CS 362 - Software Engineering II
Continuous Integration(CI)

Background

When we learned about different software engineering methodologies, version control systems, and testing, we understood that software engineering/development is mainly, a highly collaborative field.

We also learned about the importance of version control systems and their benefits:

1. **A safe workflow for your project**

Multiple developers often work together in a team environment. Team environments require multiple developers to collaborate on the same software/code at the same time. If any contribution or "***commit***" causes a problem, it can be easily backed out.

2. **Rollbacks**

Version control systems manage different versions of your code, prevent you from losing your work or rolling back to a specific version of your work, or recover old versions of your code. Version control systems always save a snapshot of your codebase for you to use later, or review or obtain a specific version later.

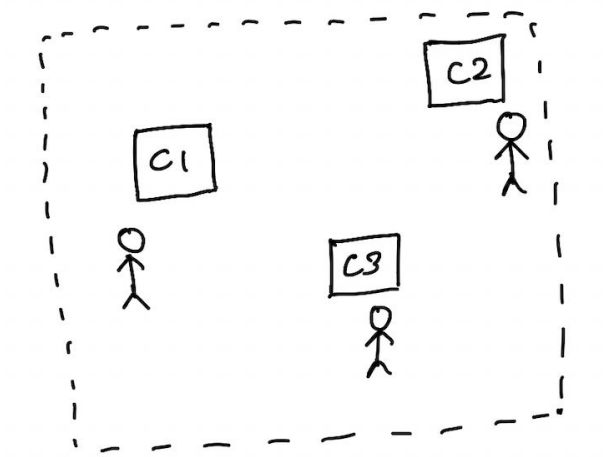
3. **Coordination**

In a collaborating environment or within a development team, manually sharing and merging code can be time-consuming and error-prone. Version control essentially avoids chaos and enables collaboration.

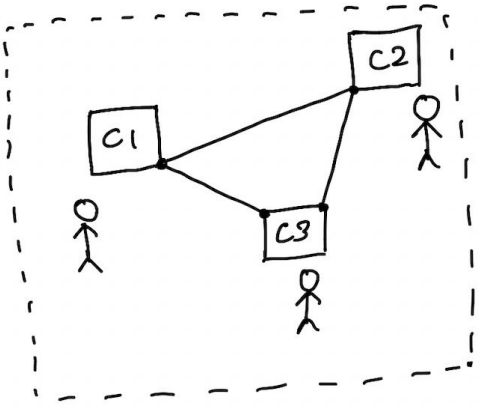
We also learned about testing - the different levels of testing and different testing strategies/techniques.
In this document, we are going to learn about how the following concepts fit together - Version control, collaboration, testing.

Continuous Integration

In a software project, developers work on different components of the system or different parts of the system.



Developers often write code in isolation, working on a particular component or sub-system. They then merge their changes to the main branch after they complete their respective developmental activities.
Merging code changes can be challenging, and can cause issues that affect delivery.



What's the role of continuous integration in this scenario?

With continuous integration, developers often commit to a shared repository using a version control system (example: Git). Before each developer makes a commit, he/she may choose to run tests on their local machine. In this case, the developer may write unit tests before integrating their code. A continuous integration tool or service mainly perform the following tasks -

1. Build, run unit tests and perform unit testing on the newer code.
2. Look for any new errors that surface after the code changes.

Activities within CI

1. Merging code changes to a central repository (Version Control).
2. Unit testing
3. Collaboration
4. Release

Essentially, continuous integration allows us to build code changes automatically, test changes, and release them to production. In continuous integration, we are mainly concerned with unit testing and builds. Every change that is committed creates a build and a test.

Continuous integration as a development practice

In software engineering, we know that members of a team collaborate together and frequently integrate their work. Continuous integration can be considered an extension of software development where members integrate their work on a frequent basis. There can be multiple integrations in a day. Each integration consists of tests to detect any errors that have been caused by integrating different components together.

During each integration, checking for errors is done as quickly as possible, this leads to reduced errors during future integrations.

Important links:

1. Building a Feature with Continuous Integration by Martin Fowler.
2. What is continuous integration?
3. Continuous integration - video explanation.

Benefits of continuous integration

1. Integrating frequently ensures that you are spending lesser time discovering issues, and spending more time working on implementing features.
2. Following continuous integration means that the time between integration is shorter. If you have longer durations between integrations, it makes it increasingly difficult to find and fix problems.
3. Continuous integration increases visibility and enabling greater communication.
4. Catches issues early help in fixing problems early.
5. Provides rapid feedback - you do not have to wait to find out if your code is going to work.
6. Reduces integration problems allowing you to deliver software more rapidly.
7. Early and improved error detection allows you to address errors early, sometimes within a few minutes of checking in your code.
8. Improves collaboration and every developer can change the code, integrate the system and quickly and determine problems with other components of the system.
9. Improved system integration reduces unexpected situations or surprises at the end of the software development lifecycle.
10. Continuously integrating reduces the number of errors during system testing.
11. Continuously integrating provides you with constantly updated systems to test against.

