

MÉTHODOLOGIE DE DÉVELOPPEMENT AVEC CONFORMITÉ

CAHIER DES CHARGES

Guide complet pour Cursor/Windsurf

1. STRUCTURE DE PROJET RECOMMANDÉE

1.1 Architecture des dossiers

```
cabinet-management-app/
├── docs/
│   ├── cahier-charges/
│   │   ├── specifications.md      # Cahier des charges principal
│   │   ├── user-stories/         # Stories détaillées par module
│   │   ├── acceptance-criteria/  # Critères d'acceptation
│   │   └── test-scenarios/        # Scénarios de test
│   └── config/
│       ├── requirements.json      # Exigences techniques structurées
│       ├── validation-rules.json  # Règles de validation automatique
│       └── compliance-checklist.json # Check-list de conformité
├── src/
│   ├── modules/                  # Code par module métier
│   ├── shared/                   # Composants partagés
│   └── tests/                     # Tests de conformité
├── tools/
│   ├── compliance-checker/        # Outils de vérification
│   ├── requirements-tracker/      # Suivi des exigences
│   └── automated-testing/         # Tests automatisés
└── scripts/
    ├── validate-compliance.js     # Script de validation
    ├── generate-reports.js        # Génération de rapports
    └── update-requirements.js      # Mise à jour des exigences
```

1.2 Configuration initiale pour Cursor/Windsurf

Fichier `.cursorrules` (pour Cursor)

yaml

Règles de conformité au cahier des charges

rules:

- enforce_requirements_traceability
- validate_user_stories_implementation
- check_acceptance_criteria_coverage
- ensure_technical_specifications_compliance

requirements_path: "./docs/cahier-charges/specifications.md"

validation_config: "./config/validation-rules.json"

test_coverage_minimum: 85%

code_standards:

- follow_modular_architecture
- implement_security_requirements
- maintain_performance_standards
- ensure_accessibility_compliance

Fichier de configuration Windsurf

json

```
{
  "projectType": "enterprise-web-app",
  "specifications": {
    "source": "./docs/cahier-charges/specifications.md",
    "validationRules": "./config/validation-rules.json",
    "complianceChecks": "./config/compliance-checklist.json"
  },
  "development": {
    "enforceRequirements": true,
    "autoValidation": true,
    "continuousTesting": true
  }
}
```

2. DÉCOMPOSITION DU CAHIER DES CHARGES

2.1 Conversion en User Stories structurées

Template de User Story

markdown

US-[MODULE]-[NUMERO] : [TITRE]

Description

En tant que [ROLE], je veux [FONCTIONNALITÉ] afin de [BÉNÉFICE]

Critères d'acceptation

- [] Critère 1 (Ref: CDC Section X.Y.Z)
- [] Critère 2 (Ref: CDC Section X.Y.Z)
- [] Critère 3 (Ref: CDC Section X.Y.Z)

Définition of Done

- [] Code implémenté et revu
- [] Tests unitaires passants (>85% coverage)
- [] Tests d'intégration validés
- [] Validation conformité cahier des charges
- [] Documentation mise à jour
- [] Démonstration fonctionnelle

Références CDC

- Section: [X.Y.Z]
- Exigences: [REQ-001, REQ-002]
- KPI associés: [Performance, Sécurité, UX]

Tests de validation

```
````javascript
// Tests automatisés de conformité
describe('US-CRM-001: Création client', () => {
 it('should respect CDC requirements REQ-001', () => {
 // Test de conformité spécifique
 });
});
```

### ### 2.2 Fichier de mapping exigences

```
`config/requirements.json`
```json
{
  "modules": {
    "crm": {
      "requirements": [
        {
          "id": "REQ-CRM-001",
          "title": "Fiche client enrichie",
          "description": "Gestion complète des informations client",
          "cdcSection": "3.1.1",
          "priority": "HIGH",
          "userStories": ["US-CRM-001", "US-CRM-002"],
          "testScenarios": ["TEST-CRM-001", "TEST-CRM-002"],
          "acceptanceCriteria": [
            "Champs obligatoires renseignés",
            "Validation données SIRET",
            "Historique modifications tracé"
          ],
          "technicalSpecs": {
            "performance": "< 2s response time",
            "security": "Data encryption required",
            "validation": "SIRET format validation"
          }
        }
      ]
    }
  }
}
```

