

Module 8 — Qualite et Securite

Jour 2 — TDD, OWASP, Git, cost

The Bearded Bear

Février 2026



Table des matières

Module 8 : Qualite et Securite (1h)	2
Objectifs	2
1. TDD/BDD avec Claude Code	2
Le Cycle Red-Green-Refactor	2
Workflow TDD avec Claude	2
Generation de tests	3
Couverture de code	3
BDD avec Gherkin	3
2. Audit de Code	4
Analyse d'architecture	4
Detection d'anti-patterns	5
Metriques de complexite	5
3. Securite	6
OWASP Top 10	6
Audit de securite generique	6
Audit des dependances	6
Detection de secrets	7
4. Git Workflow avec Claude Code	7
Conventional Commits	7
Workflow de commit	8
Creation de PR avec gh	8
Code review assistee	9
5. Cost Management	9
Suivi de consommation	9
Optimisation des tokens	9
Choix du modele selon la tache	10
Regles d'optimisation	10
Analytics	11
Exercice Pratique (15min)	11
TDD avec Claude Code (15min)	11
Verification	11
Points Cles a Retenir	11

Module 8 : Qualite et Securite (1h)

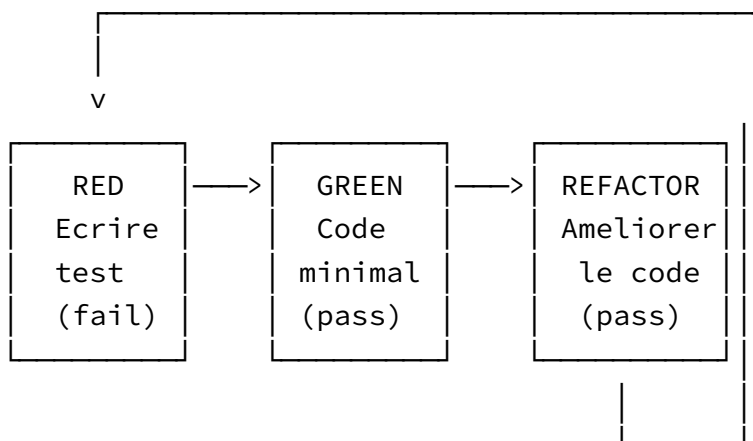
Objectifs

A la fin de ce module, vous serez capable de : - Appliquer le cycle TDD (Red-Green-Refactor) avec Claude Code - Realiser des audits de code automatises - Identifier et corriger les vulnerabilites OWASP Top 10 - Utiliser les conventional commits et creer des PRs avec Claude Code - Gerer les couts et optimiser la consommation de tokens

1. TDD/BDD avec Claude Code

Le Cycle Red-Green-Refactor

Le TDD (Test-Driven Development) est une discipline de developpement ou les tests sont ecrits **avant** le code. Claude Code excelle dans ce workflow car il peut generer les tests a partir de specifications, puis implementer le code minimal pour les faire passer.



Workflow TDD avec Claude

Etape 1 - RED : Demander les tests

"Ecris les tests pour un service de calcul de TVA.

Criteres :

- TVA standard 20%
- TVA reduite 5.5% pour l'alimentaire
- TVA 0% pour les exportations
- Arrondi a 2 decimales"

-> Claude genere les tests qui echouent (aucun code n'existe)

Etape 2 - GREEN : Demander l'implementation

"Implemente le TaxCalculatorService pour faire passer ces tests.
Code minimal uniquement."

-> Claude ecrit le code le plus simple qui passe les tests

Etape 3 - REFACTOR : Demander l'amelioration

"Refactorise le TaxCalculatorService en appliquant les principes SOLID.
Les tests doivent continuer a passer."

-> Claude ameliore le code tout en gardant les tests verts

Generation de tests

Claude Code peut generer differents types de tests :

Type	Commande typique	Usage
Unitaires	"Ecris les tests unitaires pour cette classe"	Logique metier isolee
Integration	"Ecris un test d'integration pour ce endpoint"	Interactions entre composants
Edge cases	"Ajoute les tests pour les cas limites"	Robustesse
Regression	"Ecris un test qui reproduit ce bug"	Prevention recidive

Couverture de code

> Lance les tests avec couverture et identifie les fichiers sous 80%

Claude :

- Execute la commande de couverture appropriee
- Identifie les fichiers non couverts
- Propose des tests supplementaires pour les zones non couvertes

BDD avec Gherkin

Claude Code peut generer des scenarios BDD au format Gherkin :

> Ecris les scenarios BDD pour le processus d'inscription utilisateur

Claude genere :

Feature: Inscription utilisateur

En tant que visiteur
Je veux creer un compte
Afin d'accéder aux fonctionnalités

Scenario: Inscription réussie

Given je suis sur la page d'inscription
When je remplis le formulaire avec des données valides
And je soumetts le formulaire
Then mon compte est créé
And je reçois un email de confirmation

Scenario: Email déjà utilisé

Given un utilisateur avec l'email "test@example.com" existe
When je m'inscris avec l'email "test@example.com"
Then je vois un message d'erreur "Cet email est déjà utilisé"
And aucun compte n'est créé

