🔍 CODE REVIEW & QA PROCESS - Agent Zero v1

🎯 Code Review & Quality Assurance System

Mission: Zapewnić wysoką jakość kodu poprzez systematyczny proces review, testing i quality gates

Team Standards: 100% PRs reviewed, 90%+ test coverage, zero critical security issues

---

🔄 CODE REVIEW WORKFLOW

📋 1. Pre-Review Checklist (Developer)

Przed utworzeniem Pull Request, developer musi:

✅ Code Quality

- [ ] Kod formatowany z `black shared/`
- [ ] Linting passed z `ruff shared/` (zero errors)
- [ ] Type checking passed z `mypy shared/`
- [ ] Wszystkie tests pass locally
- [ ] No console.log/print statements (except logging)
- [ ] Code follows naming conventions

✅ Testing

- [ ] Unit tests napisane dla nowych funkcji
- [ ] Integration tests dla API endpoints
- [ ] Manual testing wykonane

- [ ] Performance nie pogorszona
- [ ] Cross-browser testing (dla frontend)

✅ Documentation

- [ ] Docstrings dla nowych funkcji
- [ ] README updated jeśli potrzeba
- [ ] API documentation generated
- [ ] Comments dla complex logic

✅ Security

- [ ] No hardcoded secrets lub credentials
- [ ] Input validation dla external data
- [ ] SQL injection prevention
- [ ] XSS protection (frontend)

🔍 2. Review Process

PR Creation Requirements:

```
## 🎯 What
Description czego zostało zmienione

## 🎯 Why
Business reasoning dla zmiany

## 🎯 How
Technical implementation details

## ✅ Testing
- [ ] Unit tests added/updated
- [ ] Manual testing completed
- [ ] Performance tested
```

```
## 📈 Screenshots
(Jeśli frontend changes)

## 🔗 Related Issues
Links do Notion tasks lub GitHub issues
```

Reviewer Responsibilities:

Developer A Reviews Developer B PRs:

- Focus: Frontend quality, UX/UI, React best practices
- Timeline: Within 24 hours
- Feedback: Constructive, specific, actionable

Developer B Reviews Developer A PRs:

- Focus: Backend architecture, API design, database interactions
- Timeline: Within 24 hours
- Feedback: Technical depth, scalability concerns

Review Checklist (Reviewer):

🔍 **Code Quality:**

- [ ] Code is readable i maintainable
- [ ] No code duplication
- [ ] Appropriate abstractions used
- [ ] Error handling implemented
- [ ] Performance considerations addressed

🔍 **Architecture:**

- [ ] Follows established patterns
- [ ] No architecture violations
- [ ] Dependencies are appropriate
- [ ] Separation of concerns maintained

🔍 **Security:**

- ☐ No obvious security vulnerabilities
- ☐ Input validation present
- ☐ Authentication/authorization correct

🔍 **Testing:**

- ☐ Adequate test coverage
- ☐ Tests are meaningful
- ☐ Edge cases considered
- ☐ Integration points tested

🏁 DEFINITION OF DONE (DoD)

📋 Feature Complete Criteria:

Development:

- ☐ Code implements all acceptance criteria
- ☐ Code follows team conventions
- ☐ No TODO comments w production code
- ☐ Error handling implemented
- ☐ Logging added dla debugging

Testing:

- ☐ Unit tests written (90%+ coverage)
- ☐ Integration tests dla APIs
- ☐ Manual testing completed
- ☐ Performance acceptable (< 2s response)
- ☐ Security testing done

Code Review:

- [ ] PR created z proper description
- [ ] Code reviewed by team member
- [ ] All review comments addressed
- [ ] PR approved by reviewer

Documentation:

- [ ] API documentation updated
- [ ] Technical documentation updated
- [ ] User-facing docs updated (if needed)
- [ ] Release notes prepared

Deployment:

- [ ] CI/CD pipeline passes
- [ ] Deployment tested in staging
- [ ] Rollback plan prepared
- [ ] Monitoring alerts configured

🧪 TESTING STRATEGY

🎯 Testing Pyramid

```
      /\       E2E Tests (10%)
     /  \      - Full user journeys
    /    \     - Critical path testing
   /      \    - Cross-browser testing
  /________\
 /          \  Integration Tests (30%)
/            \ - API endpoint testing
/              \ - Database interactions
/________________\ - Service communication
```

```
    Unit Tests (60%)
    - Individual functions
    - Component testing
    - Mock external dependencies
```

🔧 Testing Tools & Frameworks

Backend Testing:

```
# pytest dla unit i integration tests
pytest shared/communication/ -v --cov=shared

# httpx dla API testing
import httpx
async def test_api_endpoint():
    async with httpx.AsyncClient() as client:
        response = await client.get("/api/agents")
        assert response.status_code == 200

# pytest-asyncio dla async code
import pytest
@pytest.mark.asyncio
async def test_async_function():
    result = await async_function()
    assert result is not None
```

Frontend Testing:

```
// Jest + React Testing Library
import { render, screen } from '@testing-library/react'
import Dashboard from './Dashboard'

test('renders agent status', () => {
  render(<Dashboard />)
  expect(screen.getByText('Agent Status')).toBeInTheDocument()
```

```
})

// Cypress dla E2E
cypress.run({
  spec: 'cypress/integration/dashboard.spec.js'
})
```

