SOLID Software Design Principles in Java

SAVING THE DAY WITH SOLID

Dan Geabunea
PASSIONATE SOFTWARE DEVELOPER | BLOGGER
@romaniancoder www.romaniancoder.com

Overview

- Problems that appear when SOLID principles are not used
- Technical debt
- SOLID Principles and their benefits
- Brief description of the sample application

“It is not enough for code to work”

Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship

SOLID principles are the foundation on which we can build clean, maintainable architectures.

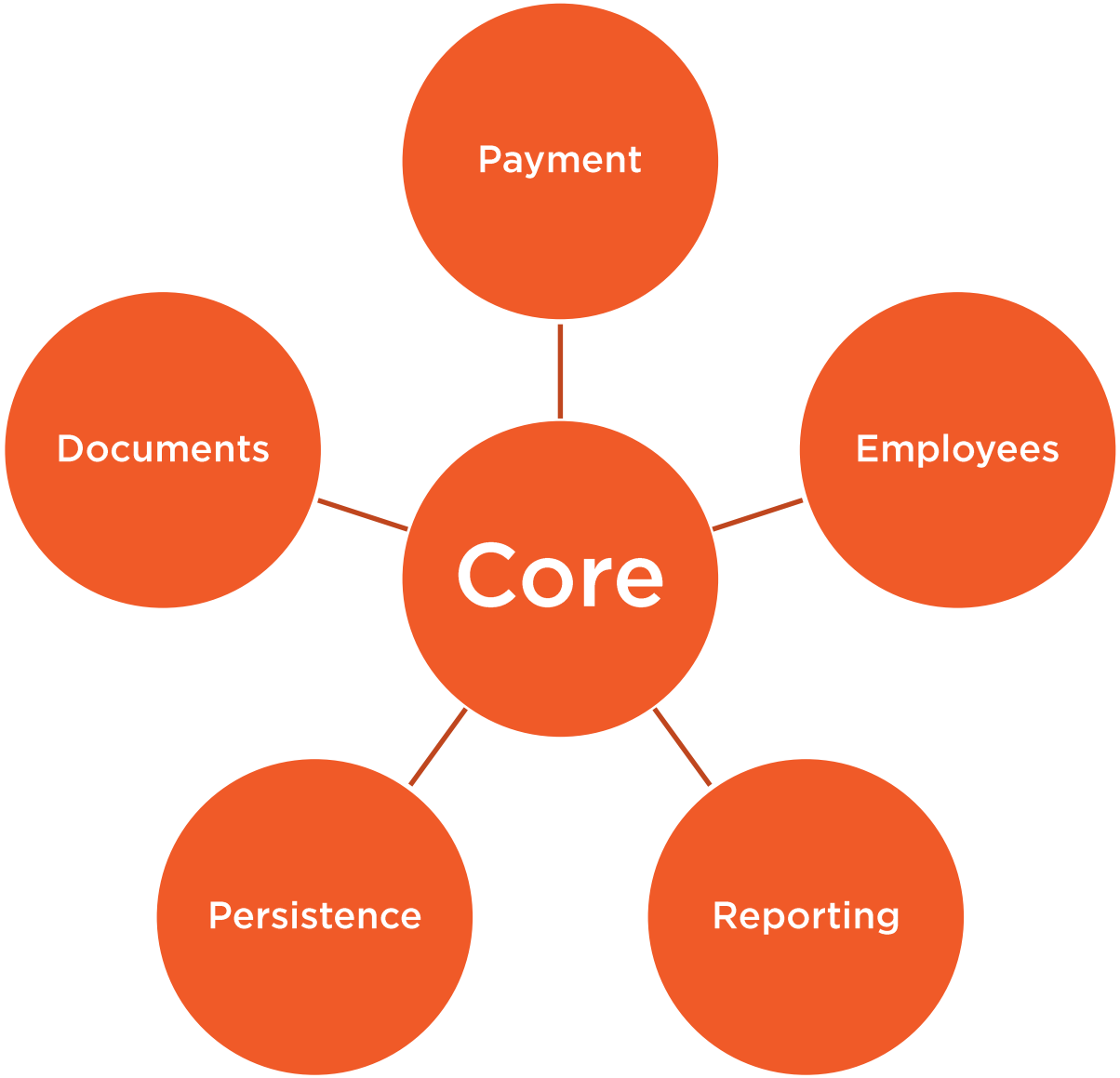
Problems That Appear When SOLID Principles Are Not Used

