중, 변경 및 변동사항이 발생하여도 반영이 가능하다.
- 변경된 코드가 반영된 소프트웨어를 주 혹은 달 단위로 제공 합니다.
- 개발자 및 업무(영업)과 매우 가까이 협력 및 협업을 진행 합니다.
- 각 개인별로 프로젝트 신뢰도를 높이며, 동기화를 부여하여 더 높은 생산성을 기대.
- 최대한 대면 형태의 커뮤니케이션(온라인 도구 포함)를 통해서 협업을 이끌어 낸다.

AGILE

- 진행 척도는 "소프트웨어 동작"상태를 기준.
- 최대한 개발속도를 일정하게 유지하여, 피로감을 최소화.
- 최신 기술 기반으로 좋은 설계를 가져가도록 노력 한다.
- 완료가 안된 작업은 최대한 단순화 및 처리.
- 좋은 소프트웨어 설계 및 품질은 팀 자체.
- 팀은 지속적으로 좋은 효율성을 찾기 위해서 틈틈이 작은 회의를 통해서 조율.

BDD VS TDD



Advantages and Disadvantages of TDD Summary

| TDD 장점 | | TDD 단점 |
|--------|-------------------------------------|----------------------------------|
| 1 | 더 나은 코드 및 품질 | 시작 초기에, 많은 시간과 자원 투자가 필요 |
| 2 | 버그의 최소화 | 완벽한 해결책은 아님(Not a silver bullet) |
| 3 | 반복적인 테스트 스위트 | 문화적인 변화가 필요 |
| 4 | 기능 코드 기반으로 손쉽게 재구성(refactoring)이 가능 | 유지보수에 많은 시간이 투입 |
| 5 | 문서기반으로 작업 | |
| 6 | 시장 출시 시간 단축 | |
| 7 | 품질 및 비용(cost) 절감 효과 | |

Advantages and Disadvantages of BDD Summary

| BDD Advantages | | BDD 단점 |
|----------------|-------------------------|-------------------|
| 1 | 협업(Collaboration) | 자동화를 위한 시간이 소모 |
| 2 | 모호함의 감소 | 문화적 변화 및 변경이 필요 |
| 3 | 소스코드의 검증 확신 | 코드 유지보수에 대한 시간 증가 |
| 4 | 근본적인 문제 해결(Shift left) | BDD에 전문화 요구 및 필요 |
| 5 | 다수가 책임 및 관리 | |
| 6 | 자동화 | |
| 7 | 실행이 가능한 문서 및 문서화 기반의 작업 | |

Advantages and Disadvantages of BDD Summary

| BDD Advantages | | BDD 단점 |
|----------------|-------------------------------------|--------|
| 8 | 반복적인 테스트 도구 | |
| 9 | 기능 코드 기반으로 손쉽게 재구성(refactoring)이 가능 | |
| 10 | 품질 및 비용(cost) 절감 효과 | |

BDD

Title: Returns and exchanges go to inventory.

As a store owner,

I want to add items back to inventory when they are returned or exchanged,

so that I can sell them again.

Scenario 1: Items returned for refund should be added to inventory.

Given that a customer previously bought a black sweater from me

and I have three black sweaters in inventory,

when they return the black sweater for a refund,

then I should have four black sweaters in inventory.

Scenario 2: Exchanged items should be returned to inventory.

Given that a customer previously bought a blue garment from me

and I have two blue garments in inventory

and three black garments in inventory,

when they exchange the blue garment for a black garment,

then I should have three blue garments in inventory

and two black garments in inventory.

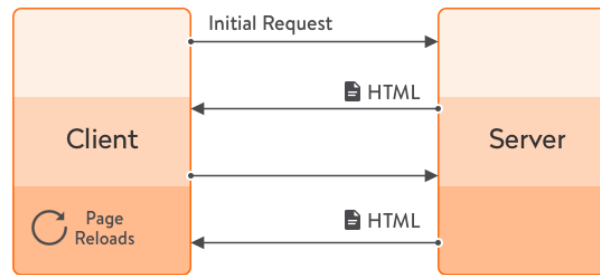
SPA(SINGLE PAGE APPLICATION)

SPA는 초기 웹 사이트에서 많이 사용하던 개발 방식이다. 말 그래도 단순한 페이지에 HTML 및 JavaScript 혹은 CGI기반으로 작성한 프로그램. 현재 대다수 웹 사이트들은 SPA기반으로 작성이 되어 있는 경우가 많다. 단점으로는 **페이지 프로세싱 시간이 발생하는** 부분이 있다.

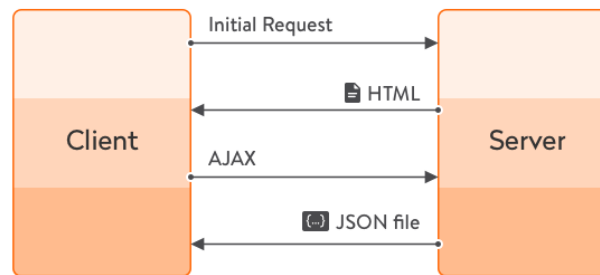
SPA반대는 MPA(Multi Page Application)이 있으며, 장점은 기능별로 페이지 디자인하기 때문에 관리가 편하다는 장점이 있지만, 해더 및 Application Page 및 JavaScript를 계속 반복적으로 전달해야 되는 단점이 있다. MPA기반으로 구성된 사이트는 보통 쇼핑몰 같은 웹 사이트가 많다.

SPA/MPA

Multi-page app lifecycle



Single-page app lifecycle



[SPA vs MPA]

참조 사이트

<https://www.altamira.ai/blog/single-page-app-vs-multi-page-app/>

:-<-<

2023-07-27

SPA/MPA

| SPA | MPA |
|--|---|
| <ul style="list-style-type: none">▪ impressive & dynamic client-side functionality – streaming, charting, real-time data, social network etc;▪ native-like / mobile-first experience;▪ already made API. | <ul style="list-style-type: none">▪ If the project has no complex logic;▪ SEO is one of the main requirements ;▪ if the project consists of several pages with static information – landing, blog, news;▪ no complex user interactions;▪ for Admin Panel or projects that have similar functionality (manage users / create users or manage products) / market places/catalogs. |

CONTINUOUS INTEGRATION

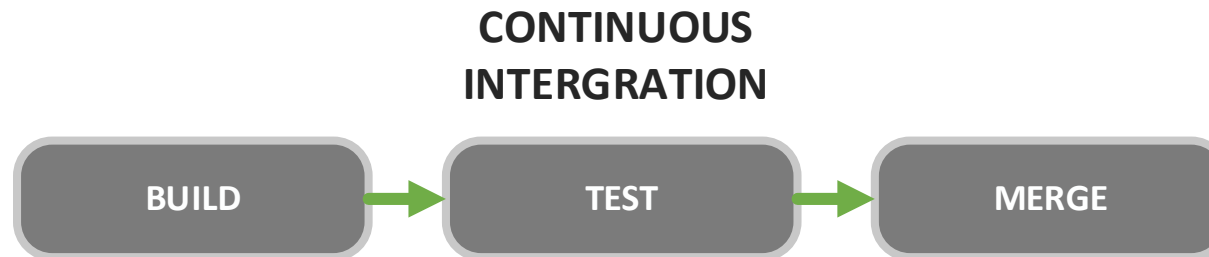
CI

CONTINUOUS INTEGRATION

CI/CD라는 단어에서는 **CI**는 Continuous Integration를 이야기 한다.

프로그램 개발을 시작하면 **변경/생성/테스트/통합**이 여러 저장소 혹은 디렉터리를 통해서 발생한다. 이러한 부분을 특정 규칙 및 프로그램을 통해서 정규화 시키며, 이를 보통 **SCM**이라고 부르기도 한다.

이전에는 이러한 부분을 SVN과 같은 도구를 사용하였지만 SVN 및 CVS는 한계가 명확하였다. CI영역은 보통 다음과 같은 단계를 구성하게 된다.



SVN의 단점

- 파일 및 디렉터리 이름 변경으로 버그가 발생할 수 있다.
- 저장소 관련 명령어의 부재. 수동으로 관리가 필요.
- 매우 느린 속도로 코드 비교.
- 리비전 및 커밋 서명 기능.