Scenario: Mot de passe trop faible

Given je suis sur la page d'inscription
When je saisis le mot de passe "123"
Then je vois un message d'erreur sur la force du mot de passe

2. Audit de Code

Analyse d'architecture

Claude Code peut analyser l'architecture globale d'un projet :

> Analyse l'architecture de ce projet. Vérifie :

1. La séparation des couches (présentation, application, domaine, infrastructure)
2. Les dépendances entre modules
3. Les violations de SOLID
4. Les couplages forts

Claude :

- Lit la structure du projet
- Analyse les imports et dépendances
- Identifie les violations architecturales
- Propose un rapport structure avec scores

Detection d'anti-patterns

Anti-pattern	Description	Impact
God Class	Classe avec trop de responsabilites	Maintenabilite
Feature Envy	Methode qui utilise plus les donnees d'une autre classe	Couplage
Shotgun Surgery	Modification necessitant des changements partout	Fragilite
Long Method	Methode > 20 lignes	Lisibilite
Data Clump	Groupes de donnees toujours passes ensemble	Abstraction manquante
Magic Numbers	Valeurs numeriques en dur	Maintenabilite

- > Cherche les anti-patterns dans le module `src/services/`.
Pour chaque anti-pattern trouve, donne :
- Le fichier et la ligne
 - Le type d'anti-pattern
 - La severite (critique, majeur, mineur)
 - La correction recommandee

Metriques de complexite

- > Calcule les metriques de complexite :
- Complexite cyclomatique (cible < 10 par methode)
 - Nombre de lignes par classe (cible < 200)
 - Nombre de parametres par methode (cible < 4)
 - Profondeur d'imbrication (cible < 3)
 - Nombre de dependances par classe (cible < 7)

Claude :

- Analyse chaque fichier
- Calcule les metriques
- Identifie les fichiers hors cible
- Recommande des refactorings

3. Securite

OWASP Top 10

Claude Code peut auditer votre code pour les 10 vulnerabilites les plus critiques :

#	Vulnerabilite	Ce que Claude verifie
1	Broken Access Control	Verification des permissions a chaque endpoint
2	Cryptographic Failures	Algorithmes obsoletes, secrets en dur
3	Injection	SQL injection, command injection, XSS
4	Insecure Design	Absence de rate limiting, validation insuffisante
5	Security Misconfiguration	Debug en production, headers manquants
6	Vulnerable Components	Dependances avec CVE connues
7	Authentication Failures	Sessions faibles, MFA absent
8	Data Integrity Failures	Signatures manquantes, CI/CD non securise
9	Logging Failures	Evenements securite non logges
10	SSRF	URLs non validees, acces reseau interne

Audit de securite generique

> Effectue un audit de securite OWASP Top 10 sur ce projet.
Pour chaque categorie, verifie les points suivants et donne un statut (OK/KO) :

1. Access Control : Les endpoints verifient-ils les permissions ?
2. Crypto : Les mots de passe sont-ils hashes correctement ?
3. Injection : Les inputs sont-ils valides et les requetes parametrees ?
4. Design : Y a-t-il du rate limiting sur les endpoints sensibles ?
5. Misconfiguration : Le mode debug est-il desactive en production ?
6. Components : Les dependances ont-elles des vulnerabilites connues ?
7. Auth : Les sessions expirent-elles ? MFA sur les acces critiques ?
8. Integrity : Les dependances sont-elles verifiees (checksums) ?
9. Logging : Les evenements securite sont-ils logges ?
10. SSRF : Les URLs utilisateur sont-elles validees ?

Audit des dependances

> Verifie les dependances du projet pour les vulnerabilites connues.
Utilise les outils disponibles (npm audit, pip audit, composer audit, etc.)
et liste les CVE trouvees par severite.

Claude :

- Execute l'outil d'audit adapte au langage
- Parse les resultats
- Priorise par severite (critical > high > medium > low)
- Propose les mises a jour necessaires

Detection de secrets

> Recherche dans le codebase des secrets qui ne devraient pas etre commites :

- Cles API en dur
- Mots de passe dans le code
- Tokens d'accès
- URLs de bases de données avec credentials
- Cles privées

Claude :

- Analyse les fichiers source
- Recherche les patterns de secrets (regex + sémantique)
- Vérifie que .gitignore couvre les fichiers sensibles
- Vérifie que .env.example ne contient pas de vraies valeurs

4. Git Workflow avec Claude Code

Conventional Commits

Claude Code genere des messages de commit au format Conventional Commits :

Type	Description	Exemple
feat	Nouvelle fonctionnalité	feat(auth): add JWT token generation
fix	Correction de bug	fix(cart): correct discount calculation
docs	Documentation	docs(readme): update installation steps
refactor	Refactoring	refactor(user): extract validation logic

Type	Description	Exemple
test	Tests	test(auth): add edge cases for login
perf	Performance	perf(query): add index on created_at
ci	CI/CD	ci: add lint step to pipeline
chore	Maintenance	chore: update dependencies
style	Formatage	style: apply prettier formatting
build	Build system	build: upgrade to Node 20

Workflow de commit

> Regarde les modifications en cours et cree un commit avec un message Conventional Commits adapte.

