Software Development Best Practices

sebnozzi

Published with GitBook

Table of Contents

Software Development Best Practices

This will be a book collecting best-practices in software development.

It will range from basic code-syntax, going through clean-code, architectural questions, up to deployment, monitoring, maintenance.

This is a work in progress.

Inbox

- Working with legacy-code
- DRY
- NIH-syndrome
- Favor composition over inheritance
- De-couple
- Defer (in responsibility)
  - Let another part of the code deal with it
  - Let another type deal with it
- Defer (in time; be asynchronous)
- Choosing Libraries / Frameworks
  - Suitability
  - License
  - Project Activity
  - Popularity
  - Online Resources

Working environment

- Breaks
- Get enough sleep
- Productivity curve
- Pomodoro
- Interruptions
- Focus
- Meetings
- The daily stand-up.

Project management

- Agile
- Scrum
- Kanban

- Ticketing system

Codebase organisation

- Packaging (structure of codebase)
- Organize into modules
  - Better separation of concerns
  - Better parallelization in team
  - Better compilation / build times

Functional Programming

- Separate computations from side-effects
  - Isolate side-effects
- Favor immutability
  - Immutable collections
  - Immutable DTOs
  - Pure functions
  - Don't throw exceptions, return values
- Encapsulate null

Others

- Avoid null
- Too many parameters

Architecture

- Onion / Hexagon
- Separate the core from the drivers
- Domains
- Separate core from UI
- Proper UI
  - Generic, reusable, UI components
  - UI Controllers
  - Adapters

Working in Teams

- Static Code Analyzers

- Commit Hooks
- Continuous Integration
- Conventions for git comments
- Managing branches
    - gitflow
    - pull requests
- Continuous integration
- Versioning (e.g. semantic versioning)
- Deployment strategies
- Upgrade strategies
- Release management
- Schema migrations
- Data migrations
- Estimation strategies
- Configuration strategies
- Scalability
- Monitoring
- Security

General Principles

- Clean over confusing
- Simple over complex
- Explicit over implicit
- Coupling (loose / tight)
- Cohesion (similar things go together)
- Flexible over rigid
- Robust over brittle (tests)
- Deterministic over non-deterministic
- Small over big
- Homogeneous over heterogeneous
    - One over many (responsibilities)
- Less over more
- Precise over ambiguous
- Fast over slow (tests, feedback)
- Sooner over later (failing)
- Automatic over manual
- Continuous Improvement
- Don't repeat yourself

Clean Code

- Code duplication
- Tight coupling
- Dependency injection
  - Be specific on your needs
- Leaking Abstractions
- Don't mix responsibilities
  - Per function
  - Per class
- Code smell: lots of private functions
- Code smell: too large / too much
- Don't mix levels of abstraction
- Single Responsibility Principle
- Principle of least power
- YAGNI
  - Premature optimization
  - Premature generalization
  - Unneeded functionality
  - Unneeded complexity
- DRY
- Metric: lines of code
  - Lines of code SPENT

Communicating Meaning

- Short methods / functions
- Short classes / modules / files
- Meaningful names
- Avoid long invocation chains
  - Describe intermediate steps
- Avoid primitive types
  - Create domain types

Comments

- Consider what can change
- Don't document the obvious
- Describe top-level intent
- Maintainable comments
- Convert comments into units of code

Concurrency

- Avoid primitive concurrency
- Avoid polling
- Avoid race-conditions
- Be reactive
- Futures / Promises
- Actors
- Distributed computing

Testing

- Aim at max ROI / min TCO
- Code-coverage can be misleading
- Refactor often!
- Maintainability is king
- Isolate implementation details
- Mocking / stubbing
- Proper UI testing
  - Avoid
  - Test wiring
  - Test typical cases
  - Make them maintainable
- Unit-tests: DO NOT hit the DB!
- Brittle tests
- Testing pyramid
- Cost of tests
- Test kinds

Practice

- Examples from other disciplines
  - Musicians
  - Scientists
  - Athletes
  - Actors
- Practice!
- Katas
- Coding Dojos
- Code Retreats
- Focus on learning, not on finishing
- Meetups
- Pairing
- Contribute to projects
- Hobby projects
- Read code
- Learn
- Solve problems
- Teach
- Ask questions
- Answer questions