Rainer Stropek | time cockpit

Surviving C#-Codereviews

Your Host

Rainer Stropek

Developer, Entrepreneur

Azure MVP, MS Regional Director

Trainer at IT-Visions

Contact

software architects gmbh
rainer@timecockpit.com
Twitter: @rstropek

Agenda

Es ist nicht unüblich, dass externe oder interne Kunden Experten beauftragen, die C#-Codebasis eines Projekts zu begutachten. Rainer Stropek, langjähriger MVP und MS Regional Director, macht solche Projekte regelmäßig. In dieser Session erklärt er, worauf er dabei Wert legt.

- Welche Tools setzt er ein, um Verbesserungspotenziale zu suchen?
- Wie findet er in großen Codebasen rasch die kritischen Bereiche?
- Welche Best und Worst Practices kontrolliert er?
- Wie würde er ein C#-Projekt aufsetzen, das jeden Codereview glänzend bestehen soll?

Auf solche Fragen wird Rainer in der Session eingehen. Sie erhalten eine konkrete Checkliste von Punkten, die Sie bei Ihren Projekten beachten können, um die Qualität Ihres gelieferten C#-Codes noch weiter zu verbessern.

Agenda

- Introduction
  - Why Code Review Projects?
  - My Rules for Review Projects
- Reviewing the code
  - Basics
  - Coding guidelines
  - Code structure
  - Documentation
  - Testing
  - Performance
  - Security
- Reviewing the process
  - Automation
  - Source code handling
  - State of the art?
  - Development process
  - Team management
- Summary
  - Recommendations

Why Code Review Projects?

Why Code Review Projects?

M&A

Buy company or source of a product

New management team

Customer-vendor relationship

Tip: Make code review criteria part of contract

Large customers buys software strategic for their business

Frequently: Large customer, small vendor

Team wants/needs external advice

Reviewer is a kind of external coach

Accompanying Reviews

Security review

Legal reviews

- Who owns the source code?
- License terms of dependencies?
- Compliance to policies (internal/external)

Standard certifications

- E.g. ISO

My Rules for Review Projects

Review Culture

Prerequisits

Fair, based on partnership

- Anti-pattern: Find flaws to prove that something is wrong (e.g. reduce price)

Set realistic expectations

- Estimate necessary time based on # of e.g. LOCs, person years, technologies
- Clearly define scope (time-boxed vs. fixed scope)
- Documentation: Level of detail and responsibility

Clarify availability of experts

- Domain experts
- People familiar with the project and the code

Send checklist/requirements upfront – it depends…

Culture

Be objective

- Best/worst practices defined by vendors (e.g. Microsoft)
- Clearly state if something is a subjective opinion

Be realistic

- Every project has technical depts
- Every project as resource constraints

Be honest

- Be polite and appreciative, but be clear about weaknesses
- Don’t just talk about bad things, call out good practices, too

Review process

Macro- and micro-level

- Macro: E.g. Architectural aspects, technical roadmap
- Micro: E.g. code quality

Manual vs. automated checks

- Use tools to find examples for “dragons”, analyze them manually
- E.g. long/complex, missing docs, high/low test coverage, many dependencies
- Tool examples: NDepend, Visual Studio Code Metrics

Review process

Ask many questions

"Show me a piece of code that has recently been written and that you are proud of"
"Show me an example of technical dept"
"Show me a cricital piece of business logic and describe how it is tested and verified"
"Walk me through an important business process and describe how the software deals with it (layers, APIs, ...)"
"What are your top three problems"

Speak with many different people

E.g. Business stakeholders, Managers (technical, project), Developers, Testers
New team members and seniors
Theory vs. real everyday practice

Spend some time alone with the code

Can you understand the system and the code?

The Basics

Basics

Do you have the complete source code?

"Are you sure that this is the code the users are currently using?
How can you be sure?" → source code control, security, traceability

Does it compile? Warnings?

→ Completeness, very basic quality checks

Can you/we debug?

→ Staging, debugging capabilities

Basics

Can you/we run existing tests? Are they green?

→ Basic quality analysis of tests

Can we create a new release together?

→ Basic version management check

Coding guidelines

Goals, process

Find violations of best/good practices for C# code

- Practices defined by Microsoft
- Categories see e.g. Code Analysis for Managed Code Warnings

Goal: Make code more readable, maintainable, secure, etc.

Do

- …focus on important things
- …reference "official" guidelines (e.g. .NET Foundation Coding Guidelines)

Don't…

- …judge based on your (reviewer) personal coding style
- …spend too much time on less important coding aspects

Tools

Old, outdated: FxCop, StyleCop

Use Analyzers (e.g. StyleCop.Analyzers) instead

Visual Studio Code Analysis Tools

.NET Compiler Platform ("Roslyn") Analyzers

Analyzers on NuGet

Commercial 3rd party code analysis tools

E.g. ReSharper, NDepend

Tools

SonarQube with SonarLint for Visual Studio

- Blog: SonarAnalyzer for C#: The Rule Engine You Want to Use
- Tip: Ready-made Docker image (Docker Hub)
- Good Azure support (e.g. AAD, Azure SQL DB)
- SonarQube build tasks for TFS/VSTS (VS Marketplace)

Resources (not complete!)