3. OUTILS DE VÉRIFICATION AUTOMATIQUE

3.1 Script de validation de conformité

`tools/compliance-checker/validator.js`


```
/**
```

```
 * Valideur de conformité au cahier des charges
```

```
 */
```

```
class ComplianceValidator {
```

```
  constructor(requirementsPath, configPath) {
```

```
    this.requirements = require(requirementsPath);
```

```
    this.config = require(configPath);
```

```
    this.violations = [];
```

```
  }
```

```
  // Vérification de la couverture des exigences
```

```
  validateRequirementsCoverage() {
```

```
    const implementedRequirements = this.scanImplementedFeatures();
```

```
    const missingRequirements = this.findMissingRequirements(implementedRequirements);
```

```
    if (missingRequirements.length > 0) {
```

```
      this.violations.push({
```

```
        type: 'MISSING_REQUIREMENTS',
```

```
        items: missingRequirements,
```

```
        severity: 'HIGH'
```

```
      });
```

```
    }
```

```
  }
```

```
  // Vérification des critères d'acceptation
```

```
  validateAcceptanceCriteria() {
```

```
    this.requirements.modules.forEach(module => {
```

```
      module.requirements.forEach(req => {
```

```
        const testResults = this.runAcceptanceTests(req.id);
```

```
        if (!testResults.allPassed) {
```

```
          this.violations.push({
```

```
            type: 'ACCEPTANCE_CRITERIA_FAILED',
```

```
            requirement: req.id,
```

```
            failedCriteria: testResults.failed,
```

```
            severity: 'HIGH'
```

```
          });
```

```
        }
```

```
      });
```

```
    });
```

```
  }
```

```
  // Vérification des spécifications techniques
```

```
  validateTechnicalSpecs() {
```

```
    const performanceTests = this.runPerformanceTests();
```

```
    const securityTests = this.runSecurityTests();
```

```
if (!performanceTests.passed) {
  this.violations.push({
    type: 'PERFORMANCE_VIOLATION',
    details: performanceTests.details,
    severity: 'MEDIUM'
  });
}

// Génération du rapport de conformité
generateComplianceReport() {
  return {
    timestamp: new Date().toISOString(),
    overallCompliance: this.calculateComplianceScore(),
    violations: this.violations,
    recommendations: this.generateRecommendations(),
    nextSteps: this.suggestNextSteps()
  };
}
```

3.2 Intégration avec les AI Code Editors

Prompt système pour Cursor/Windsurf

CONTEXTE PROJET:

Tu développes une application de gestion de cabinet selon un cahier des charges précis.

RÈGLES OBLIGATOIRES:

1. Avant chaque implémentation, vérifie la conformité avec `./docs/cahier-charges/specifications.md`
2. Chaque feature doit correspondre à une User Story documentée
3. Implémente les critères d'acceptation définis
4. Respecte les spécifications techniques (performance, sécurité, UX)
5. Génère les tests de conformité automatiquement

WORKFLOW DE DÉVELOPPEMENT:

1. Analyser l'exigence dans le cahier des charges
2. Identifier la User Story correspondante
3. Vérifier les critères d'acceptation
4. Implémenter en respectant les specs techniques
5. Générer les tests de validation
6. Exécuter le script de vérification de conformité

VÉRIFICATIONS AUTOMATIQUES:

- Performance: temps de réponse < 2s
- Sécurité: chiffrement, authentification, audit trail
- UX: responsive design, accessibilité
- Code quality: coverage > 85%, standards respectés

FICHIERS DE RÉFÉRENCE:

- Cahier des charges: `./docs/cahier-charges/specifications.md`
- Exigences: `./config/requirements.json`
- Règles validation: `./config/validation-rules.json`
- Check-list: `./config/compliance-checklist.json`

Avant de commencer à coder, demande toujours:

"Quelle section du cahier des charges implémentons-nous ?"