CI(GIT)

현재 소스코드 관리(Source-Control Management, **형상관리**) 도구는 아래와 같이 많이 사용한다.

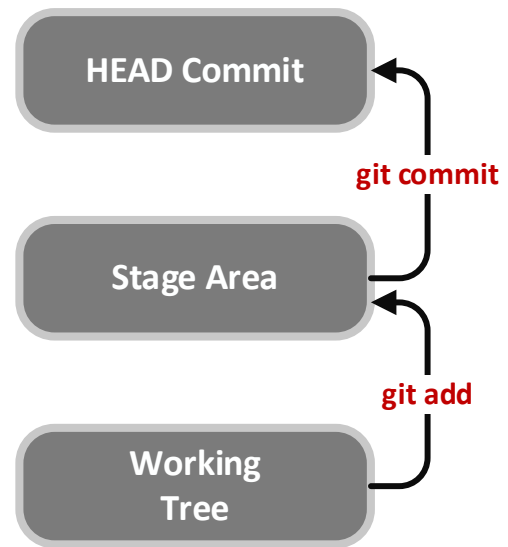
1. svn
2. git
3. BitKeeper

GIT놀랍게도, 리눅스 커널을 만든 리눅스 토발즈가 2005년도에 SCM도구로 만듦.

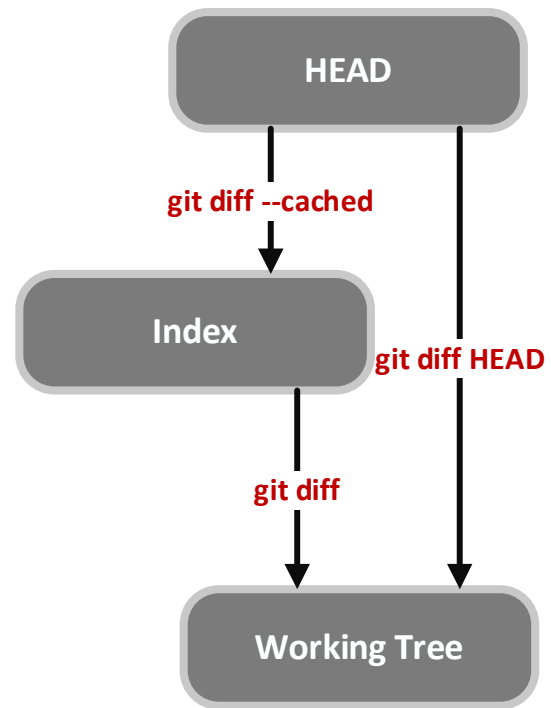
이전에 리눅스 토발즈는 **Bitkeeper**를 사용하였으나, 라이선스 및 갱신 시, 보통 30초 정도 메타정보 업데이트에 시간이 발생하기 때문에, 좀 더 빠르고 오픈소스 기반으로 사용이 가능한 SCM를 원했다. 리눅스 토발즈는 다음과 같은 조건으로 SCM도구를 만들기 시작했다.

- CVS처럼 모호한 시스템 결정 및 그리고 문제가 발생하면 작업 수행에 대해서 거부 한다.
- BitKeeper처럼 분산처리(오브젝트)를 작업순서를 제공한다.
- 악의적 혹은 우발/실수로 발생할 수 있는 손상에 대해서 보호기능을 제공한다.

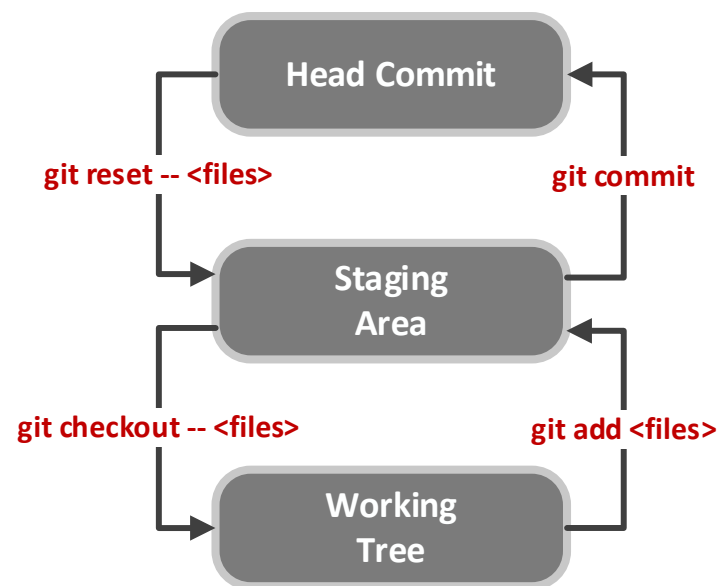
GIT



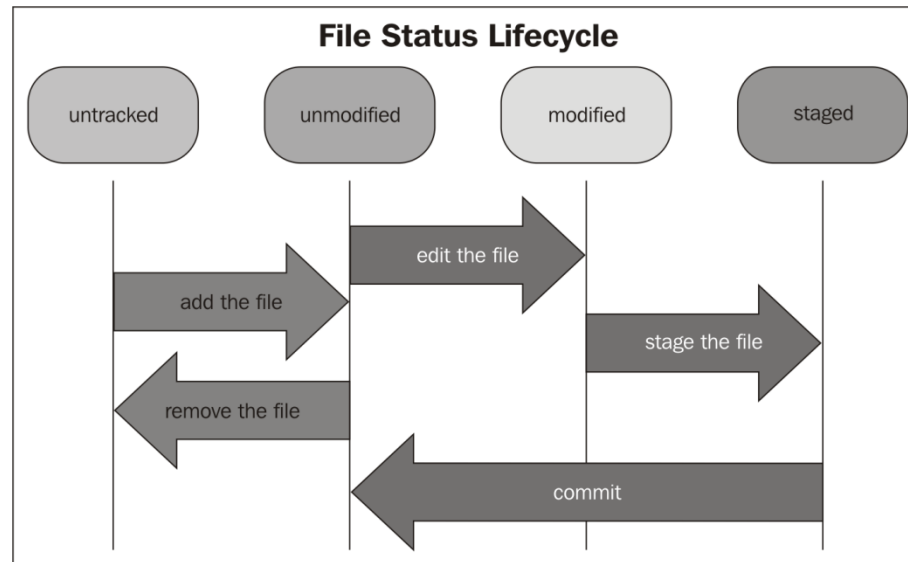
GIT



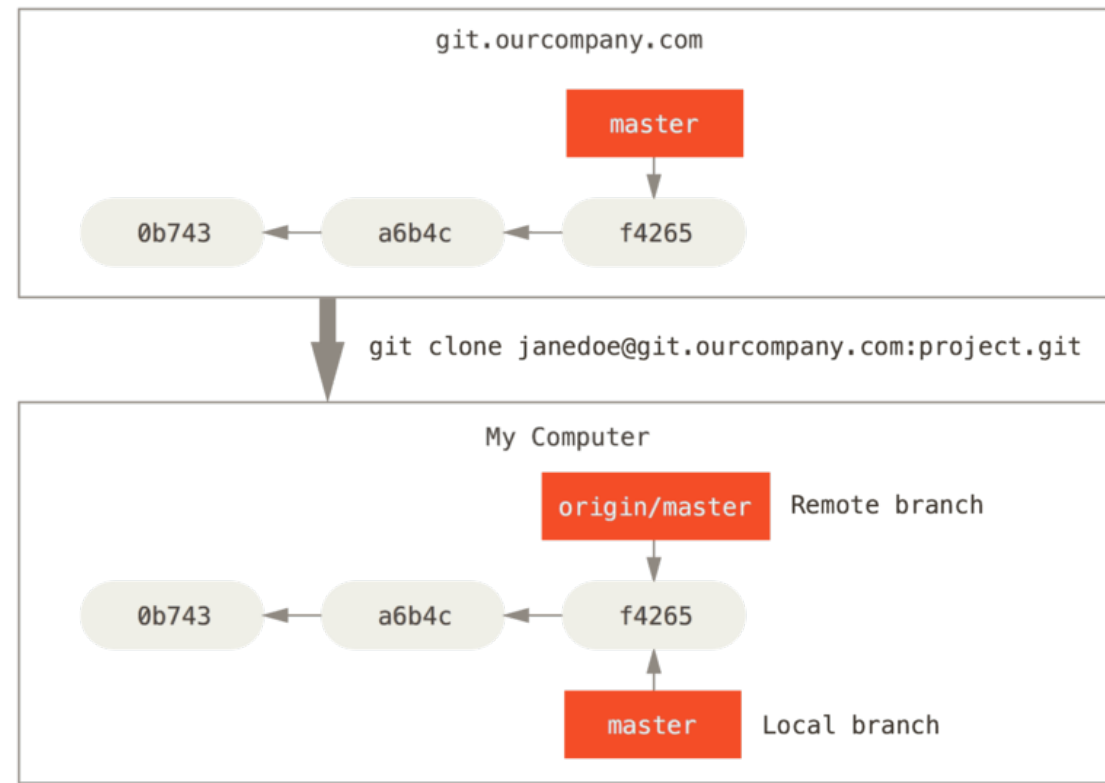
GIT



GIT



GIT ORIGIN



GIT INIT

```
$ mkdir git-test && cd git-test
```

```
$ git init -b main
```

Initialized empty Git repository in /root/test/.git/

```
$ git status
```

On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)