Change request: add new payment method

Implement change

Deploy application

Bugs in other sub-systems



Code Fragility

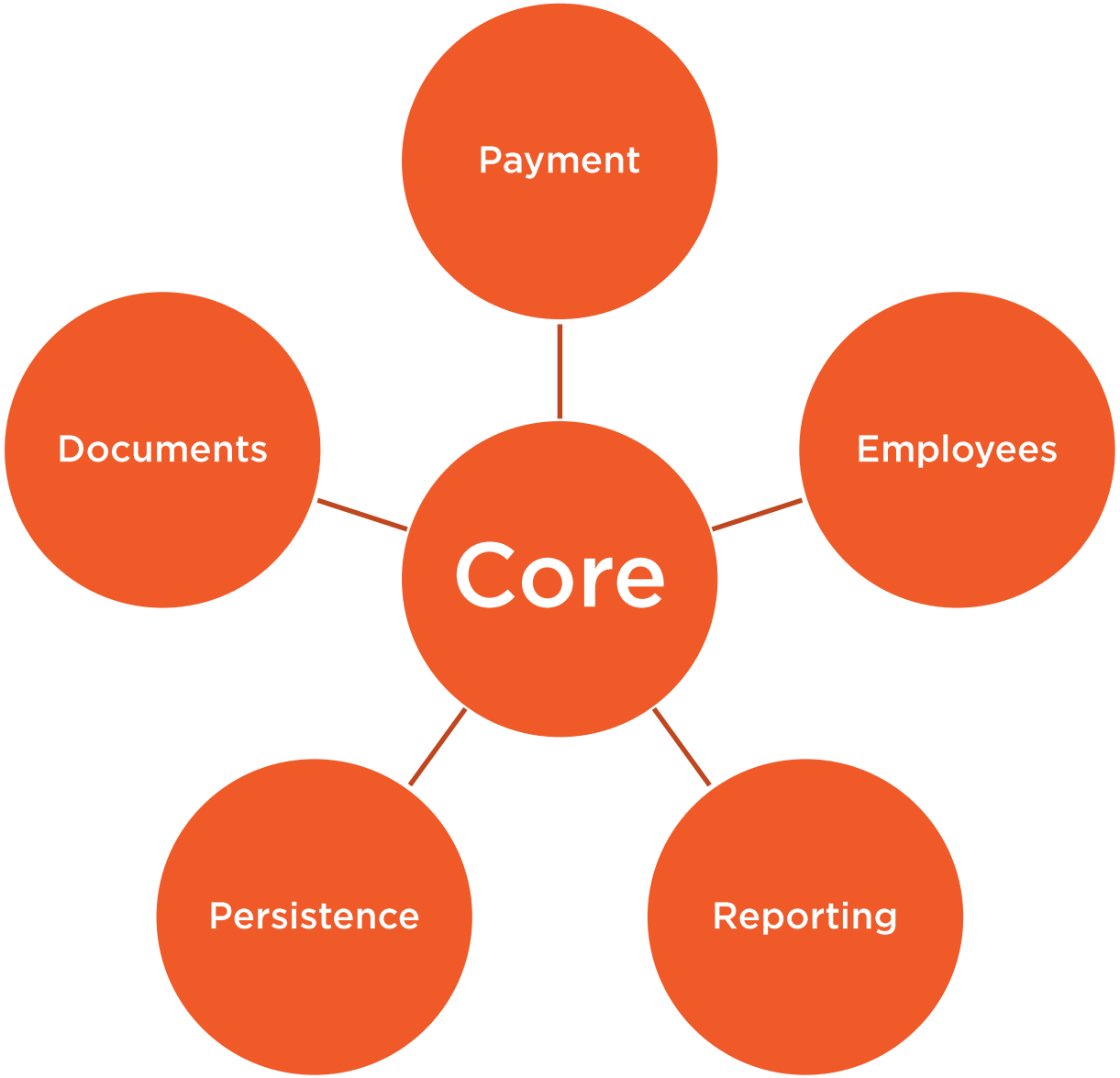
Fragility is the tendency of the software to break in many places every time it is changed.

- Robert C. Martin

Change request: update reports with new data

Start implementing change

We have to modify other parts of the system



Code Rigidity

Rigidity is the tendency for software to be difficult to change, even in simple ways. Every change causes a cascade of subsequent changes in dependent modules.

- Robert C. Martin

Fragility and rigidity are
symptoms of high technical debt.

Technical Debt

Technical Debt

The cost of prioritizing fast delivery over code quality for long periods of time.

