Contents

[I. Introduction 1](#_Toc129993790)

[II. La construction logicielle (build) avec make 1](#_Toc129993791)

[a. Expliquez comment vous l'invoquez à partir de la ligne de commande 1](#_Toc129993792)

[b. Expliquez comment vous l'invoquez à partir de Visual Studio Code ? 1](#_Toc129993793)

[c. Expliquez les tâches accomplies par votre build 1](#_Toc129993794)

[a. Expliquez la mise en place de l'environnement CI 2](#_Toc129993795)

[a. Expliquez le fonctionnement de l'environnement CI 4](#_Toc129993796)

[III. Importance des enjeux reliés à la qualité et à la productivité 4](#_Toc129993797)

[b. Par l'utilisation d'un build 4](#_Toc129993798)

[c. Par l'utilisation d'un environnement CI 4](#_Toc129993799)

[IV. Conclusion 4](#_Toc129993800)

# Introduction

# La construction logicielle (build) avec make

Le Makefile est un fichier qui permet d'automatiser l'exécution de tâches récurrentes, telles que la compilation, la génération de documentation, l'exécution de tests, etc. Il contient des règles qui spécifient comment les fichiers sources doivent être compilés pour générer des fichiers exécutables ou des documents.

Le Makefile est souvent utilisé pour automatiser le processus de construction (ou build) d'un projet. Le Makefile spécifie les règles pour la compilation, l'exécution de tests, la génération de documentation, etc. en utilisant une syntaxe simple basée sur des cibles et des dépendances.

## Expliquez comment vous l'invoquez à partir de la ligne de commande

Nous pouvons exécuter ces cibles en utilisant la commande "make" suivie du nom de la cible. Par exemple : « make unittest » exécute la cible unittest

Pour invoquer le build à partir de la ligne de commande, il suffit d'ouvrir un terminal dans le répertoire où se trouve le Makefile et d'exécuter la commande « make build ». Cela va lancer le build et exécuter toutes les étapes définies dans le Makefile.

## Expliquez comment vous l'invoquez à partir de Visual Studio Code ?

Pour invoquer le build à partir de VS Code, nous utilisons l'extension "Tasks" de l’IDE. Voici les étapes à suivre :

On ouvre le menu "Terminal" dans la barre de menu supérieure de VS Code, puis on ouvre un "Nouveau terminal".

Dans le terminal, on tape la commande **make -p** pour lister les tâches disponibles. **Puis make build** pour exécuter la tâche de build.

## Expliquez les tâches accomplies par votre build

**Notre makefile contient plusieurs cibles pour effectuer différentes tâches :**

* "***lint***" : cette cible exécute *Pylint* pour effectuer une analyse statique du code et détecter les erreurs de style.
* "***unittest***" : cette cible exécute les tests unitaires avec *unittest*.
* "***pytest***" : cette cible exécute les tests unitaires avec *pytest* et génère un rapport de couverture du code en HTML.
* "***docs***" : cette cible utilise *pdoc* pour générer la documentation HTML pour le code. Cela générera une documentation HTML dans le dossier "docs" du projet.
* "***coverage***" : cette cible utilise *Coverage* pour vérifier la couverture du code.
* **"*all*"** : cette cible exécute toutes les tâches *(lint, unittest, docs, coverage*) en une seule fois.

Le Makefile peut générer des rapports XML pour les résultats des tests en utilisant des outils tels que unittest par exemple.

Pour générer un rapport de tests en format XML avec unittest on a installé le module "xmlrunner" qui permet de générer des rapports XML à partir des résultats de tests unittest.

Dans notre code de tests, on utilise le module *xmlrunner* pour générer le rapport XML à la fin de l'exécution des tests. On a le code suivant a la fin du fichier de test :

* if \_\_name\_\_ == '\_\_main\_\_':

with open('resultats\_tests.xml', 'wb') as output:

unittest.main(

testRunner=xmlrunner.XMLTestRunner(output=output),

failfast=False, buffer=False, catchbreak=False)

Puis dans le makefile, on a rajoute la ligne suivante dans la cible unittest pour executer les tests.

Rajouter ligne

Cela exécutera les tests et générera un rapport de tests en format XML dans un fichier appelé "resultats\_tests.xml".

## Expliquez la mise en place de l'environnement CI

**Environnement d'intégration continue avec Jenkins :**

**installer minikube**

**installer jenkins**

[**http://localhost:8080**](http://localhost:8080) **pour acceder a l’interface web de jenkins, installer le plugin kubernetes, créer un job pipeline**

## Expliquez le fonctionnement de l'environnement CI

# Importance des enjeux reliés à la qualité et à la productivité

## Par l'utilisation d'un build

En utilisant un build, on peut automatiser le processus de construction de notre logiciel. Cela permet de garantir la cohérence et la qualité de notre produit final. Par exemple, le build peut effectuer des tâches telles que la compilation, les tests unitaires, la génération de rapports de couverture de code, l'analyse du code, la création de packages et la génération de documentation. En automatisant ces tâches, nous pouvons détecter rapidement les erreurs et les problèmes de qualité dès qu'ils surviennent, ce qui permet d'économiser du temps et de l'argent.

## Par l'utilisation d'un environnement CI

L'environnement CI est comme un assistant qui travaille en arrière-plan pour s'assurer que tout se passe bien lorsque des modifications sont apportées au code source d'un projet. Il vérifie automatiquement que les modifications sont correctement intégrées au code existant en effectuant des tests et en construisant le logiciel Si le build ou les tests échouent, l'environnement CI envoie une notification immédiatement au développeur afin qu'il puisse corriger le problème avant que cela ne devienne un problème plus important. Cela permet de maintenir la qualité du code et d'améliorer la productivité de l'équipe de développement.