GIT CLONE

```
$ git clone <REPO_URL>
```

```
$ git clone ssh://[USER]@[SERVER_ADDRESS]/<USERNAME>/<REPOSITORY-NAME>.git
```

```
$ git clone -b <BRANCH_NAME> <REPO_URL>
```

GIT STATUS

\$ git status

\$ git clone https://github.com/tangt64/duststack-k8s-auto

Cloning into 'duststack-k8s-auto'...

remote: Enumerating objects: 1276, done.

remote: Counting objects: 100% (509/509), done.

remote: Compressing objects: 100% (241/241), done.

remote: Total 1276 (delta 233), reused 482 (delta 225), pack-reused 767

Receiving objects: 100% (1276/1276), 41.21 MiB | 9.29 MiB/s, done.

Resolving deltas: 100% (425/425), done.

GIT ADD

\$ git add <FILENAME>

개별적으로 파일 추가 시 보통 'add'로 특정 파일 명시.

\$ git add .

새로 추가된 파일 및 디렉터리를 GIT오브젝트에 등록.

\$ git add -A

'git add'와 동일.

GIT ADD

```
$ touch 1{1..10}
```

```
$ ls
```

```
11 110 12 13 14 15 16 17 18 19
```

```
$ git add 11
```

GIT ADD

\$ git status

On branch main

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: 11

Untracked files:

(use "git add <file>..." to include in what will be committed)

110

12

13

14

GIT COMMIT

```
$ git commit -m "<MESSAGE>"
```

```
$ git commit -m "11 file"
```

```
$ git log
```

```
# git logcommit 2fbbd7e13d1c877e08195b6e2be035cbe8969c72 (HEAD -> main)
```

```
Author: root <root@tang-laptop>
```

```
Date: Thu Jul 27 09:49:46 2023 +0900
```

```
11 file
```

GIT오브젝트에 등록된 객체에 메시지를 남긴다.

GIT RM

git rm -r <FILENAME>

GIT오브젝트에 등록된 파일을 제거한다. 실제 파일 시스템에서 제거되지는 않으며, refs메타에서만 제외된다.

\$ git rm 16

\$ git rm 11

GIT RM

\$ git log

commit 2fbbd7e13d1c877e08195b6e2be035cbe8969c72 (HEAD -> main)

Author: root <root@tang-laptop>

Date: Thu Jul 27 09:49:46 2023 +0900

11 file

\$ git commit -m "deleted"

\$ git log

GIT BRANCH

git branch/git branch -a

깃에 등록이 되어있는 지점(branch)목록 확인.

\$ git branch *master*

GIT에 지점 생성

\$ git switch *master*

GIT에 등록된 지점 변경.

\$ git branch -d main

GIT에 등록된 특정 지점 제거(-D옵션과 동일)

\$ git branch -m master main

GIT에

GIT PUSH

```
$ git remote -v
```

```
$ git push origin --delete <BRANCH_NAME>
```

```
$ git push
```

GIT CHECKOUT

\$ git checkout -b <BRANCH_NAME>

GIT서버에 변경된 내용이 있는지 확인한다.

\$ git checkout -b <BRANCH_NAME> origin/<BRANCH_NAME>

\$ git checkout <BRANCHE_NAME>

\$ git checkout -

\$ git checkout -- <FILENAME>

GIT MERGE

```
git merge <BRANCH_NAME>
```

```
git merge <SOURCE_BRANCH> <TARGET_BRANCH>
```


GIT STASH

`git stash`

`git stash clear`

GIT PUSH

`git push origin <BRANCH_NAME>`

`git push -u origin <BRANCH_NAME>`

`git push`

`git push origin --delete <BRANCH_NAME>`

GIT PULL

`git pull`

`git pull origin <BRANCH_NAME>`

GIT REMOTE

```
git remote add origin ssh://
```

```
git remote set-url origin ssh://
```

GIT LOG

`git log`

`git log --summary`

`git log --oneline`

GIT DIFF

```
git diff <SOURCE_BRANCH> <TARGET_BRANCH>
```

GIT REVERT

`git revert commitid`

GIT CONFIG

```
$ git config --global user.name
```

```
$ git config --global user.email
```

```
$ git config --global --list
```

```
$ git config --global credential.helper cache
```

```
$ vi ~/.git/config
```


REVERT COMMIT

```
$ git revert <SHA_ID>
```

```
$ mkdir revert_example/ && cd revert_example/
```

```
$ git init
```

```
$ touch alpha.html beta.html gamma.html rc.html
```

```
$ git add alpha.html && git commit -m "1ST"
```

```
$ git add beta.html && git commit -m "2nd"
```

```
$ git add gamma.html && git commit -m "3rd"
```

```
$ git add rc.html && git commit -m "4th"
```

```
$ ls
```

REVERT COMMIT

git reflog

git rever <SHA_ID>

git reset

undo every change

git revert

undo exactly the SHA ID

SVN/BITKEEPER

현재 대다수 SCM는 GIT으로 전환이 되고 있다. 위의 두 개의 CI 도구를 사용하지 않는 이유는 아래와 같다.

1. Bit Keeper는 라이선스 문제로 모든 오픈소스 사용자가 사용이 어려움
2. SVN의 모호한 메타 관리 방법 및 파일 관리 방법 및 중앙 파일 관리형식

SVN은 특히 리누즈 토발즈가 직접 비평한 영상이 있다. 아래 주소에서 확인이 가능하다.

<https://www.youtube.com/watch?v=4XpnKHJAok8>

CONTINUOUS DELIVERY

CD

CONTINUOUS DELIVERY

CI/CD에서 **Continuous Delivery**약자이다. CI에서 빌드 및 테스트가 완료가 되면, 자동적으로 컴파일 및 생성이 된 바이너리 결과물을 저장소에 릴리즈 한다.

보통 CI에서 발생한 결과물은 **파이프라인(Pipeline)**를 통해서 **배포(Deployment)**를 준비한다. 이를 통해서 자동화를 통한 빌드 및 검증이 완료가 된 패키지를 빠르게 서비스로 배포할 수 있도록 합니다.

CONTINUOUS DEPLOYMENT



JENKINS

대표적인 CD도구는 다음과 같다.

- Jenkins(Jenkins-X)
- Tekton

대다수가 많이 사용하는 도구는 **Jenkins**이다. **Jenkins**프로젝트 본래 이름은 **허드슨(Hudson)**으로 개발이 되었다.

오라클과 분쟁이 있는 후, 프로젝트를 **분기(forked)**하여, **젠킨스(Jenkins)**라는 이름으로 다시 시작하였다. 젠킨스는 자바로 작성이 되었으며, 2011년도에 릴리즈 하였다. 본 프로젝트인 허드슨은 2017년도에 오라클에서 공식적으로 종료 하였다. (하여간, 오라클과 붙는 오픈소스는 죄다 종말이...)