4. PROCESSUS DE DÉVELOPPEMENT ITÉRATIF

4.1 Workflow de développement

Étape 1: Planification Sprint

bash

Script de planification automatique

npm run sprint-planning

Sélection des User Stories prioritaires

Vérification des dépendances

Estimation de l'effort

Validation de la faisabilité technique

Étape 2: Développement avec validation continue

bash

Pre-commit hooks

git add .

git commit -m "feat: implement US-CRM-001 client creation"

Hooks automatiques:

1. Validation syntaxe et standards

2. Exécution tests unitaires

3. Vérification conformité CDC

4. Mise à jour traçabilité exigences

Étape 3: Validation et tests

bash

Tests de conformité complets

npm run compliance-check

Tests de performance

npm run performance-test

Tests de sécurité

npm run security-audit

Génération rapport de conformité

npm run generate-compliance-report

4.2 Scripts NPM configurés

package.json

json

```
{
  "scripts": {
    "dev": "npm run validate-requirements && next dev",
    "build": "npm run full-compliance-check && next build",
    "test": "jest --coverage --watchAll=false",
    "test:compliance": "node tools/compliance-checker/run-validation.js",
    "test:performance": "lighthouse-ci autorun",
    "test:security": "npm audit && node tools/security-scanner.js",
    "validate-requirements": "node tools/requirements-tracker/validate.js",
    "compliance-check": "npm run test:compliance && npm run test:performance && npm run test:security",
    "full-compliance-check": "npm run compliance-check && npm run validate-requirements",
    "generate-compliance-report": "node tools/compliance-checker/generate-report.js",
    "sprint-planning": "node tools/requirements-tracker/plan-sprint.js"
  }
}
```

5. TESTS DE CONFORMITÉ AUTOMATISÉS

5.1 Tests par module métier

Tests CRM (exemple)

javascript

```
// tests/compliance/crm.compliance.test.js
describe('Module CRM - Conformité CDC', () => {
  describe('REQ-CRM-001: Fiche client enrichie', () => {
    it('should validate SIRET format according to CDC 3.1.1', async () => {
      const client = await createClient({ siret: '12345678901234' });
      expect(client.siret).toMatch(/^\\d{14}$/);
      // Référence CDC: Section 3.1.1 - Informations générales
    });

    it('should maintain audit trail as per CDC security requirements', async () => {
      const client = await createClient(validClientData);
      const auditLog = await getAuditTrail(client.id);
      expect(auditLog).toHaveProperty('createdBy');
      expect(auditLog).toHaveProperty('timestamp');
      // Référence CDC: Section 9.3.1 - Audit trail complet
    });
  });

  describe('REQ-CRM-002: Segmentation intelligente', () => {
    it('should auto-segment clients according to CDC criteria', async () => {
      const client = await createClient({ ca: 5000000 });
      await runSegmentationAlgorithm();
      const updatedClient = await getClient(client.id);
      expect(updatedClient.segment).toBeDefined();
      // Référence CDC: Section 3.1.2 - Segmentation automatique
    });
  });
});
```

5.2 Tests de performance

javascript

// tests/performance/response-time.test.js

```
describe('Performance Requirements - CDC Section 10.1', () => {  
  it('should respond within 2 seconds for 95% of requests', async () => {  
    const requests = Array(100).fill().map(() =>  
      makeRequest('/api/clients')  
    );  
  
    const responses = await Promise.all(requests);  
    const responseTimes = responses.map(r => r.responseTime);  
    const percentile95 = getPercentile(responseTimes, 95);  
  
    expect(percentile95).toBeLessThan(2000);  
    // Référence CDC: Section 10.1 - Performance et scalabilité  
  });  
});
```

6. TABLEAUX DE BORD DE CONFORMITÉ

6.1 Dashboard de suivi en temps réel

`tools/compliance-dashboard/dashboard.js`

javascript

```
/**
 * Dashboard de conformité temps réel
 */
class ComplianceDashboard {
  generateMetrics() {
    return {
      requirementsCoverage: this.calculateCoverage(),
      testsPassing: this.getTestResults(),
      performanceMetrics: this.getPerformanceData(),
      securityScore: this.getSecurityScore(),
      codeQuality: this.getCodeQualityMetrics(),
      cdcCompliance: this.calculateCDCCompliance()
    };
  }

  calculateCDCCompliance() {
    const totalRequirements = this.getTotalRequirements();
    const implementedRequirements = this.getImplementedRequirements();
    const validatedRequirements = this.getValidatedRequirements();

    return {
      implementation: (implementedRequirements / totalRequirements) * 100,
      validation: (validatedRequirements / totalRequirements) * 100,
      overall: (validatedRequirements / totalRequirements) * 100
    };
  }
}
```

