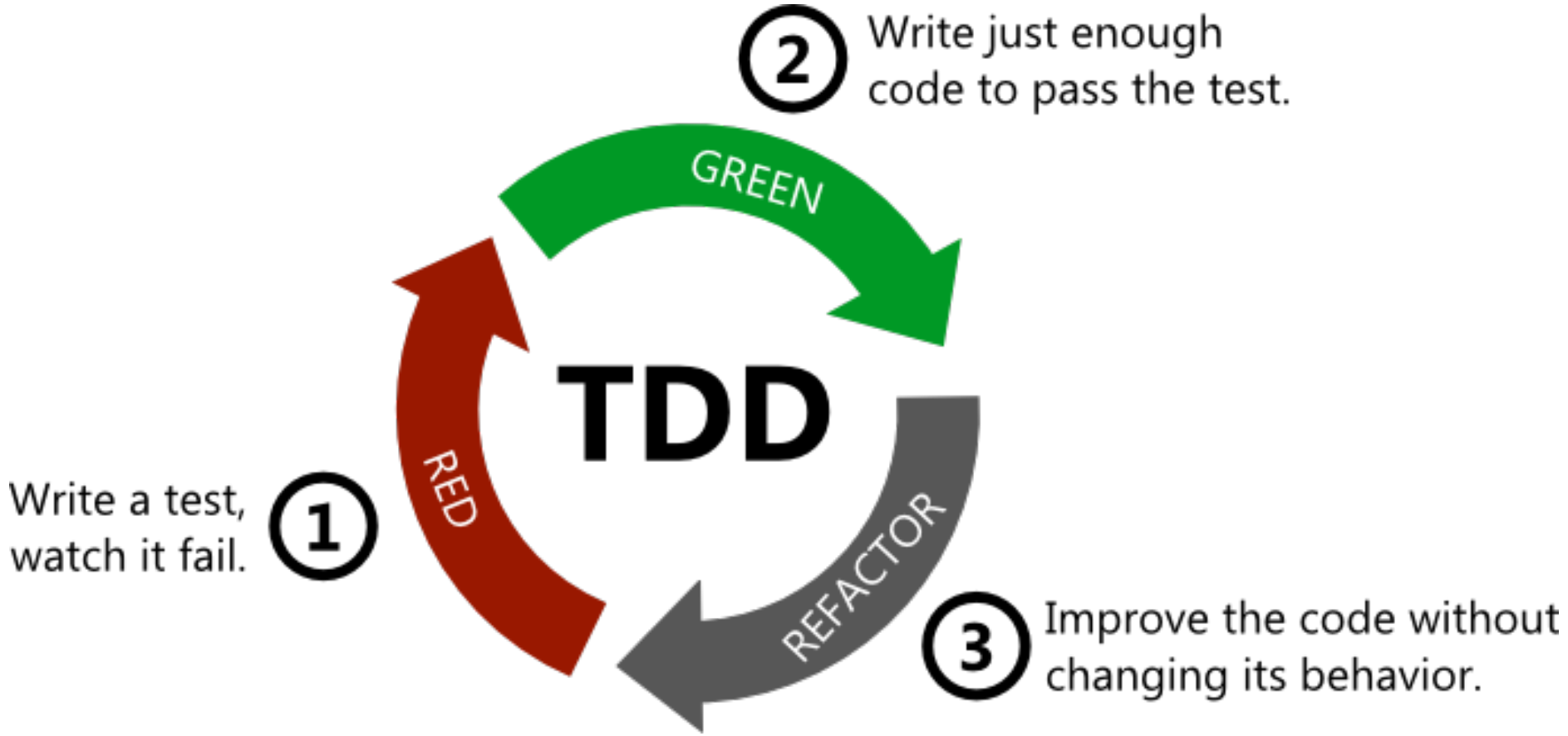
TDD

Méthode de développement piloté par les tests.


2
Write just enough
code to pass the test.
GREEN
TDD
RED
REFACTOR
Write a test,
watch it fail.
1
3
Improve the code without
changing its behavior.

Les trois lois du TDD

1. Vous devez écrire un test qui échoue avant d'écrire tout code de production

2. Vous ne devez pas écrire plus d'un test suffisant pour échouer, ou qui échouera à la compilation

3. Vous ne devez pas écrire plus de code que nécessaire pour faire passer le test en cours

Etapes de développement

1. Ecrire un seul test qui décrit une partie du problème à résoudre
2. Vérifier que le test échoue
3. Ecrire le code le plus simple pour que le test passe
4. Vérifier que le test passe
5. Refactorer

Guidelines de tests

- Tester le comportement et pas l'implémentation.
- Chaque test doit être indépendant des autres tests.
- Chaque test doit être répétable.
- Appliquer une convention de nommage pour les tests.
- Temps d'exécution des tests doit être très rapide.
- Refactorer les tests et le code après 3 répétitions.
- Les résultats de tests sont binaire
  - (soit il passe ou soit il ne passe pas.)

Object Calisthenics

1. Utiliser un seul niveau d'indentation par méthode.
2. Ne pas utiliser le "else".
3. Utiliser un seul "." par ligne de code.
4. Encapsuler les types primitifs et les strings.
5. Ne pas abréger.
6. Faire des petites classes (50 lignes de codes).
7. Ne pas utiliser des classes avec plus de deux variables d'instance.
8. Encapsuler les collections.
9. Ne pas utiliser les propriété Get/Set.

But: Object Calisthenics

- Maintenabilité
- Lisibilité
- Testabilité
- Compréhensibilité

Transformation Priority Premise

- **({}–>nil)** no code at all->code that employs nil
- **(nil->constant)**
- **(constant->constant+)** a simple constant to a more complex constant
- **(constant->scalar)** replacing a constant with a variable or an argument
- **(statement->statements)** adding more unconditional statements.
- **(unconditional->if)** splitting the execution path
- **(scalar->array)**
- **(array->container)**
- **(statement->recursion)**
- **(if->while)**
- **(expression->function)** replacing an expression with a function or algorithm
- **(variable->assignment)** replacing the value of a variable.

But: Transformation Priority Premise

- En prenant dans l'ordre de priorité les transformations, le code deviendra plus générique.
- En suivant ces transformations on évite des complications "accidentel" du code.
- Aide a guider l'ordre d'écriture des tests.
  - Exemple: Si on décide d'écrire un test qui ajoute une condition, ce test appliquera la transformation (unconditional -> if)

FIN