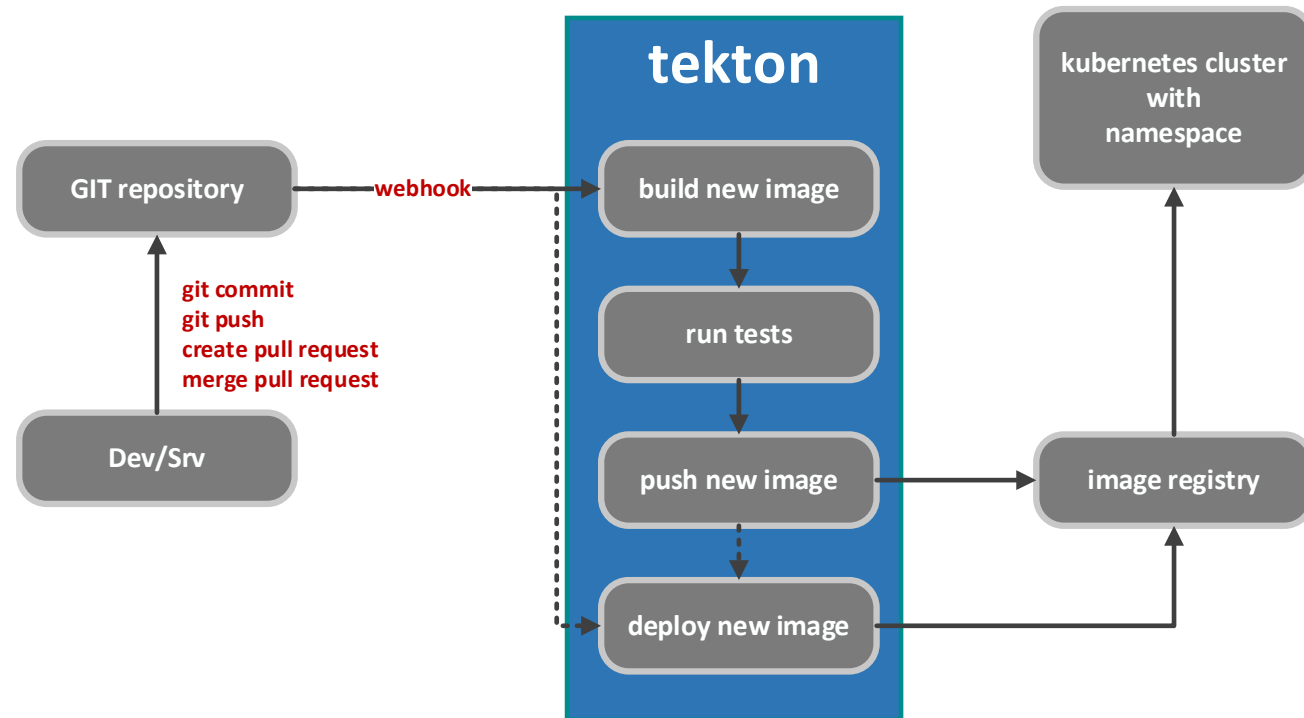
젠킨스는 몇 안되는 아시아인 프로젝트 중 하나이며, 일본인 [Kohsuke Kawaguchi](#)이 최초 작성자이다. DevOps가 활성화 되면서 젠킨스는 더 이상 자바 이외 다른 분야에도 사용하기 시작하였고, 좀 더 작고 DevOps환경에 맞게 작은 크기로 Jenkins X프로젝트가 시작이 되었다.

젠킨스는 크게 두 가지 역할을 가지고 있다.

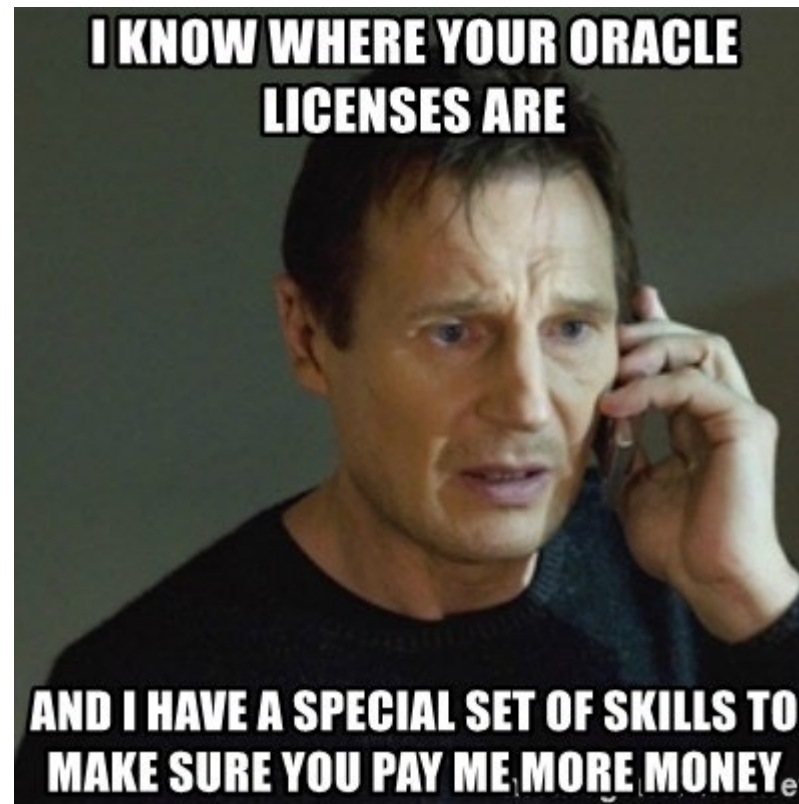
1. 소스코드 빌드
2. 프로그램 테스트 및 검증
3. 플러그인을 통한 기능 확장

TEKTON

테크톤(Tekton)은 CNCF에 표준에 맞게 작성된 CD도구이다. 젠킨스와 다른 부분은, 테크톤은 쿠버네티스 기반으로 동작하는 CD도구로 작성이 되었다. 사용자는 손쉽게 파이프라인(Pipeline)을 YAML기반으로 작성 및 구성이 가능하다.



CD



DSL(domain-specific language)

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the [Pipeline domain-specific language \(DSL\) syntax](#). ^[1]

- **Jenkinsfile**

blueocean

TEST UNIT

DEVSEC

OPENSCAPE

SPRING/QUARKS

| Metrics | Spring Boot JVM | Quarkus JVM | Spring Boot Native | Quarkus Native |
|---------------------------|-----------------|-------------|--------------------|----------------|
| Startup time (sec) | 1.865 | 1.274 | 0.129 | 0.110 |
| Build artifact time (sec) | 1.759 | 5.243 | 113 | 91 |
| Artifact size (MB) | 30.0 | 31.8 | 94.7 | 80.5 |
| Loaded classes | 8861 | 8496 | 21615 | 16040 |
| CPU usage max(%) | 100 | 100 | 100 | 100 |
| CPU usage average(%) | 82 | 73 | 94 | 92 |
| Heap size startup (MB) | 1048.57 | 1056.96 | - | - |
| Used heap startup (MB) | 83 | 62 | 12 | 58 |
| Used heap max (MB) | 780 | 782 | 217 | 529 |
| Used heap average (MB) | 675 | 534 | 115 | 379 |
| RSS memory startup (MB) | 494.04 | 216.1 | 90.91 | 71.92 |
| Max threads | 77 | 47 | 73 | 42 |
| Requests per Second | 7887.29 | 9373.38 | 5865.02 | 4932.04 |

QUARKUS TDD SAMPLE

```
$ sudo dnf module list
```

```
$ sudo dnf module enable maven
```

```
$ mvn io.quarkus.platform:quarkus-maven-plugin:3.2.2.Final:create -DprojectGroupId=com.redhat -  
DprojectArtifactId=quarkus-tdd -DclassName="com.redhat.tdd.TDDResource" -Dpath="/tdd" -  
Dextensions="resteasy-reactive-jackson"
```


QUARKUS TDD DEV MODE

\$ vi pom.xml


```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-junit5</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>io.rest-assured</groupId>
  <artifactId>rest-assured</artifactId>
  <scope>test</scope>
</dependency>
```



\$ mvn quarkus:dev

firefox://http://localhost:8080/q/dev -> q/dev-ui/extensions


 Dev UI


Extensions


quarkus-tdd 1.0.0-SNAPSHOT [Back to the previous Dev UI](#) 


 ArC


Build time CDI dependency injection

 Beans 35

 Observers 2


 Removed components 69







RETEasy Reactive

A Jakarta REST implementation utilizing build time processing and Vert.x. This extension is not compatible with the quarkus-resteasy extension, or any of the extensions that depend on it.

