|  |  |  |
| --- | --- | --- |
| Name | Category | Severity |
| Large Class | Class-level | 4 |
| Primitive Obsession | Bad practice | 5 |
| Switch Statements | Object-level | 5 |
| Refused Bequest | Sub-class level | 3 |
| Divergent Change | Class-level | 7 |
| Shotgun Surgery | Application-level | 6 |
| Comments | Bad practice | 3 |
| Feature Envy | Code-level | 5 |
| Middleman | Class-level | 3 |
| Lazy Class | Class-level | 5 |
| The Blob | Application-level | 6 |
| Dead End | Design-level | 4 |
| Data Clumps | Code-level | 4 |
| Spaghetti Code | Application-level | 7 |
| Boat Anchor | Design-level | 3 |
| Lava Flow | Application-level | 8 |
| Ambiguous Viewpoint | Design-level | 3 |
| Functional Decomposition | Application-level | 8 |
| Poltergeists | Application-level | 8 |
| Golden Hammer | Application-level | 7 |
| Mushroom Management | Management-level | 7 |

Code maintainability and simplicity are closely related because they go hand in hand in software development and beyond. If you want to make your code easily maintainable one of the best places to start is to improve your practices in ways that make your code easier to read and work with. In other words, the simpler you make your work the easier it’s going to be to go back and make changes, updates or work with the code in general. Better handling of coupling, readability and consistency throughout your work will increase your code’s maintainability.

Although that’s true that balancing runtime and efficiency is a challenge, it’s also true that they’re ways to make your programs fast and efficient. For example, they’re certain practices like staying away from deep nested loops and hard coding, taking advantage of good architecture when designing program, and simple things like proper indention and naming of variables. If you can take advantage of good practices like these you will find your projects be more consistently both simple and efficient.