6.2 Rapports automatiques

Template de rapport de conformité

markdown

RAPPORT DE CONFORMITÉ CDC

Date: {{date}}

Version: {{version}}

Sprint: {{sprint}}

RÉSUMÉ EXÉCUTIF

- **Conformité globale**: {{overallCompliance}}%
- **Exigences implémentées**: {{implementedReqs}}/{{totalReqs}}
- **Tests passants**: {{passingTests}}%
- **Performance**: {{performanceScore}}

DÉTAIL PAR MODULE

{{#modules}}

Module {{name}}

- Conformité: {{compliance}}%
- Exigences: {{implemented}}/{{total}}
- Violations: {{violations}}

{{/modules}}

ACTIONS REQUISES

{{#violations}}

- **{{type}}**: {{description}}
- Priorité: {{severity}}
- Action: {{recommendation}}

{{/violations}}

PROCHAINES ÉTAPES

{{#nextSteps}}

- {{step}}

{{/nextSteps}}

7. INTÉGRATION CI/CD AVEC VALIDATION

7.1 Pipeline GitHub Actions

.github/workflows/compliance-check.yml

name: Compliance Validation

on: [push, pull_request]

jobs:

compliance-check:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3
- name: Setup Node.js
uses: actions/setup-node@v3
with:
node-version: '18'
- name: Install dependencies
run: npm ci
- name: Validate Requirements Coverage
run: npm run validate-requirements
- name: Run Compliance Tests
run: npm run test:compliance
- name: Performance Testing
run: npm run test:performance
- name: Security Audit
run: npm run test:security
- name: Generate Compliance Report
run: npm run generate-compliance-report
- name: Upload Report
uses: actions/upload-artifact@v3
with:
name: compliance-report
path: reports/compliance-report.html
- name: Comment PR with Results
if: github.event_name == 'pull_request'
uses: actions/github-script@v6
with:
script: |
const report = require('./reports/compliance-summary.json');
github.rest.issues.createComment({


```
issue_number: context.issue.number,  
owner: context.repo.owner,  
repo: context.repo.repo,  
body: `## Conformité CDC: ${report.overallCompliance}%
```

- ☒ Exigences: $\frac{\text{\${report.implementedReqs}}}{\text{\${report.totalReqs}}}$
- ☒ Tests: $\text{\${report.passingTests}}\%$
- ☒ Performance: $\text{\${report.performanceScore}}$

```
 $\text{\${report.violations.length > 0 ? ' ⚠️ Violations détectées - voir rapport complet' : ' ☒ Aucune violation'}}$   
));
```

8. COMMANDES PRATIQUES POUR CURSOR/WINDSURF

8.1 Prompts optimisés pour le développement

Prompt de démarrage de feature

Je vais implémenter [FEATURE_NAME] selon le cahier des charges.

Analyse d'abord:

1. Section CDC correspondante: [SECTION]
2. User Stories impactées: [US_IDS]
3. Critères d'acceptation: [CRITERIA]
4. Spécifications techniques: [TECH_SPECS]

Génère le code en respectant:

- Architecture modulaire définie
- Standards de sécurité (chiffrement, auth, audit)
- Performance (< 2s response time)
- Tests de conformité automatiques

Vérifie la conformité avec: `npm run compliance-check`

Prompt de refactoring

Refactor [COMPONENT_NAME] en maintenant la conformité CDC.

