DevOps and Continuous Integration
DevOps: Development + IT Operations

Continuous Integration: a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.
持续集成（Continuous integration，简称CI），简单来说持续集成就是频繁地（一天多次）将代码集成到主干。
每次集成都通过自动化的构建（包括编译、发布、自动化测试）来验证，从而尽快地发现集成错误。

Benefits of Continuous Integration:

- Reduce integration problems
- Allows team to develop cohesive software more rapidly

Integration Problems

- merge conflict（同时修改了一个文件）
- compile conflict
- test conflict

10 Principles of Continuous Integration

- Maintain a code repository – version control
- Automate the build
- Make your build self-testing
- Everyone commits to mainline every day
- Every commit should build mainline on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what's happening
- Automate deployment

Automate the build

- Daily build: Compile working executable on a daily basis
  - Allows you to test the quality of your integration
  - Visible Progress
  - Quickly catches/exposes bug that breaks the build

Build from command line

Make your build self-testing

- Automated tests: Tests that can be run from the command line
- Examples:
  - Unit tests
  - Integration tests
  - Smoke test

Smoke Test

A quick set of tests run on the daily build.

- Cover most important functionalities of the software but NOT exhaustive
- Check whether code catches fire or “smoke” (breaks)
- Expose integration problems earlier

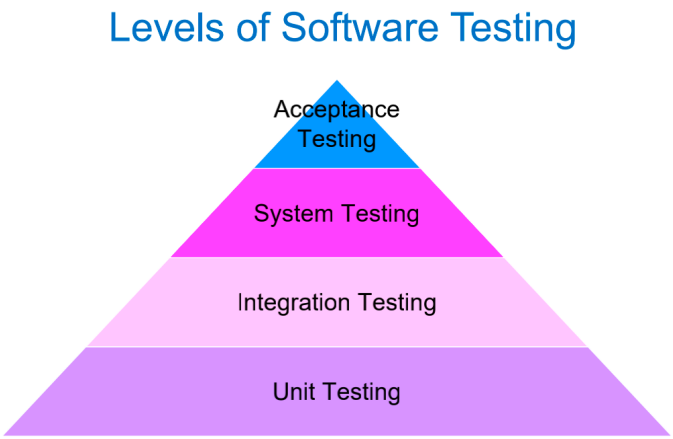
冒烟测试就是在每日build建立后，对系统的 基本功能进行简单的测试。

Continuous Integration server

CruiseControl, Jenkins, Bamboo, Travis CI

An external machine that automatically pulls your latest repo code and fully builds all resources.

- If anything fails, contacts your team (e.g. by email).
- Ensures that the build is never broken for long.



Unit Testing

- Test individual units of a software
- A unit: smallest testable part of an application

Integration testing

Verify software quality by testing two or more dependent software modules as a group.

将单元测试模块逐个集成/组合，并将行为测试为组合单元。该测试的主要功能或目标是测试单元/模块之间的接口。

Big-bang Integration Testing - All component are integrated together at once

一次性集成了所有模块

- Advantages:
  - Convenient for small systems.
- Disadvantages:
  - Finding bugs is difficult. 如果在完全集成的模块中检测到任何问题，则很难找出导致该问题的模块
  - Due to large number of interfaces that need to be tested, some interfaces could be missed easily.
    - Testing team need to wait until everything is integrated so will have less time for testing.
    - High risk critical modules are not isolated and tested on priority.

Incremental Integration Testing

- Incremental integration:
  - Develop a functional "skeleton" system
  - Design, code, test, debug a small new piece
  - Integrate this piece with the skeleton
  - test/debug it before adding any other pieces
- Advantages:
  - Errors easier to isolate, find, fix. Reduces developer bug-fixing load
  - System is always in a (relatively) working state. Good for customer relations, developer morale
- Disadvantages: May need to create "stub" versions of some features that have not yet been integrated

Top-down integration: Start with outer UI layers and work inward

- Must write (lots of) stub lower layers for UI to interact with
- Allows postponing tough design/debugging decisions (bad?)

Bottom-up integration: Start with low-level data/logic layers and work outward

- Must write test drivers to run these layers
- Won't discover high-level / UI design flaws until late

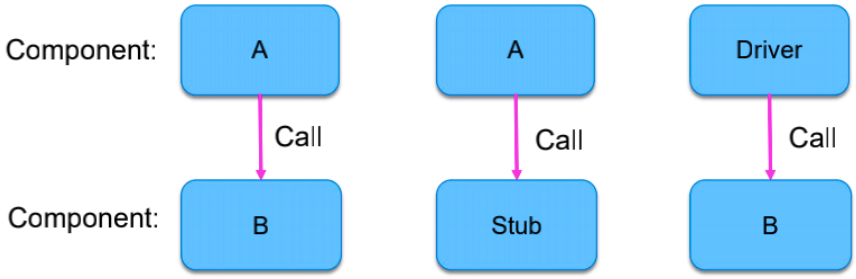
"Sandwich" integration: Connect top-level UI with crucial bottom-level classes

