

Module 5 — Projet Existant

Jour 2 — Intégrer Claude Code dans un projet legacy

The Bearded Bear

Février 2026



Table des matières

Module 5 : Intégrer Claude-Craft sur un Projet Existant	2
Objectifs	2
1. Audit Initial du Projet	2
Vue d'ensemble des commandes d'audit	2
/symfony :check-architecture	2
/symfony :check-code-quality	3
/symfony :check-security	3
/symfony :check-testing	4
/symfony :check-compliance	4
2. Interprétation des Rapports	5
Niveaux de sévérité	5
Priorisation	5
Exemple de plan de remédiation	5
3. Stratégie d'Adoption Progressive	6
Principe fondamental	6
Approche recommandée	6
Strangler Fig Pattern	6
Coexistence Legacy + Clean Architecture	7
4. Migration Progressive vers Clean Architecture	7
Étape 1 : Identifier les Bounded Contexts	7
Étape 2 : Extraire le Domain	7
Étape 3 : Créer les Interfaces (Ports)	8
Étape 4 : Implémenter les Adapters	8
5. Atelier : Audit Projet Réel	8
Exercice	8
Template de rapport	9
Points Clés à Retenir	10

Module 5 : Intégrer Claude-Craft sur un Projet Existant

Objectifs

À la fin de ce module, vous serez capable de : - Auditer un projet Symfony existant avec Claude-Craft - Élaborer une stratégie d'adoption progressive - Gérer la dette technique avec Claude - Migrer progressivement vers Clean Architecture

1. Audit Initial du Projet

Vue d'ensemble des commandes d'audit

Commande	Focus	Output
<code>/symfony:check-architecture</code>	Structure, couches, dépendances	Score architecture
<code>/symfony:check-code-quality</code>	PHPStan, CS-Fixer, complexité	Rapport qualité
<code>/symfony:check-security</code>	OWASP Top 10, vulnérabilités	Alertes sécurité
<code>/symfony:check-testing</code>	Couverture, qualité des tests	Rapport tests
<code>/symfony:check-compliance</code>	Audit global complet	Rapport consolidé

`/symfony:check-architecture`

Vérifie la conformité avec Clean/Hexagonal Architecture :

```
/symfony:check-architecture
```

```
# Output exemple:
# [=] AUDIT ARCHITECTURE
#
# Score global: 45/100 [!]
#
# [OK] Points positifs:
# - Séparation Controller/Service présente
# - Entités dans src/Entity/
#
# [KO] Violations:
# - src/Service/OrderService.php accède directement à Doctrine (ligne 45)
```

```
# - src/Controller/UserController.php contient de la logique métier (ligne  
  ↪ 78-120)  
# - Pas d'interface pour les repositories  
# - Couplage fort: 12 services dépendent de l'implémentation concrète  
#  
# [>] Recommandations:  
# 1. Extraire la logique métier dans le Domain  
# 2. Créer des interfaces pour les repositories  
# 3. Implémenter le pattern Command/Handler
```

/symfony:check-code-quality

Analyse la qualité du code :

`/symfony:check-code-quality`

```
# Output exemple:  
# [#] AUDIT QUALITE CODE  
#  
# PHPStan Level 5: 23 erreurs  
# PHP-CS-Fixer: 156 violations de style  
#  
# Complexité cyclomatique:  
# - OrderService::process() = 15 (max recommandé: 10) [!]  
# - UserController::register() = 12 (max: 10) [!]  
#  
# Méthodes trop longues (> 20 lignes):  
# - PaymentService::handlePayment() = 87 lignes  
# - ReportGenerator::generate() = 156 lignes  
#  
# Code dupliqué détecté:  
# - src/Service/EmailService.php:45-67  
# - src/Service/NotificationService.php:23-45
```

/symfony:check-security

Audit de sécurité OWASP :

`/symfony:check-security`

```
# Output exemple:  
# [*] AUDIT SECURITE  
#  
# Score: 65/100 [!]  
#  
# [CRIT] CRITIQUE:  
# - SQL Injection potentielle: src/Repository/ProductRepository.php:34  
#   `$this->em->createQuery("... WHERE name = '$name'")`
```

```
#  
# [HIGH] ELEVE:  
# - XSS: src/templates/user/profile.html.twig:12  
#   `{{ user.bio|raw }}` sans sanitization  
#  
# [MED] MOYEN:  
# - CSRF: Formulaire sans token: src/Form/ContactType.php  
# - Headers sécurité manquants: X-Frame-Options, CSP  
#  
# [OK] BON:  
# - Authentification: JWT correctement configuré  
# - Password hashing: bcrypt utilisé
```