 Endpoint scores

 Exception Mappers

 Parameter converter providers

Eclipse Vert.x

Write reactive applications with the Vert.x API

Mutiny

Write reactive applications with the modern Reactive Programming library Mutiny

SmallRye Context Propagation

Propagate contexts between managed threads in reactive applications

JSON-P

JSON Processing support




Continuous Testing

 Extensions

 [Start](#)

 Configuration

 Continuous Testing

 Dev Services

 Build Metrics

```
@Test
public void getAll() {
    given()
        .when().get("/tdd")
        .then()
            .statusCode(200)
            .body(
                "$.size()", is(1),
                "[0].id", is(1),
                "[0].message", is("Hello")
            );
}
```

```
@Test
public void getOneFound() {
    given()
        .when().get("/tdd/1")
        .then()
            .statusCode(200)
            .body(
                "id", is(1),
                "message", is("Hello")
            );
}
```

```
@Test
```

```
public void getOneNotFound() {
```

```
    given()
```

```
        .when().get("/tdd/2")
```

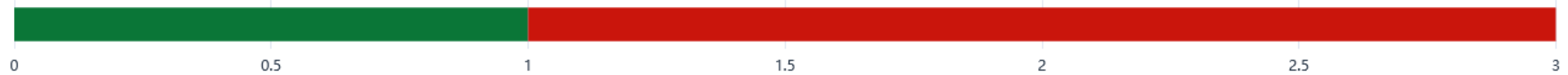
```
        .then().statusCode(404);
```

```
}
```

Stop Run all Run failed Only run failing tests

Total time: 270ms

| Test Class | Name | Time |
|------------------------------------|-------------------|------|
| </> com.redhat.tdd.TDDResourceTest | getAll() | 37ms |
| </> com.redhat.tdd.TDDResourceTest | findOneFound() | 15ms |
| </> com.redhat.tdd.TDDResourceTest | findOneNotFound() | 8ms |



| Test Class ↕ | Name ↕ | Time ↕ |
|------------------------------------|---------------------|--------|
| </> com.redhat.tdd.TDDResourceTest | getAll() | 17ms |
| </> com.redhat.tdd.TDDResourceTest | findOneFound() | 13ms |
| </> com.redhat.tdd.TDDResourceTest | testHelloEndpoint() | 11ms |
| </> com.redhat.tdd.TDDResourceTest | findOneNotFound() | 14ms |

```
$ vi src/main/java/com/redhat/tdd/Item.java
```

```
package com.redhat.tdd;
```

```
public class Item {
```

```
    private Long id;
```

```
    private String message;
```

```
    public Item() {
```

```
    }
```

```
    public Item(Long id, String message) {
```

```
        this.id = id;
```

```
        this.message = message;
```

```
    }
```



```
public Long getId() {  
    return this.id;  
}
```

```
public void setId(Long id) {  
    this.id = id;  
}
```

```
public String getMessage() {  
    return this.message;  
}
```

```
public void setMessage(String message) {  
    this.message = message;  
}  
}
```

```
$ vi src/main/java/com/redhat/tdd/TDDResource.java  
# private final List items = List.of(new Item(1L, "Hello"));
```

```
$ vi src/main/java/com/redhat/tdd/TDDResource.java
```

```
package com.redhat.tdd;
```

```
import java.util.List;
```

```
import javax.ws.rs.GET;
```

```
import javax.ws.rs.Path;
```

```
import javax.ws.rs.PathParam;
```

```
import javax.ws.rs.Produces;
```

```
import javax.ws.rs.core.MediaType;
```

```
import javax.ws.rs.core.Response;
```

```
import javax.ws.rs.core.Response.Status;
```

```
@Path("/tdd")

public class TDDResource {

    private final List<Item> items = List.of(new Item(1L, "Hello"));

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Item> getAll() {
        return items;
    }
}
```

```
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("/{id}")
public Response getOne(@PathParam("id") Long id) {
    return items.stream()
        .filter(item -> id == item.getId())
        .findFirst()
        .map(item -> Response.ok(item).build())
        .orElseGet(() -> Response.status(Status.NOT_FOUND).build());
}
}
```

TEKTON YAML

apiVersion: tekton.dev/v1beta

kind: Task

metadata:

name: mvn

spec:

workspaces:

- name: maven-repo

TEKTON YAML

params:

- name: GOALS

description: The Maven goals to run

type: array

default: ["package"]

resources:

- name: source

type: git

TEKTON YAML

steps:

- name: mvn

image: gcr.io/cloud-builders/mvn

workingDir: /workspace/source

command: ["/usr/bin/mvn"]

args:

- -Dmaven.repo.local=\$(workspaces.maven-repo.path)

- "\$\$(inputs.params.GOALS)"

JENKINS NODE SELECTOR

<https://plugins.jenkins.io/kubernetes/>

젠킨스에서 사용하는 빌드 POD를 특정 컴퓨트 노드에서 동작하기 위해서는 해당 플러그인 기반으로 설정해야 한다.

<https://github.com/redhat-developer/jenkins-operator>

OCP 4.9이후 부터는 젠킨스는 사용하기 위해서는 오퍼레이터를 컴파일 후 설치 권장. 가급적이면, 위의 주소에서 복제 후 사용을 권장. 별도로 YAML기반으로 생성 및 운영은 비권장

JENKINS OPERATOR DEPRECATED

Jenkins Operator missing (after OpenShift 4.9)

🔧 **SOLUTION IN PROGRESS** - 업데이트됨 15시 32분 2021년 10월 18일 - [English](#) ▾

환경

- OpenShift Container Platform (OCP)
- 4.x

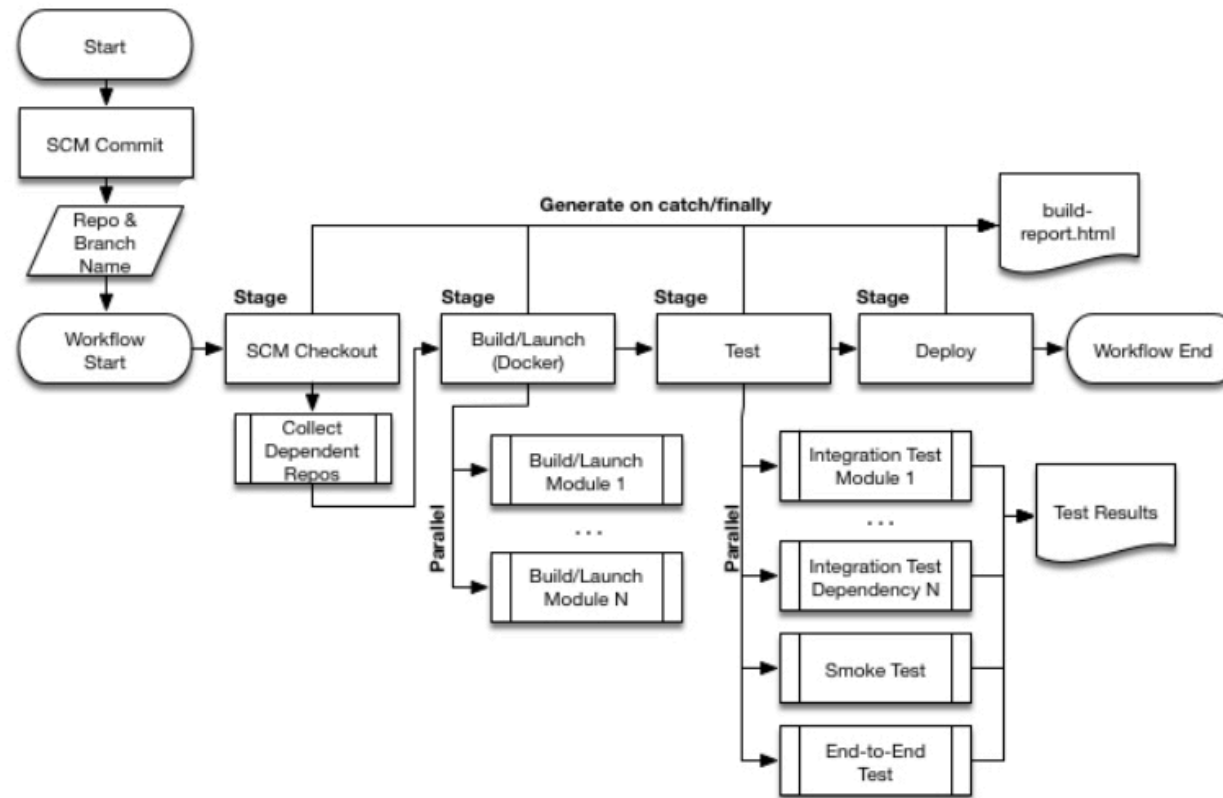
문제

- The Jenkins operator is not available in a version of OpenShift above 4.9

해결

- This is expected, the Jenkins operator has been deprecated.
- OCP 4.9 users receive a deprecation notice, but upgrades to OCP 4.9 allow the use of the operator. With 4.10, the operator is no longer available.
- It is available only in Tech Preview in OCP 4.6 - 4.8

JENKINS PIPELINE



JENKINS PIPELINE

The following concepts are key aspects of Jenkins Pipeline, which tie in closely to Pipeline syntax (see the overview below).

