Synthèse

* Code, style, …
* Simplification, mutualisation, factorisation

Mot-clés à rechercher

* Convention
* Clean code
* Test
* Intégration continue
* Git flow / git policy

Convention de nommage

Convention / commentaires (cartouche, …)

Convention d’écriture (indentation, retour à la ligne, …)

**Clean code**

## Le Clean Code, c’est quoi ?

Le Clean Code n’est pas un ensemble de règles strictes mais désigne plutôt une série de principes pour produire un code **compréhensible de façon intuitive** et **facile à modifier**. Compréhensible signifie dans ce cas un code immédiatement intelligible par n’importe quel développeur qualifié. Les caractéristiques suivantes augmentent la lisibilité du Clean Code :

* Le **processus d’exécution** de l’ensemble de l’application est **logique et structuré de façon simple**.
* Le **rapport** entre les différentes parties du code est **transparent**.
* La **tâche ou le rôle** de chaque classe, fonction, méthode et variable est **immédiatement compréhensible**.

Un code est facile à modifier lorsqu’il peut être facilement ajusté et complété. À l’inverse, il est également plus simple de corriger les erreurs dans le code. Par conséquent, le Clean Code est très facile à entretenir. Un code facilement modifiable comporte les attributs suivants :

* Les classes et les méthodes sont **petites** et, dans la mesure du possible, ont une seule et unique tâche.
* Les classes et les méthodes sont **prévisibles**, fonctionnent de la façon attendue et sont accessibles au grand public grâce à des API bien documentées (interfaces).
* Le code dispose de [**tests d'unité**](https://www.ionos.fr/digitalguide/sites-internet/developpement-web/test-unitaire-dans-le-developpement-de-logiciels/).

Les avantages d’une telle programmation sont évidents : le Clean Code n’est **pas dépendant du développeur initial**. En principe, n’importe quel programmeur peut ainsi travailler avec un tel code. Ceci permet par exemple d’éviter les problèmes survenant lorsque l’on travaille avec un [legacy code](https://www.ionos.fr/digitalguide/sites-internet/developpement-web/quest-ce-que-le-code-legacy/" \o "Qu’est-ce que le code legacy ?). L’**entretien du logiciel s’en trouve également facilité** puisque les bugs peuvent être trouvés et corrigés facilement.

**Intégration continue**

L'**intégration continue** est un ensemble de pratiques utilisées en [génie logiciel](https://fr.wikipedia.org/wiki/G%C3%A9nie_logiciel) consistant à vérifier à chaque modification de [code source](https://fr.wikipedia.org/wiki/Code_source) que le résultat des modifications ne produit [pas de régression](https://fr.wikipedia.org/wiki/Non-r%C3%A9gression) dans l'application développée.

Avec la complexification constante des programmes **informatiques**, les méthodes agiles du **Test Driven Development** (**TDD**) sont de plus en plus privilégiées. Il y a de bonnes raisons à cela : grâce au **TDD**, les programmeurs s'assurent que le design d'un logiciel est bien pensé avant de commencer à écrire le code.

----------------------------

**Règles / convention de nommage**

Les **règles de codage** sont un ensemble de règles à suivre pour uniformiser les pratiques de [développement logiciel](https://fr.wikipedia.org/wiki/D%C3%A9veloppement_logiciel), diffuser les [bonnes pratiques](https://fr.wikipedia.org/wiki/Bonne_pratique) de développement et éviter les erreurs de développement "classiques" au sein d'un groupe de [développeurs](https://fr.wikipedia.org/wiki/D%C3%A9veloppeur).

Les règles de codage s'articulent autour de plusieurs thèmes, les plus courants étant :

* Le nommage et l'organisation des fichiers du [code source](https://fr.wikipedia.org/wiki/Code_source)
* le [style d'indentation](https://fr.wikipedia.org/wiki/Style_d%27indentation)
* Les [conventions de nommage](https://fr.wikipedia.org/wiki/Convention_de_nommage), ou [règles de nommage](https://fr.wikipedia.org/wiki/Identificateur#Règles_de_nommage)
* Les [commentaires et documentation](https://fr.wikipedia.org/wiki/Commentaire_(informatique)) du code source
* Recommandations sur la déclaration des [variables](https://fr.wikipedia.org/wiki/Variable_(informatique))
* Recommandations sur l'écriture des [instructions](https://fr.wikipedia.org/wiki/Instruction_(informatique)), des [structures de contrôle](https://fr.wikipedia.org/wiki/Structure_de_contr%C3%B4le) et l'usage des [parenthèses](https://fr.wikipedia.org/wiki/Parenth%C3%A8se) dans les [expressions](https://fr.wikipedia.org/wiki/Expression_(informatique)).

<https://en.wikipedia.org/wiki/Coding_best_practices>

 [1 Software quality](https://en.wikipedia.org/wiki/Coding_best_practices#Software_quality)

 [2 Prerequisites](https://en.wikipedia.org/wiki/Coding_best_practices#Prerequisites)

* [2.1 Life cycle](https://en.wikipedia.org/wiki/Coding_best_practices#Life_cycle)
* [2.2 Requirements](https://en.wikipedia.org/wiki/Coding_best_practices#Requirements)
* [2.3 Architecture](https://en.wikipedia.org/wiki/Coding_best_practices#Architecture)
* [2.4 Design](https://en.wikipedia.org/wiki/Coding_best_practices#Design)
* [2.5 Choice of programming language(s)](https://en.wikipedia.org/wiki/Coding_best_practices#Choice_of_programming_language(s))

 [3 Coding standards](https://en.wikipedia.org/wiki/Coding_best_practices#Coding_standards)

* [3.1 Commenting](https://en.wikipedia.org/wiki/Coding_best_practices#Commenting)
* [3.2 Naming conventions](https://en.wikipedia.org/wiki/Coding_best_practices#Naming_conventions)
* [3.3 Keep the code simple](https://en.wikipedia.org/wiki/Coding_best_practices#Keep_the_code_simple)
* [3.4 Portability](https://en.wikipedia.org/wiki/Coding_best_practices#Portability)
* [3.5 Scalability](https://en.wikipedia.org/wiki/Coding_best_practices#Scalability)
* [3.6 Reusability](https://en.wikipedia.org/wiki/Coding_best_practices#Reusability)
* [3.7 Construction guidelines in brief](https://en.wikipedia.org/wiki/Coding_best_practices#Construction_guidelines_in_brief)

 [4 Code development](https://en.wikipedia.org/wiki/Coding_best_practices#Code_development)

* [4.1 Code building](https://en.wikipedia.org/wiki/Coding_best_practices#Code_building)
* [4.2 Testing](https://en.wikipedia.org/wiki/Coding_best_practices#Testing)
* [4.3 Debugging the code and correcting errors](https://en.wikipedia.org/wiki/Coding_best_practices#Debugging_the_code_and_correcting_errors)
* [4.4 Deployment](https://en.wikipedia.org/wiki/Coding_best_practices#Deployment)

**Git flow / Git Policy**

<https://m-applications.devoteam.com/2020/02/24/gitflow-les-bonnes-pratiques-de-git/>