/symfony:check-testing

Analyse de la couverture de tests :

/symfony:check-testing

```
# Output exemple:  
# [~] AUDIT TESTS  
#  
# Couverture globale: 35% (cible: 80%) [KO]  
#  
# Par type:  
# - Unit tests: 45%  
# - Integration tests: 25%  
# - Functional tests: 15%  
#  
# Code non testé critique:  
# - OrderService (0%)  
# - PaymentGateway (0%)  
# - UserRegistration (12%)  
#  
# Tests fragiles détectés:  
# - ProductControllerTest::testList() dépend de l'ordre des données
```

/symfony:check-compliance

Audit global consolidé :

/symfony:check-compliance

```
# Combine tous les audits + rapport consolidé  
# avec priorisation des actions
```

2. Interprétation des Rapports

Niveaux de sévérité

Niveau	Symbole	Action
Critique	[CRIT]	Corriger immédiatement
Élevé	[HIGH]	Corriger cette semaine
Moyen	[MED]	Planifier dans le sprint
Faible	[LOW]	Backlog
Info	[i]	Amélioration optionnelle

Priorisation

MATRICE DE PRIORISATION		
IMPACT	Faible effort	Fort effort
Fort	QUICK WINS ✓ (Faire en 1er)	PROJETS MAJEURS (Planifier)
Faible	FILL-INS (Si temps)	ÉVITER (ROI faible)

Exemple de plan de remédiation

Plan de Remédiation - Sprint 1

Quick Wins (cette semaine)

1. [OK] Corriger SQL Injection ProductRepository (2h)
2. [OK] Ajouter |escape sur template user/profile (30min)
3. [OK] Activer CSRF sur ContactType (1h)

Projets Moyen Terme (ce mois)

1. [>] Extraire Domain layer pour Order (3j)
2. [>] Ajouter tests pour OrderService (2j)
3. [>] Réduire complexité PaymentService (1j)

Backlog

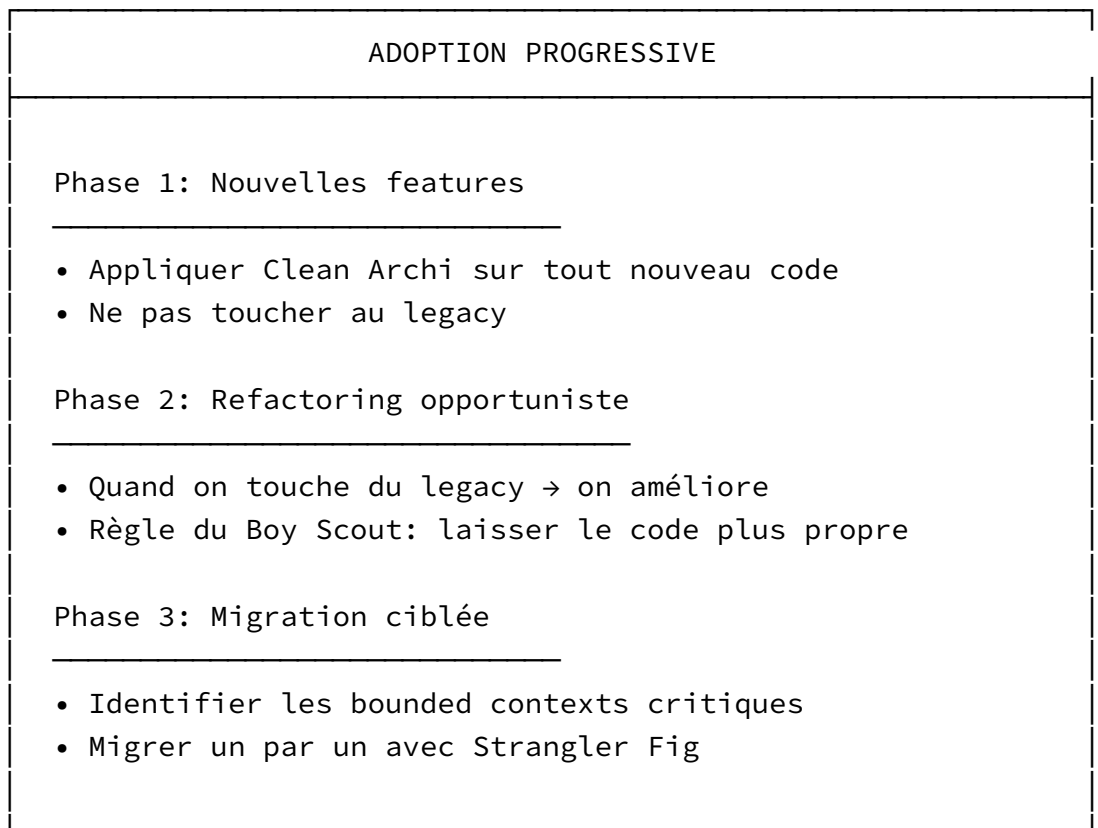
- Migrer vers Clean Architecture complète
- Atteindre 80% de couverture
- Implémenter CQRS

3. Stratégie d'Adoption Progressive

Principe fondamental

Pas de “Big Bang”! Migration progressive, feature par feature.