Wikipedia: Software Quality

.NET Foundation Coding Guidelines

MSDN

- Analyzing Application Quality by Using Code Analysis Tools
- Framework Design Guidelines
- C# Programming Guide

.NET related papers

- Patterns for Parallel Programming
- Asynchronous Programming Patterns

Framework-related papers, books and articles

Code structure

Criteria

Solutions and project structure

Monolithic? Too fine grained? Fits to size of the solution?

NuGet for library distribution

Clear separation of layers and modules

Separation of Concerns

Over-engineering

Often ask "why?"

Question "read for"-statements (automated testing, patching DLLs) - YAGNI?

Criteria

KISS - write the obvious code first

- Look for premature optimizations, in particular parallel programming

Dependency management

- External dependencies
- Evaluation process for new dependencies
- Process for keeping dependencies up to date
- Isolation using DI/IoC

Documentation

Documentation

Start with a question

"Imagine I am completely new in your team, what do you give me to read?"
Documentation of processes, guidelines, architecture
Reduce dependency –
but remember: Working software over comprehensive documentation

Documentation history

How old are the chapters – screenshots and samples are revealing ;-)

Naming consistency

Language consistency
Important business terms (glossary?)

Documentation Types

Architectural and design documentation

Standards available (Wikipedia)

Code Documentation

C# XML Documentation
Inline code documentation
Tools: Sandcastle Help File Builder, DocFX

RESTful web APIs

Tools: Swagger, Swashbuckle, readme.io (commercial)

Conceptual documentation

Tools: Markdown, GitHub pages/Jekyll, MAML/SHFB, DocFX, Word ☹

Testing

Testing Checklist

Test types (not complete, see also Wikipedia)

- Unit Tests
- Integration Tests
- UI Automation Tests
- Manual Tests
- Tests by customers (e.g. previews)
- TiP (Test in Production)
- Performance tests

Are the tests automated?

- CI/CD
- „Can we run the tests now?“

Unit Tests

Do they exist?

What's important?

- Code coverage
- Assertations
- Documentation
- Code quality in tests
- Dependency Injection (many frameworks available, e.g. MEF, Unity)
- Mocking (many frameworks available, MS Fakes built into VS)
- Execution time

Performance

Performance

Questions

"Show me some code where you fight with perf problems"
"Describe an example for an optimization that you did based on a profiling session"
"Describe an example for an optimization that you did based on telemetry findings"
"Show me an example of performance optimization using parallel programming“

Use of profilers

CPU, Memory
Tools: PerfView, VS Profilers, 3rd party commercial profilers (e.g. ANTS)

Use of telemetry

Tools: VS Application Insights, 3rd party commercial tools (e.g. Dynatrace)

Security

Security

Dedicated security review out of scope

- However, code analysis contains security aspects

Examples for security basics covered

- SQL Injections
- Authentication and authorization in C# code
- OWASP Top 10
- Security-critical configuration values (e.g. connection strings)
- Handling of API keys, certificates etc.

Automation

Automation

Why?

- Agility
- Reduce dependency on certain persons
- Repeatable, consistent quality
- Security (build and deployment servers)
- Reduce costs

Categories

- Build (CI)
- Tests
- Relase (CD)
- Software distribution (e.g. installers, install scripts, containers, NuGet)

Automation

Staging process

- Environments
- Cost efficiency
- Representative test environment for integration and performance tests

Version management

- Standard: Semver

Source code Handling

Source code Handling

Source code control present?

- One system or many solutions?
- Integrated ALM solution?

Quality of source code management

- E.g. Quality of checkins, comments,
  links to other systems (e.g. backlog, support)
- Documented process (e.g. pull requests, reviews)
- Example: Contributing Code (.NET Foundation)

Security

State of the art?

Largely depends on the goals of the organization, here just some examples…

Technical Debts

What technical debts are present?

- Outdated code, technologies or standards
- Historical sins
- Outdated dependencies, dependencies without maintenance
- Technical road blockers for innovations (e.g. mobile)

Is the team aware of technical debts?

Is there a plan for overcoming technical depts?

- Technical road map
- Planned refactorings
- Part of backlog?

See also: https://en.wikipedia.org/wiki/Technical_debt

Cloud Readiness

Clusters

- Fail-over
- Load balancing

Ready for PaaS and/or containers?

Follow best practices of cloud vendors

- E.g. retry logic

Contract and access management

- Who owns the subscriptions?
- User and permission management

See also: https://en.wikipedia.org/wiki/Technical_debt

Process

Process

Specification

- UI, API
- Technical dept of specifications
- "Show me the specification of a recently developed work item and how it was implemented in code" (checkins)
- Implementation reflects specification

Project management methodology

- E.g. Scrum, Kanban

Process

Backlog

- Product backlog
- Sprint backlog
- Size of WIP

Done-done Checklist

- Existence
- Completeness

Technical debt management

- Code-level
- Strategic and architectural debts

Process

Resource management

- Estimations
- Resource allocation (new projects vs. maintenance vs. support)

Support

- Backflow to backlog (root-cause analysis)
- Tools: VSTS, Zendesk

Team

Distribute learnings within the team

- Root-cause analysis
- Retrospection
- Evolve guidelines and quality tools (more, reduce)

Trainings

Access to knowledge

- Video learning courses
- Books
- Internal/external workshops

Lighthouse/side projects, technical studies

Summary

Summary

Use ALM

Live DevOps

„You build it, you run it“

Automate as much as possible

„If something hurts, do it more often“

Embrace agile development

Get better every day

Thank you for coming!

Questions?