The Choice You Have to Make

Fast delivery

Easiest fix/change

Fast

Poor written code

Code quality

Takes more time

Adds a bit of complexity

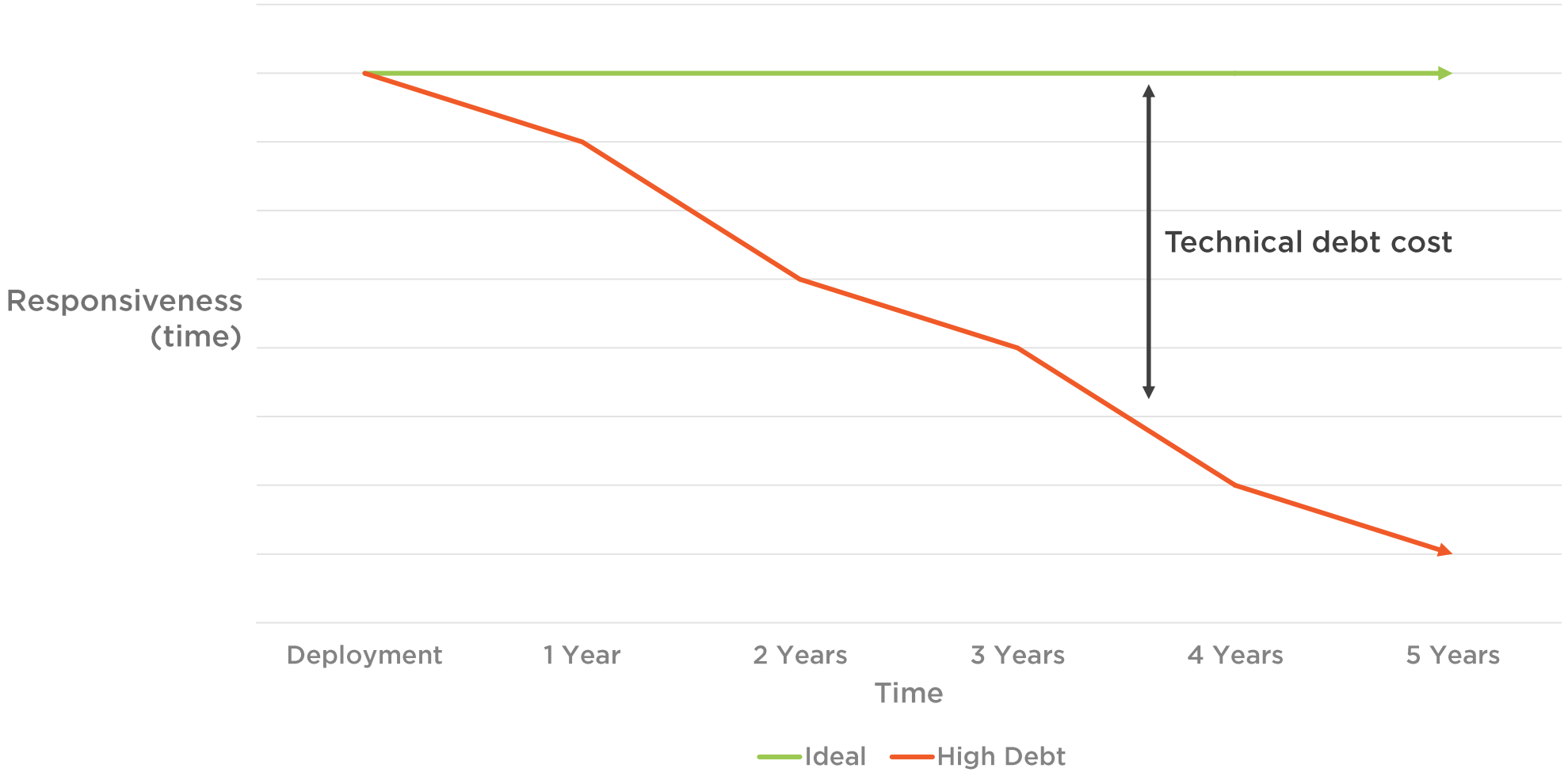
Maintainable

Cost of Change

Cost of Change

Customer Responsiveness

Customer Responsiveness



Technical Debt Facts

No matter how good the team is, technical debt will accumulate over time

Left uncontrolled, it will kill your project

The key is to keep it under control

Controlling Technical Debt

Write code

Pay debt (refactor)

Write more code

Pay debt (refactor)

SOLID

SOLID Principles

Acronym for 5 software design principles that help us to keep technical debt under control.

SOLID

Single Responsibility Principle

Open Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

Top Benefits of SOLID Code

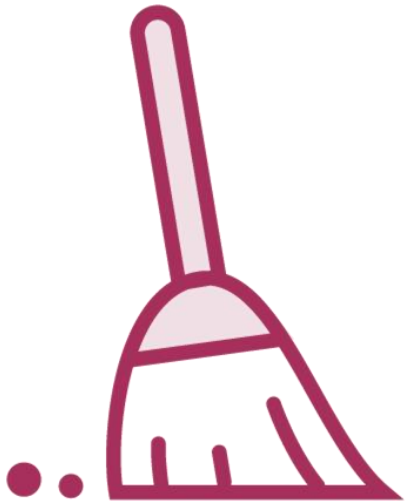
Easy to understand and reason about

Changes are faster and have a minimal risk level

Highly maintainable over long periods of time

Cost effective

Other Ways to Keep Your Architecture Clean



Constant refactoring

Design patterns

Unit testing (TDD)

Clean Code to the Next Level

Design Patterns in Java series
by Bryan Hansen

Test-driven Development
Practices in Java
by Mike Nolan

The Sample Application

Globomantics HR

- Employee management
- Tax calculation
- Pay slip generation
- Reporting

Console Based Application

```
java.exe …
PAYSLIP GENERATION
Payslip content Anna Smith; Days off 2; Income $1870
Sent via email to Anna.Smith@globomanticshr.com
#####
Payslip content Billy Lech; Days off 2; Income $860
Sent via email to Billy.Lech@globomanticshr.com
#####
Process finished with exit code 0
```

Summary

It is not enough to write code that works

How to control technical debt

SOLID principles to the rescue