Approche recommandée



Strangler Fig Pattern

AVANT



→

APRÈS (progressif)





Coexistence Legacy + Clean Architecture

```
// Ancienne structure (coexiste)
src/
├── Entity/           # Legacy Doctrine entities
├── Repository/       # Legacy repositories
├── Service/          # Legacy services
└── Controller/       # Legacy controllers

// Nouvelle structure (nouvelles features)
├── Domain/
│   └── Order/        # Nouveau bounded context
├── Application/
│   └── Order/
├── Infrastructure/
│   └── Order/
├── UserInterface/
│   └── Api/Order/
```

4. Migration Progressive vers Clean Architecture

Étape 1 : Identifier les Bounded Contexts

```
@architecture-decision "Identifie les bounded contexts de ce projet legacy"
```

```
# Claude analyse le code et propose:
# - User (authentification, profil)
# - Order (commandes, paiement)
# - Catalog (produits, catégories)
# - Notification (emails, push)
```

Étape 2 : Extraire le Domain

```
"Extrait le domaine Order du legacy vers Clean Architecture"
```

```
# Claude va:
# 1. Créer Domain/Order/Order.php (entité pure)
# 2. Créer Domain/Order/OrderRepositoryInterface.php
# 3. Adapter l'ancien OrderRepository pour implémenter l'interface
# 4. Migrer la logique métier de OrderService vers le Domain
```


Étape 3 : Créer les Interfaces (Ports)

// AVANT: Couplage direct

```
class OrderService
{
    public function __construct(
        private EntityManagerInterface $em,    // Couplage Doctrine
        private Mailer $mailer              // Couplage Mailer
    ) {}
}
```

// APRÈS: Inversion de dépendance

```
class OrderService
{
    public function __construct(
        private OrderRepositoryInterface $orderRepository, // Interface
        private NotifierInterface $notifier                // Interface
    ) {}
}
```

Étape 4 : Implémenter les Adapters

// Infrastructure/Persistence/DoctrineOrderRepository.php

```
class DoctrineOrderRepository implements OrderRepositoryInterface
{
    public function __construct(
        private EntityManagerInterface $em
    ) {}

    public function findById(OrderId $id): ?Order
    {
        // Implémentation Doctrine
    }
}
```

5. Atelier : Audit Projet Réel

Exercice

Prendre un projet existant de l'équipe et :

1. Exécuter l'audit complet

```
cd /chemin/vers/projet
claude
/symfony:check-compliance
```

2. Analyser les résultats

- Identifier les 3 issues critiques
- Identifier les 3 quick wins

3. Élaborer le plan de remédiation

- Court terme (cette semaine)
- Moyen terme (ce mois)
- Long terme (backlog)

4. Corriger un quick win

- Choisir l'issue la plus simple
- Corriger avec l'aide de Claude
- Vérifier avec un nouvel audit

Template de rapport

```
# Rapport d'Audit - [Nom du Projet]
```

```
## Scores
```

Critère	Score	Cible
Architecture	/100	70+
Qualité code	/100	80+
Sécurité	/100	90+
Tests	%	80%+

```
## Issues Critiques
```

- [Description] - [Fichier:ligne]
- ...

```
## Quick Wins
```

- [Action] - Effort: [Xh] - Impact: [Élevé/Moyen]
- ...

```
## Plan de Remédiation
```

```
### Sprint 1
```

```
- [ ] ...
```

```
### Sprint 2
```

```
- [ ] ...
```

```
### Backlog
```

```
- [ ] ...
```

Points Clés à Retenir

1. **Audit avant action** : Toujours commencer par un diagnostic
 2. **Priorisation** : Quick wins d'abord, projets majeurs planifiés
 3. **Pas de Big Bang** : Migration progressive, bounded context par bounded context
 4. **Strangler Fig** : Le nouveau code enveloppe progressivement le legacy
 5. **Coexistence** : Legacy et Clean Archi peuvent cohabiter
-

Durée estimée : 1h30 **Prochain module** : Qualité et Sécurité