Continuous integration and Agile

By now we are well aware of the agile development methodology and its related aspects. We know that agile is focused on how software development teams organize themselves, adapt to changes in requirements, and continuously release software.

> Agile teams deliver quality software fast, without death marches or heroics. CI makes this possible.
>
> - Atlassian

Continuous integration and agile methodology development share some characteristics. Agile organizes software development into sprints. It is iterative, adapts to change, and is scalable. In terms of continuous integration, agile is a methodology that delivers software in iterations based on the features that have been prioritized.

Similarly, in continuous integration, developers often integrate work frequently, often many times a day. The integration is verified by automated builds that detect errors as early as possible, thereby providing rapid feedback.

How do things look without CI?

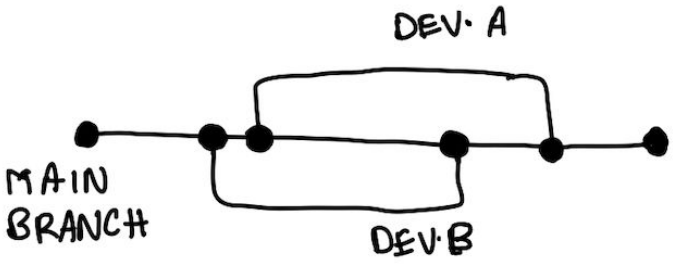
For example, if you have a team that relies on manual testing, it is not possible to get “rapid” feedback. At most, the team may receive feedback in a couple of hours. Sometimes detailed feedback may take longer than expected, during which more changes might have occurred. This is not a fast process. Investing time and effort into continuous integration reaps the benefits that we discussed earlier. The changes are “rapid” - feedback is obtained within minutes of any code changes that occur.

CI from a Software developer's perspective

Continuous Integration benefits developers in the following ways:

1. Developers can ensure that the code works on every machine, or is compatible with different operating systems, or browsers.
2. With CI, developers worrying less about introducing newer problems every time code changes are applied.
3. Integrating code and collaborating daily becomes a part of the team's engineering culture.
4. Developers can be a part of teams that build software faster with fewer bugs.
5. Get feedback faster on things they're currently working on.

CI Terminologies



1. **Branching**
   A branch in a VCS is an independent copy of the source code. Changes can be made to a branch without affecting other branches. Most often, teams have a main branch, which is the current state of the project.
   For a developer to work on a particular aspect of the code, he/she

should create a copy of the main branch and merge changes back to the main branch.

2. **Build**
   A build is in simple terms, an activity that translates source code that is human-readable into an executable program. It is the process of creating the application binaries for a software release, by taking all the relevant source code files and compile them, and then creating a build artifact (such as binaries and executable program, etc.)
3. **Automated Build**
   Automating the build process allows developers to commit often, providing rapid feedback about changes or conflicts. As feedback is provided within minutes of the changes being made, it also encourages developers to break their work down into smaller chunks, making it easier to test and track. It's easier to locate errors if there are numerous small changes compared to massive updates.

CI in practice

1. Creating a Continuous Integration pipeline

Things to know by now - Git, Python
Things we're getting to know - CircleCI

Prerequisites - Accounts in Github and CircleCI

We are going to learn about the steps required to implement continuous integration. These steps are also known as a CI pipeline.

Specification - revisiting the calculator app from the in-class activity of the unit testing module.

Steps

1. Create a repository and give it a name of your choice. For this tutorial, we'll use the name CalculatorApp.
   1.1 Create a readme, a git ignore and clone it to your local system.
2. Set up a virtual environment
   We need to replicate working conditions for the CI server and create a virtual environment.

   In your terminal, type in the following commands:
   `python3 -m venv calculator`
   This command creates a virtual environment.

   Activate your virtual environment by using the command below:
   `. calculator/bin/activate`

   You will know that you are in the virtual environment when you see it in your terminal.

```
(calculator) (base) vijaytadimeti@Vijays-MBP ~ %
```

3. From the in-class activity you should have a file named calculator.py(any name of your choice) that contains its related methods.

For now, the file includes the following methods:

calculator.py > ...

```python
"""
Calculator app containing basic methods.
"""


def add(a, b):
    return a + b


def subtract(a, b):
    return a - b

```

Next:

Commit your changes to Github. Make sure you are in the right directory. Do a git add and then a git commit with a message of your choice.