Pipeline

A Pipeline is a user-defined model of a CD pipeline. A Pipeline's code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it. Also, a pipeline block is a key part of Declarative Pipeline syntax.

Node

A node is a machine which is part of the Jenkins environment and is capable of executing a Pipeline.

Also, a node block is a key part of Scripted Pipeline syntax.

JENKINS PIPELINE

Stage

A stage block defines a conceptually distinct subset of tasks performed through the entire Pipeline (e.g. "Build", "Test" and "Deploy" stages), which is used by many plugins to visualize or present Jenkins Pipeline status/progress. [6]

Step

A single task. Fundamentally, a step tells Jenkins what to do at a particular point in time (or "step" in the process). For example, to execute the shell command make use the sh step: sh 'make'. When a plugin extends the Pipeline DSL, [1] that typically means the plugin has implemented a new step.

JENKINS Declarative

Jenkinsfile (Declarative Pipeline)

```
pipeline {  
    agent any  
    stages {  
        stage('Build') {  
            steps {  
                //  
            }  
        }  
    }  
}
```

JENKINS Declarative

```
stage('Test') {  
    steps {  
        //  
    }  
}
```


JENKINS Declarative

```
stage('Deploy') {  
    steps {  
        //  
    }  
}  
}
```

JENKINS Declarative

Jenkinsfile (Declarative Pipeline)

```
pipeline {  
    agent any  
    options {  
        skipStagesAfterUnstable()  
    }  
}
```

Jenkinsfile

```
stages {  
    stage('Build') {  
        steps {  
            sh 'make'  
        }  
    }  
}
```

Jenkinsfile

```
stage('Test'){  
    steps {  
        sh 'make check'  
        junit 'reports/**/*.*xml'  
    }  
}
```

Jenkinsfile

```
stage('Deploy') {  
    steps {  
        sh 'make publish'  
    }  
}  
}
```

JVM MONITOR

이전

<https://developers.redhat.com/blog/2016/03/30/jolokia-jvm-monitoring-in-openshift#>

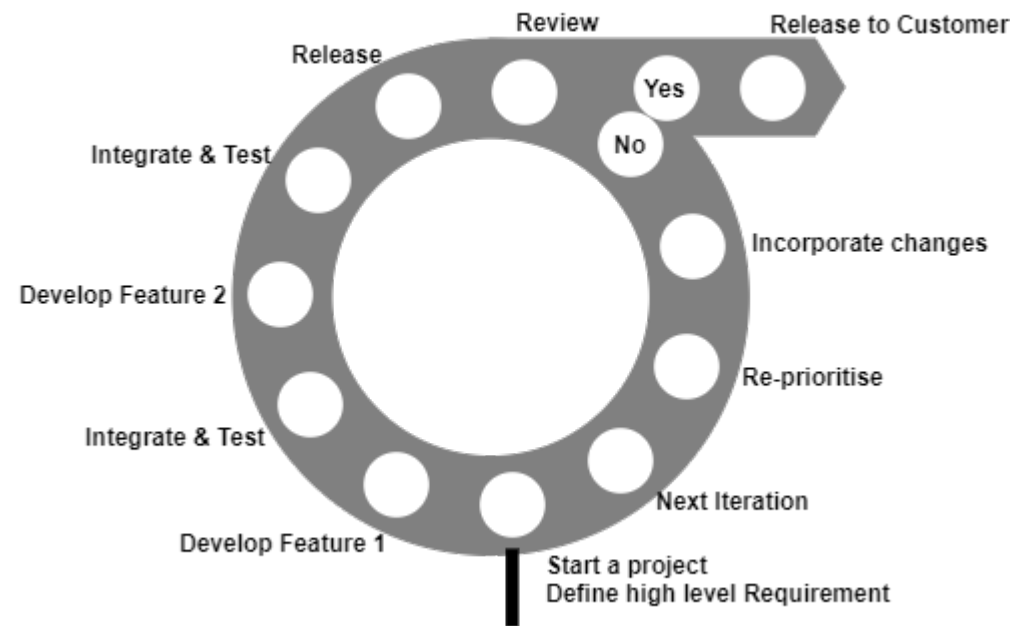
현재

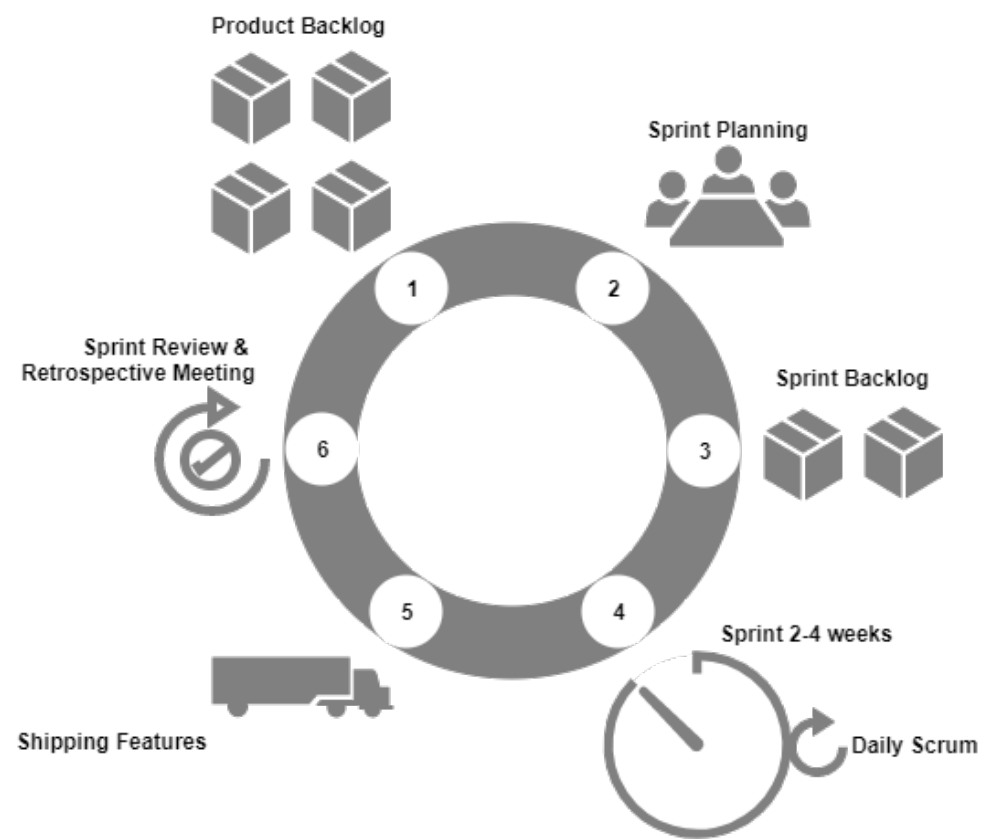
<https://developers.redhat.com/blog/2021/01/25/introduction-to-containerjfr-jdk-flight-recorder-for-containers>