Vérifications obligatoires:

1. Fonctionnalités existantes préservées
2. Critères d'acceptation toujours validés
3. Performance maintenue ou améliorée
4. Sécurité renforcée si possible
5. Tests mis à jour

Exécute après refactor: `npm run full-compliance-check`

8.2 Scripts de productivité

`scripts/dev-helper.js`


```
#!/usr/bin/env node
```

```
/**
```

```
 * Assistant de développement avec validation CDC
```

```
 */
```

```
const { Command } = require('commander');
```

```
const program = new Command();
```

```
program
```

```
  .command('start-feature <featureName>')
```

```
  .description('Démarrer le développement d\'une nouvelle feature')
```

```
  .action((featureName) => {
```

```
    console.log(`Démarrage feature: ${featureName}`);
```

```
    // 1. Vérifier les exigences CDC
```

```
    const requirements = checkCDCRequirements(featureName);
```

```
    console.log('Exigences CDC:', requirements);
```

```
    // 2. Créer la branche
```

```
    execSync(`git checkout -b feature/${featureName}`);
```

```
    // 3. Générer les templates de tests
```

```
    generateTestTemplates(featureName, requirements);
```

```
    // 4. Ouvrir les fichiers pertinents
```

```
    openRelevantFiles(featureName);
```

```
  });
```

```
program
```

```
  .command('validate-feature <featureName>')
```

```
  .description('Valider la conformité d\'une feature')
```

```
  .action((featureName) => {
```

```
    console.log(`Validation feature: ${featureName}`);
```

```
    // Exécution des tests de conformité
```

```
    const results = runComplianceTests(featureName);
```

```
    displayResults(results);
```

```
    if (results.allPassed) {
```

```
      console.log('✅ Feature conforme au CDC');
```

```
    } else {
```

```
      console.log('❌ Violations détectées');
```

```
      console.log('Actions requises:', results.actions);
```

```
    }
```

```
  });
```

```
program.parse();
```

9. SURVEILLANCE CONTINUE ET MÉTRIQUES

9.1 Monitoring de conformité

Configuration de monitoring

javascript

```
// config/monitoring.js
module.exports = {
  compliance: {
    thresholds: {
      requirementsCoverage: 95,
      testCoverage: 85,
      performanceScore: 90,
      securityScore: 95
    },
    alerts: {
      email: ['team-lead@cabinet.com'],
      slack: '#dev-alerts',
      frequency: 'daily'
    },
    reports: {
      weekly: true,
      monthly: true,
      beforeRelease: true
    }
  }
};
```

9.2 Métriques de vélocité et qualité

javascript

// tools/metrics/velocity-tracker.js

```
class VelocityTracker {  
  trackImplementation() {  
    return {  
      requirementsPerSprint: this.getRequirementsVelocity(),  
      defectRate: this.calculateDefectRate(),  
      reworkRate: this.calculateReworkRate(),  
      cdcComplianceScore: this.getCDCComplianceScore(),  
      timeToCompliance: this.getTimeToCompliance()  
    };  
  }  
}
```

10. BONNES PRATIQUES ET RECOMMANDATIONS

10.1 Do's et Don'ts

✓ À FAIRE

- Toujours partir du cahier des charges pour toute nouvelle feature
- Maintenir la traçabilité entre code et exigences
- Exécuter les tests de conformité avant chaque commit
- Documenter les décisions techniques et leur impact CDC
- Mettre à jour les tests quand les exigences évoluent

✗ À ÉVITER

- Implémenter des features non spécifiées dans le CDC
- Modifier les critères d'acceptation sans validation
- Ignorer les alertes de performance ou sécurité
- Déployer sans validation complète de conformité
- Laisser des exigences non implémentées sans justification

10.2 Checklist de déploiement

markdown

Checklist Déploiement Conforme CDC

Pré-déploiement

- ☐ Toutes les exigences du sprint sont implémentées
- ☐ Tests de conformité à 100% passants
- ☐ Performance validée (< 2s response time)
- ☐ Sécurité auditée et validée
- ☐ Documentation mise à jour
- ☐ Rapport de conformité généré

Déploiement

- ☐ Validation en environnement de staging
- ☐ Tests de non-régression passants
- ☐ Sauvegarde des données effectuée
- ☐ Plan de rollback préparé

Post-déploiement

- ☐ Monitoring de conformité activé
- ☐ Métriques de performance surveillées
- ☐ Feedback utilisateurs collecté
- ☐ Rapport post-déploiement rédigé

Cette méthodologie garantit que chaque ligne de code développée respecte le cahier des charges, avec une validation automatique et continue. Elle s'intègre parfaitement avec Cursor et Windsurf pour un développement efficace et conforme.