📈 Coverage Requirements

Minimum Coverage Targets:

- **Unit Tests**: 90% line coverage
- **Integration Tests**: 80% API endpoints
- **E2E Tests**: 100% critical user paths

Coverage Monitoring:

```
# Generate coverage report
pytest --cov=shared --cov-report=html

# Fail build if coverage < 90%
pytest --cov=shared --cov-fail-under=90
```

🚨 QUALITY GATES

🔴 Gate 1: Pre-Commit (Local)

Automated Checks:

```
# Pre-commit hook
#!/bin/bash
set -e

# Format code
black shared/
```

```
# Linting
ruff shared/

# Type checking
mypy shared/

# Fast tests only
pytest shared/communication/test_unit/ -x

echo "✅ Pre-commit checks passed"
```

Criteria: All checks must pass przed commit

🟡 Gate 2: Pull Request (CI)

Automated Pipeline:

```
# .github/workflows/pr-check.yml
name: PR Quality Check
on: pull_request

jobs:
  quality:
    runs-on: ubuntu-latest
    steps:
      - name: Code Quality
        run: |
          black --check shared/
          ruff shared/
          mypy shared/

      - name: Security Scan
        run: |
          bandit -r shared/
          safety check
```

```
    - name: Tests
      run: |
        pytest shared/ --cov=shared --cov-fail-under=90

    - name: Performance
      run: |
        python scripts/performance_test.py
```

Criteria: Pipeline musi być zielony dla merge

🟢 Gate 3: Pre-Production (Staging)

Manual Testing:

- [ ] Full regression testing
- [ ] Performance testing
- [ ] Security penetration testing
- [ ] User acceptance testing
- [ ] Load testing

Automated Checks:

- [ ] All E2E tests pass
- [ ] Performance benchmarks met
- [ ] Security scans clean
- [ ] Database migrations successful

🐛 BUG TRIAGE PROCESS

🏇 Bug Priority Levels

🔴 P0 - Critical (Fix within 4h)

- System down lub major functionality broken
- Data loss lub security breach

- Blocks all development work

🟡 **P1 - High (Fix within 24h)**

- Major feature not working
- Performance degradation > 50%
- Blocks significant user workflows

🟡 **P2 - Medium (Fix within 1 week)**

- Minor feature issues
- UI/UX problems
- Non-critical performance issues

🟢 **P3 - Low (Fix in next sprint)**

- Cosmetic issues
- Nice-to-have improvements
- Documentation problems

🔄 Bug Lifecycle

1. **Discovery** → **Triage** → **Assignment** → **Fix** → **Verification** → **Closed**

Triage Process (Daily @ 9:30 AM):

- Review new bugs
- Assign priority levels
- Assign to developer
- Estimate effort
- Update stakeholders

📈 PERFORMANCE BENCHMARKS

🎯 Performance Targets

API Response Times:

- **GET /agents**: < 200ms (95th percentile)
- **POST /agents/task**: < 500ms (95th percentile)
- **WebSocket connection**: < 100ms initial
- **LLM inference**: < 5s per request

Frontend Performance:

- **First Contentful Paint**: < 1s
- **Time to Interactive**: < 2s
- **Bundle size**: < 1MB gzipped

System Resources:

- **Memory usage**: < 2GB per service
- **CPU usage**: < 70% sustained
- **Database queries**: < 50ms average

🔧 Performance Testing Tools

```
# Load testing z locust
from locust import HttpUser, task, between

class AgentUser(HttpUser):
    wait_time = between(1, 3)

    @task
    def get_agents(self):
        self.client.get("/api/agents")

    @task(3)
    def create_task(self):
        self.client.post("/api/agents/task", json={
            "description": "Test task",
            "priority": 1
        })
```

🔄 REVIEW METRICS & KPIs

📈 Code Review Metrics

Quality Metrics:

- **Review Coverage**: 100% PRs reviewed
- **Review Time**: < 24h average
- **Rework Rate**: < 20% PRs require major changes
- **Defect Escape Rate**: < 5% bugs found post-merge

Process Metrics:

- **PR Size**: Average < 400 lines changed
- **Review Comments**: 2-5 comments per PR average
- **Approval Time**: < 2 days from creation

📉 Weekly Review Report Template

```
## Week XX Quality Report

### Metrics
- PRs Reviewed: X/X (100%)
- Average Review Time: Xh
- Critical Bugs Found: X
- Test Coverage: X%

### Highlights
- ✅ What went well
- 🚨 Issues identified
- 🎯 Improvements for next week

### Action Items
```

```
- [ ] Item 1 (Owner: X)
- [ ] Item 2 (Owner: Y)
```

🔧 TOOLS INTEGRATION

GitHub Integration:

- Branch protection rules
- Required status checks
- Automated PR templates
- Code owners file (CODEOWNERS)

IDE Integration:

- Pre-commit hooks
- Linting extensions
- Test runner integration
- Code coverage display

CI/CD Integration:

- Automated quality gates
- Performance regression detection
- Security scanning
- Deployment gates

Code Review Process Last Updated: 7 października 2025, 13:02 CEST

Next Review: Weekly during retrospective

Process Owner: Development Team