Jasmine Lisondra

Chapter 11: Systems

How Would You Like To Build a City?

Building clean code is like building a city. Different teams manage different parts of the city or project, making sure everything runs smoothly.

Separate Constructing a System from Using It

Keep the process of building the software separate from its everyday use. This helps make the code easier to manage, update, and test by organizing object construction and dependencies better.

Scaling Up

Building software is like growing a city. Start with basic features and add more over time as new needs arise.

Java Proxies

Java proxies let you change how objects behave while the program is running. They can be complex and aren't always the best solution for every situation.

Pure Java AOP Frameworks

Pure Java AOP (Aspect-Oriented Programming) frameworks help you keep your business logic clean by handling common tasks separately. This makes your code easier to maintain.

Test Drive the System Architecture

Use test-driven development to keep your code flexible and adaptable. Focus on separating concerns and avoiding too much upfront design. This helps deliver value and adapt to changes more easily.

Optimize Decision Making

Modularity and decentralized decision-making are important in both software and large projects. Assign responsibilities to the right people and make decisions when you have the best information. This helps you respond to feedback and experience.

Use Standards Wisely, When They Add Demonstrable Value

Use standards in software development when they add real value. Standards can help with reusability and integration, but don't follow them blindly. Focus on practical needs and avoid trendy standards that don't actually help your project.

Systems Need Domain-Specific Languages

Domain-Specific Languages (DSLs) are specialized languages or APIs for specific areas of your software. They help bridge the gap between the technical and business sides, making the code clearer and reducing misunderstandings.

Chapter 12: Emergence

Getting Clean via Emergent Design

Follow Kent Beck's four rules to create well-designed software that evolves over time.

Simple Design Rule 1: Runs All the Tests

Make sure your system can run all its tests to verify it works correctly. Testable systems lead to smaller, single-purpose classes that follow the Single Responsibility Principle (SRP).

Simple Design Rules 2–4: Refactoring

After writing tests, continuously refactor the code. This means cleaning up the code to improve its design without breaking the tests. Refactoring helps maintain good software design principles like increasing cohesion and decreasing coupling.

No Duplication

Avoid duplicating code because it adds complexity and extra work. Remove identical or similar code to make your code cleaner and easier to maintain.

Expressive

Use clear and descriptive names for your variables, methods, and classes. This makes your code easier to read and understand.

Minimal Classes and Methods

Keep your classes and methods small and focused. Even though they are small, make sure they are concise and do one thing well.