- Add middle layers later as needed
- More practical than top-down or bottom-up

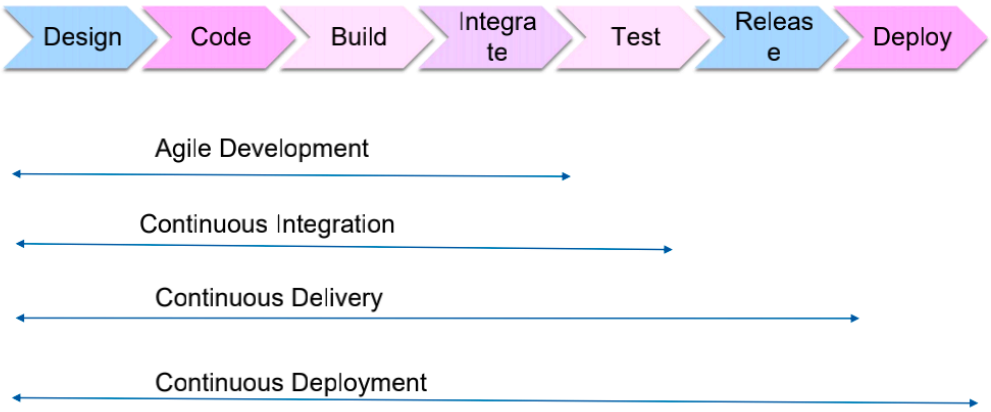
Stub versus Driver

Both are used to replace the missing software and simulate the interface between components (create dummy code)

Stub: Dummy function gets called by another function Driver: Dummy function to call another function



Continuous Integration & Continuous Deployment

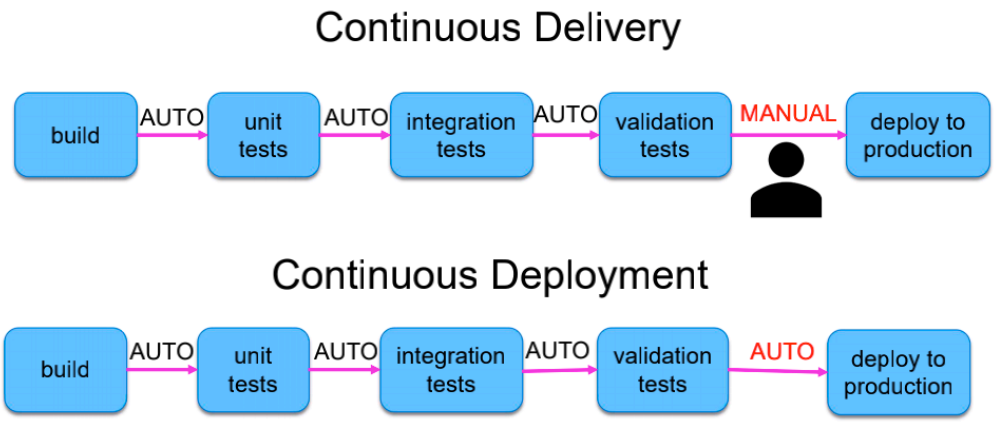


Continuous Delivery build software so that it is always in a state where it could be put into production
continuously running a deployment pipeline that tests if this software is in a state to be delivered

Continuous Integration != Continuous Delivery: CI != CD

- CD = CI + automated test suite
- Not every change is a release
  - manual trigger
  - trigger on a key file(version)
  - tag release
- CD - the key is automated testing

Continuous Delivery vs. Deployment (delivery手动部署，deployment自动)



Continuous Deployment

Continuous Deployment = CD + Automatic Deployment

Every change that passes the automated tests is deployed to production automatically

Deployment schedule:

- Release when a feature is complete
- Release every day

Deployment strategies

- Strategy 1: Zero-downtime deployment
- Strategy 2: Blue-green deployment
  - advantage: no shut down
  - disadvantage: need to maintain 2 copies; double effort; migration of data (更安全的做法还是shut down migrate)

Regression Testing

changes can both: improve software, adding feature and fixing bugs / break software, introducing new bugs

We call such “breaking changes” regressions

Regression testing

Testing that are performed to ensure that changes made does not break existing functionality

It means re-running test cases from existing test suites to ensure that

software changes do not introduce new faults
(保证修改不会造成之前过掉的test break，没有造成新的fault)

- Presubmit Testing
- postsubmit testing

Flaky Test
Test which could fail or pass for the same code
Sources of test flakiness:

- Concurrency
- Environment / setup problems
- Non-deterministic or undefined behaviors

应对：重跑处理/隔离处理
● AsyncWait ● Concurrency ● Test Order Dependency ● Resource Leak ● Network ● Time ● IO ● Randomness ● Floating Point Operations ● Unordered Collections