Your CalculatorApp folder should have the following files by now:
`.git`
`.gitignore`
`README.md`
`calculator.py`

Installing dependencies

In our in-class activity from weeks 8 & 9, we learned how to install pytest and pytest-cov.
If you do not have them installed, install them now. You will also need to install a linter.

```
pip install pytest pytest-cov flake8
```

Storing dependencies

We need to store our dependencies in a file. We need to “freeze” our requirements in a file named `requirements.txt`

```
pip freeze > requirements.txt
```

Running Linter

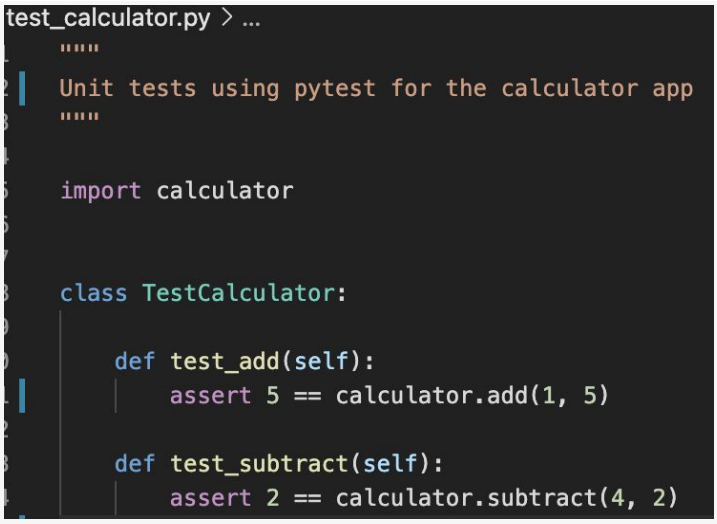
Run the linter you just installed using the following command.
`flake8 --statistics`

Unit Testing

For this example, we will use pytest for unit testing. Let's write our tests for the methods that we have so far.

Note: Let's not use our existing test file from the in-class activity. Create a new file `test_calculator.py`

Your tests should look something like this:



Running Tests

From the in-class activity, we learned how to run our pytest unit tests. Run the command below to run your tests.

```
pytest -v --cov
```

You should get the output shown below. If you've noticed, we used the -v flag and the --cov flag. Using these we get a detailed output and also the results of pytest-cov with a coverage report.

```
collected 2 items

test_calculator.py::TestCalculator::test_add PASSED                      [ 50%]
test_calculator.py::TestCalculator::test_subtract PASSED                 [100%]

---------- coverage: platform darwin, python 3.8.5-final-0 -----------
Name                 Stmts   Miss  Cover
----------------------------------------
calculator.py            4      0   100%
test_calculator.py       6      0   100%
----------------------------------------
TOTAL                   10      0   100%


============================== 2 passed in 0.03s ===============================
```

Commit to Github

We are going to stop for a bit and push the changes to our main branch. Do a git add to add files that you have made changes to. Do a git commit with an appropriate commit message. Finally, do a git push.

Our directory structure should now look like this:

```
.git
.gitignore
README.md
calculator.py
requirements.txt
test_calculator.py
```

Enter CircleCI

For CircleCI to know our build and to run it, we need to have a `.circleci` folder and a YAML config file in it.

If you can recall, we used YAML files in our *Black-box testing in-class activity.* Let's name our config file `config.yml`

Paste the following content into your yml file. Let's worry about the details

later.

ci > ! config.yml

```yaml
# CircleCI config file
version: 2
jobs:
  build:
    docker:
      - image: circleci/python:3.7

    working_directory: ~/repo

    steps:
      #1: Got repository from GitHub
      - checkout
      #2: Created virtual environment and install dependencies
      - run:
          name: install dependencies
          command: |
            python3 -m venv venv
            . venv/bin/activate
            pip install -r requirements.txt
      #3: Ran linter and unit tests
      - run:
          name: run tests
          command: |
            . venv/bin/activate
            flake8 --exclude=venv* --statistics
            pytest -v --cov=calculator
```

Starting our pipeline

So far, we got everything ready that was essential for our CI pipeline.

Next steps:

Log into your CircleCI account. Look for your repository for Github and click

We have a config.yml file. Skip the next steps and click on *Start Building.*

If all went well, you should be seeing your job succeed.

Your final pipeline should now like this:

```
.git
.gitignore
README.md
calculator.py
requirements.txt
test_calculator.py
```

Next steps:
Now try and apply TDD. Write the test for the multiply method first before you write the code.

Every time a change has been made, i.e, a push to the main branch is made, a new job is triggered.

1. Write the test for the multiply method and push the code to the main branch. The job will fail to ensure that our continuous integration is working.

2. Pass the test and by writing the code in your calculator file.

Note: Keep an eye on the spaces between the functions. The code may fail the linter check otherwise.

If you now look at the job, it should be successful.

The same can be done with the division method as well.

Useful links and resources:

1. CircleCI documentation
2. Continuous Integration in Agile
3. Continuous Integration, Continuous Delivery, and Continuous Deployment
4. Continuous integration using Github actions