Claude :

1. git status pour voir les fichiers modifies
2. git diff pour analyser les changements
3. git log pour voir le style des commits precedents
4. Propose un message de commit adapte
5. Execute le commit apres validation

Creation de PR avec gh

Claude Code utilise la CLI gh (GitHub CLI) pour creer des Pull Requests :

> Cree une Pull Request pour la branche actuelle.

Claude :

1. Verifie les fichiers modifies et les commits
2. git diff main...HEAD pour voir tous les changements
3. Redige un titre et une description de PR
4. Execute gh pr create avec le titre et le body
5. Retourne l'URL de la PR creee

Code review assistee

> Effectue une code review du diff de cette branche par rapport a main.
Verifie :

- Architecture et principes SOLID
- Qualite du code (KISS, DRY, YAGNI)
- Couverture de tests
- Securite
- Performance
- Documentation

Format : liste de commentaires par fichier avec severite.

5. Cost Management

Suivi de consommation

La commande `/cost` affiche la consommation de la session en cours :

`/cost`

Resultat :

Session actuelle :

Input tokens: 45,230

Output tokens: 12,780

Total cost: \$0.42

Historique (dernieres 24h) :

Total: \$3.85

Optimisation des tokens

Strategie	Economie	Description
<code>/clear</code> entre taches	30-50%	Evite l'accumulation de contexte inutile
Sub-agents pour la recherche	20-40%	Isole les explorations du contexte principal
Prompts precis	15-25%	Moins d'allers-retours

Strategie	Economie	Description
Fichiers specifiques	10-20%	Lire uniquement ce qui est necessaire
Plan mode	Variable	Explorer avant d'agir

Choix du modele selon la tache

Modele	Cout relatif	Tache recommandee
Haiku 4.5	\$ (\$1/\$5 par M tokens)	Taches simples, hooks prompt-based, formatage
Sonnet 4.6	(3/15 par M tokens) Usage quotidien, architecture complexe, agents, refactoring *Opus4.6* \$ (\$5/\$25 par M tokens)	Architecture complexe, agents, refactoring majeur
Opus 4.6 Fast	\$\$\$\$ (\$30/\$150 par M tokens)	Urgences, generation rapide (6x le prix)

Regles d'optimisation

1. MODELE ADAPTE

- Bug simple ? Sonnet suffit
- Refactoring 50 fichiers ? Opus justifie
- Hook de validation ? Haiku 4.5 est parfait

2. CONTEXTE PROPRE

- /clear entre taches non liees
- Sub-agents pour les explorations
- Ne pas lire 20 fichiers "au cas ou"

3. PROMPTS EFFICACES

- Preciser le scope (fichier, module, ligne)
- Donner le resultat attendu
- Eviter les descriptions vagues

4. BOUCLES COURTES

- Fournir des tests pour la verification
- Eviter les allers-retours inutiles
- Valider incrementalement

Analytics

Surveillez votre consommation pour identifier les patterns couteux :

Indicateurs a suivre :

- Cout moyen par session
 - Cout par tache (feature, bugfix, review)
 - Ratio input/output tokens
 - Nombre de compactions par session (signe de contexte surcharge)
 - Temps moyen par tache
-

Exercice Pratique (15min)

TDD avec Claude Code (15min)

Realisez un cycle TDD complet avec Claude Code :

1. **RED** (5min) : Demandez a Claude de generer les tests pour un service de validation de mot de passe :
 - Minimum 12 caracteres
 - Au moins 1 majuscule, 1 minuscule, 1 chiffre, 1 caractere special
 - Pas dans une liste de mots de passe courants
 - Pas de sequences repetitives (aaa, 111)
2. **GREEN** (5min) : Demandez a Claude d'implementer le service pour faire passer les tests.
3. **REFACTOR** (5min) : Demandez a Claude de refactoriser en appliquant le pattern Strategy pour chaque regle de validation.

Verification

- ☐ Les tests sont ecrits AVANT le code
 - ☐ Tous les tests passent apres l'implementation
 - ☐ Le refactoring ne casse aucun test
 - ☐ Le pattern Strategy est correctement applique
 - ☐ Chaque regle de validation est dans sa propre classe
-

Points Cles a Retenir

1. **TDD avec Claude** : toujours demander les tests en premier (RED), puis l'implementation (GREEN), puis le refactoring

2. **Les audits de code** couvrent architecture, anti-patterns, complexite et securite
 3. **OWASP Top 10** est le standard de reference pour les audits de securite
 4. **Conventional Commits** et gh pr create standardisent le workflow Git
 5. **Le cout se gere** : modele adapte, contexte propre, prompts precis
 6. **Haiku 4.5 pour le simple, Sonnet 4.6 pour le quotidien, Opus 4.6 pour le complexe**
 7. **Les boucles de verification** (tests, screenshots, outputs attendus) multiplient la qualite par 2-3x
 8. **/cost** pour surveiller la consommation en temps reel
-

Duree : 1h Prochain module : Module 9 : Bonus – Claude Craft