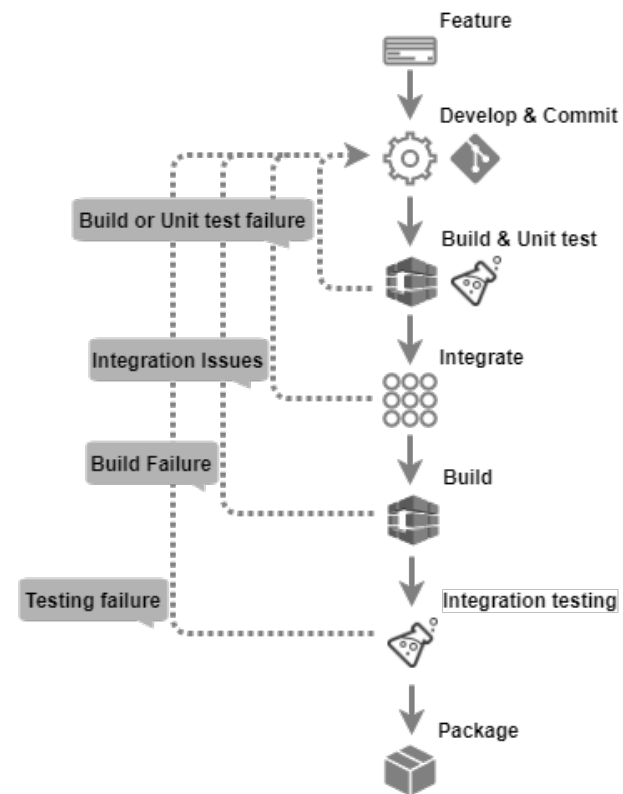
DELIVERY

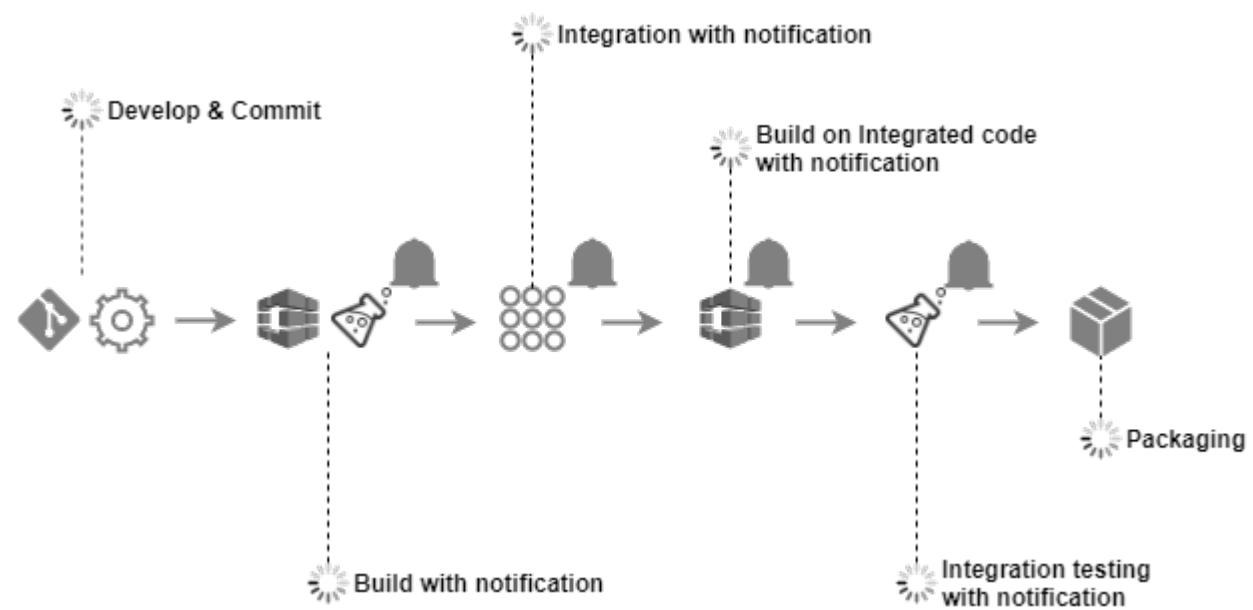
CI/CD

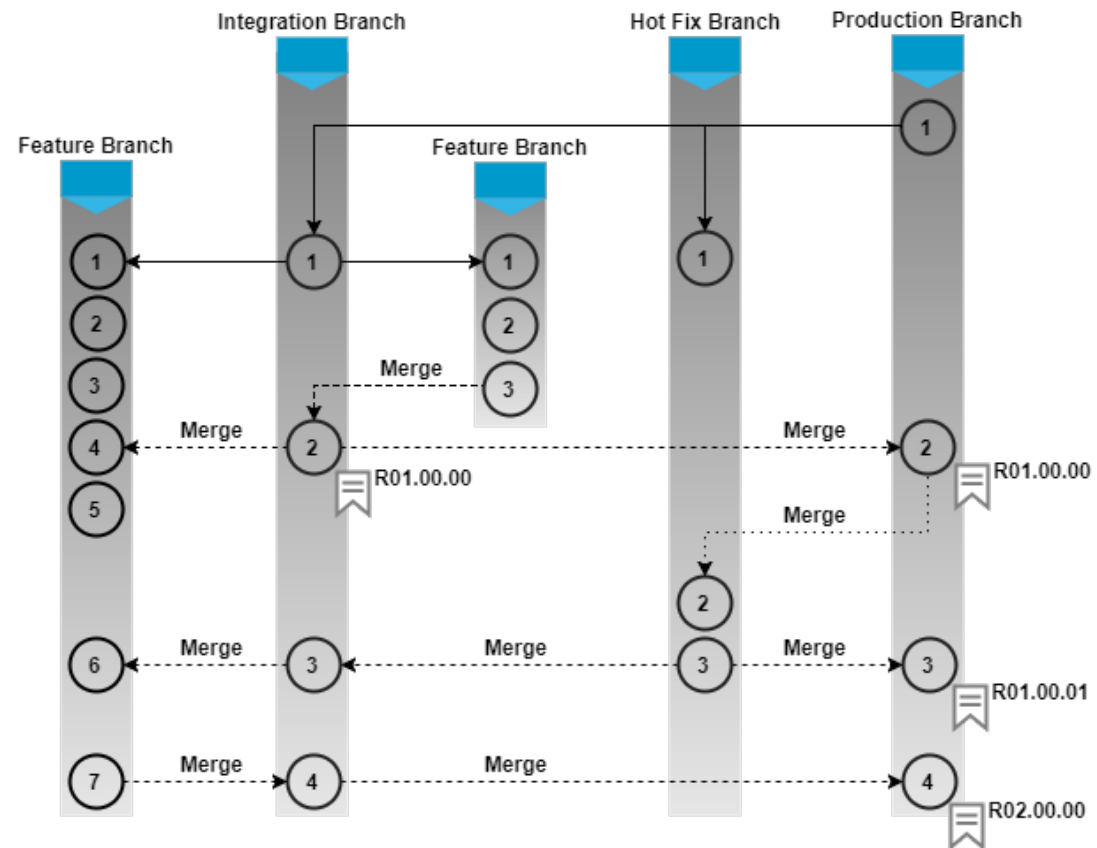


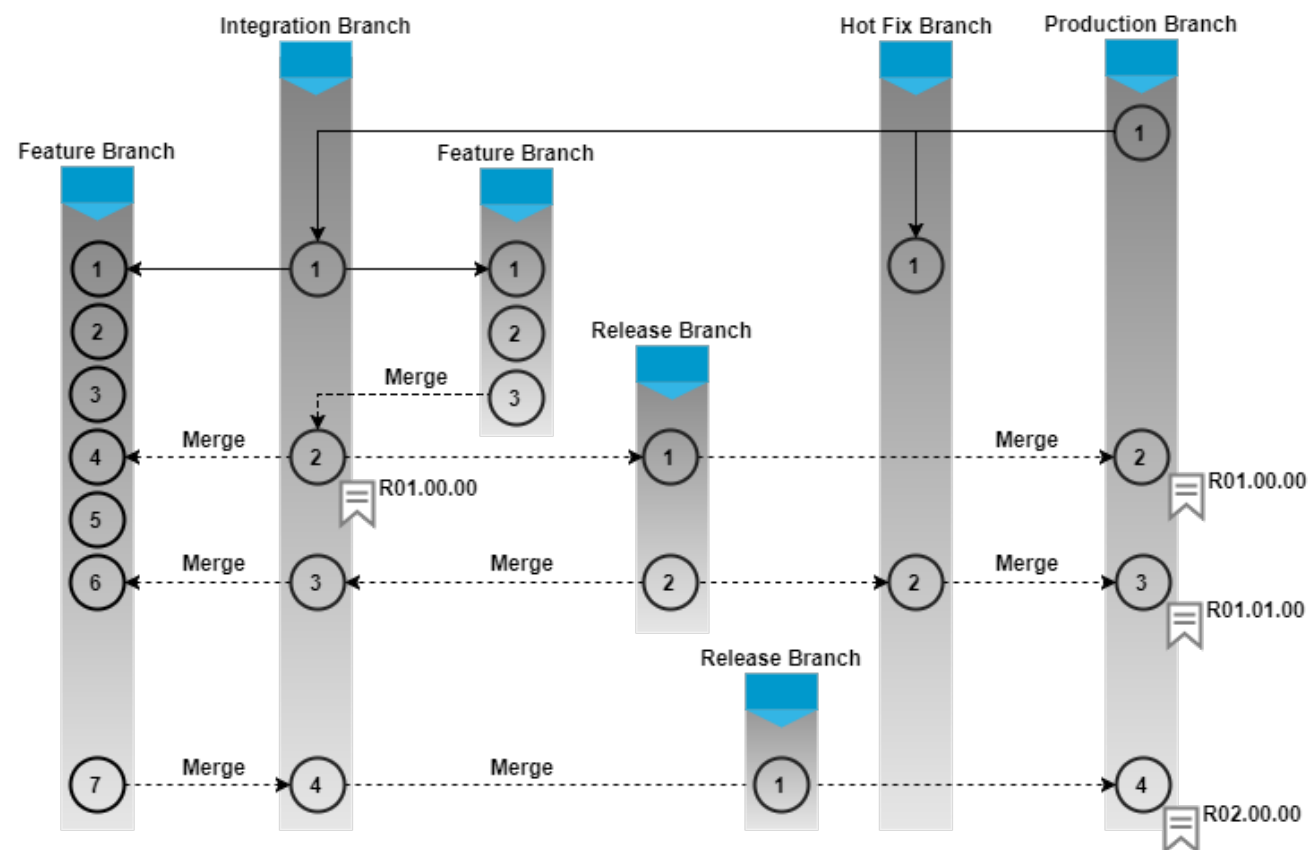


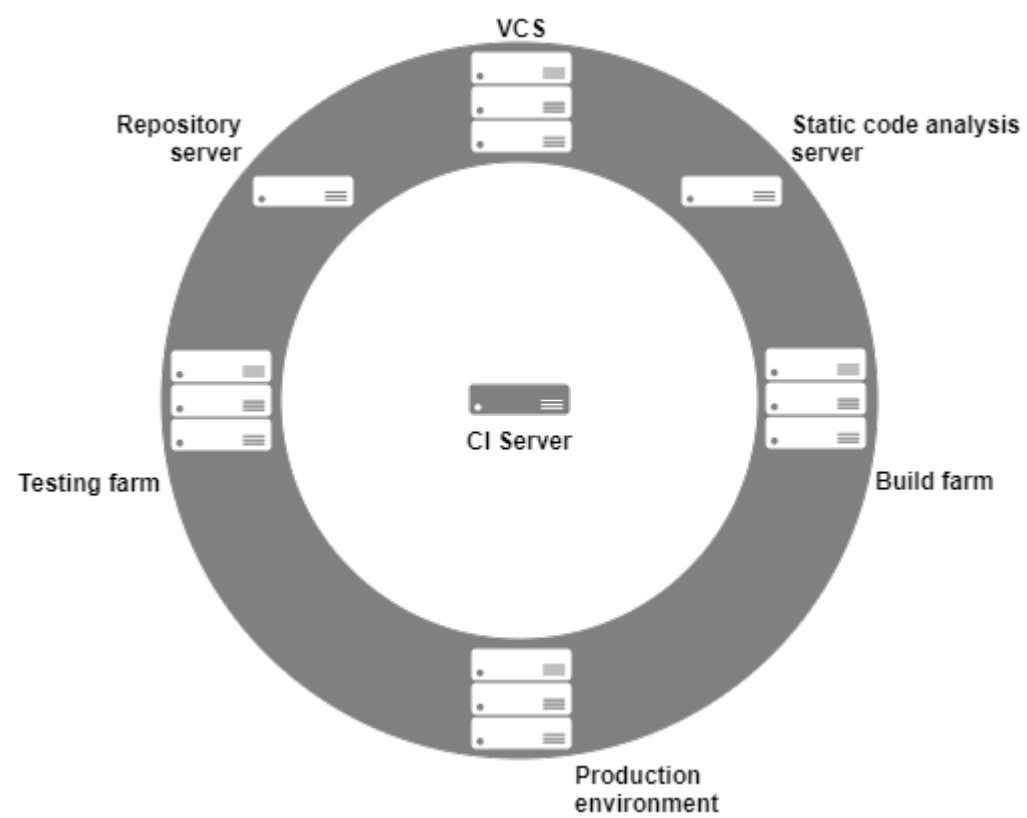


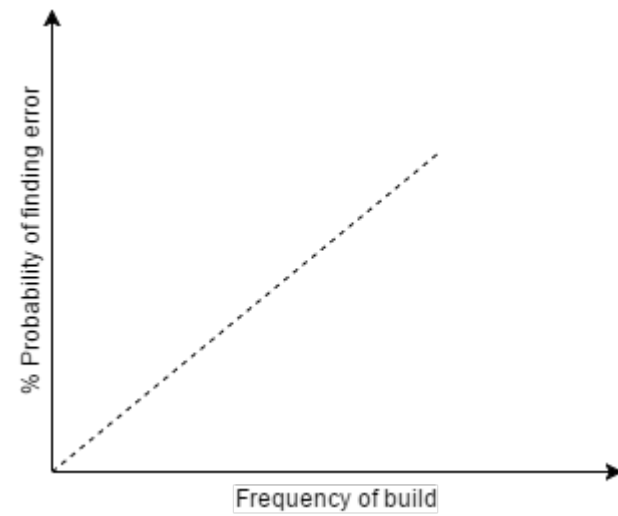












Language

Java

C#/.NET

C++

Python

Ruby

Tools

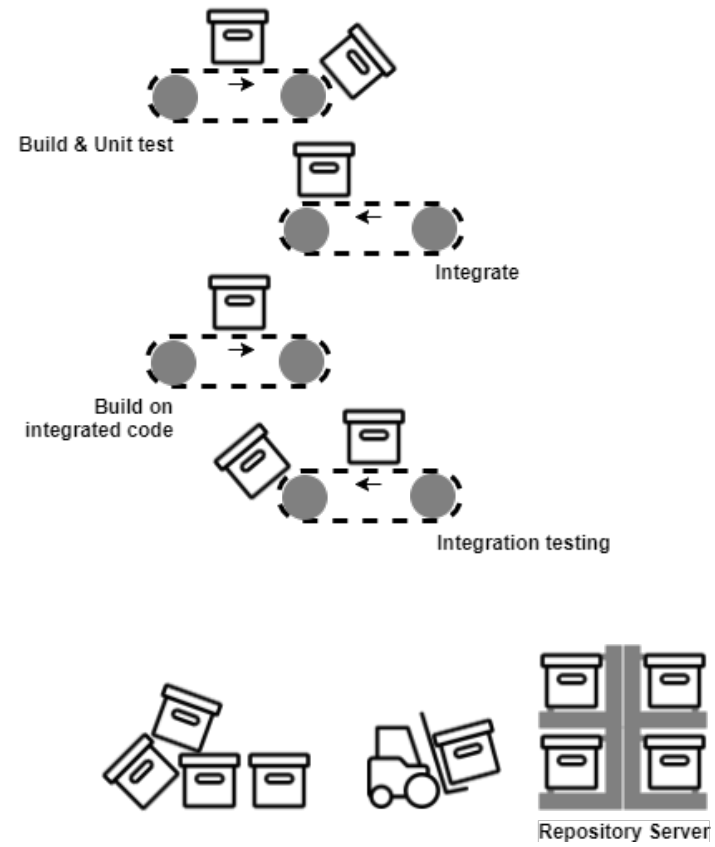
Atlassian Clover, Cobertura, JaCoCo

OpenCover, dotCover

OpenCppCoverage, gcov

Coverage.py

SimpleCov



Jenkins



apache groovy programming

<https://groovy-lang.org/syntax.html>

CI/CD