SOLID Principles

Produced by:

Eamonn de Leastar (edeleastar@wit.ie)

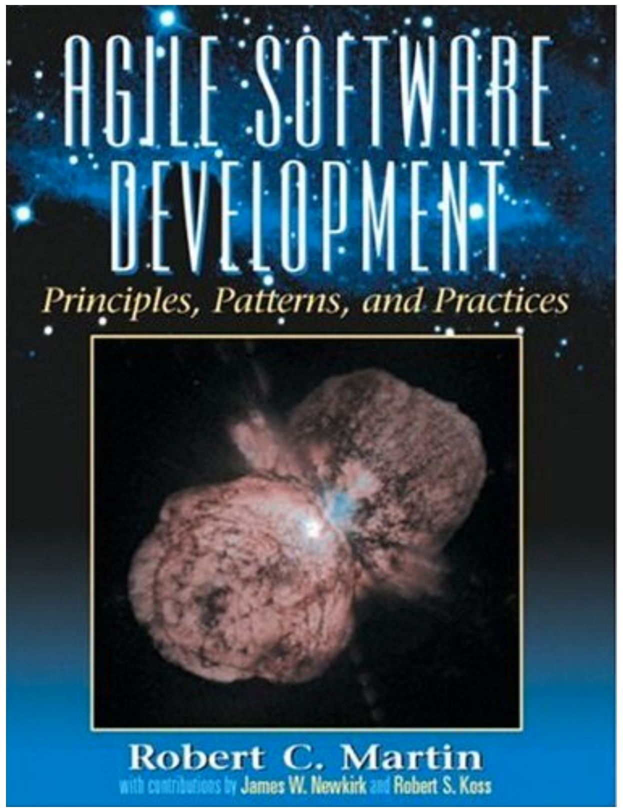
Dr. Siobhán Drohan (sdrohan@wit.ie)

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

SOLID Class Design Principles

In the mid-1990s, Robert C. Martin gathered five principles for object-oriented class design, presenting them as the best guidelines for building a maintainable object oriented system.


AGILE SOFTWARE DEVELOPMENT
Principles, Patterns, and Practices
Robert C. Martin
with contributions by James W. Newkirk and Robert S. Koss

Michael Feathers attached the acronym SOLID to these principles in the early 2000s.

SOLID Class Design Principles

S Single Responsibility Principle (SRP). Classes should have one, and only one, reason to change. Keep your classes small and single-purposed.

O Open-Closed Principle (OCP). Design classes to be open for extension but closed for modification; you should be able to extend a class without modifying it. Minimize the need to make changes to existing classes.

L Liskov Substitution Principle (LSP). Subtypes should be substitutable for their base types. From a client’s perspective, override methods shouldn’t break functionality.

I Interface Segregation Principle (ISP). Clients should not be forced to depend on methods they don’t use. Split a larger interface into a number of smaller interfaces.

D Dependency Inversion Principle (DIP). High-level modules should not depend on low-level modules; both should depend on abstractions. Abstractions should not depend on details; details should depend on abstractions.

SOLID Principles in Poster form…

http://blogs.msdn.com/b/cdndevs/archive/2009/07/15/the-solid-principles-explained-with-motivational-posters.aspx

SOLID

Software development is not a Jenga game.

Single Responsibility Principle

Just because you *can* doesn't mean you *should.*

“S” in SOLID - Single Responsibility Principle

Every object should have a single responsibility and all of its services should be aligned with that responsibility.

“Responsibility” is defined as “a reason to change”

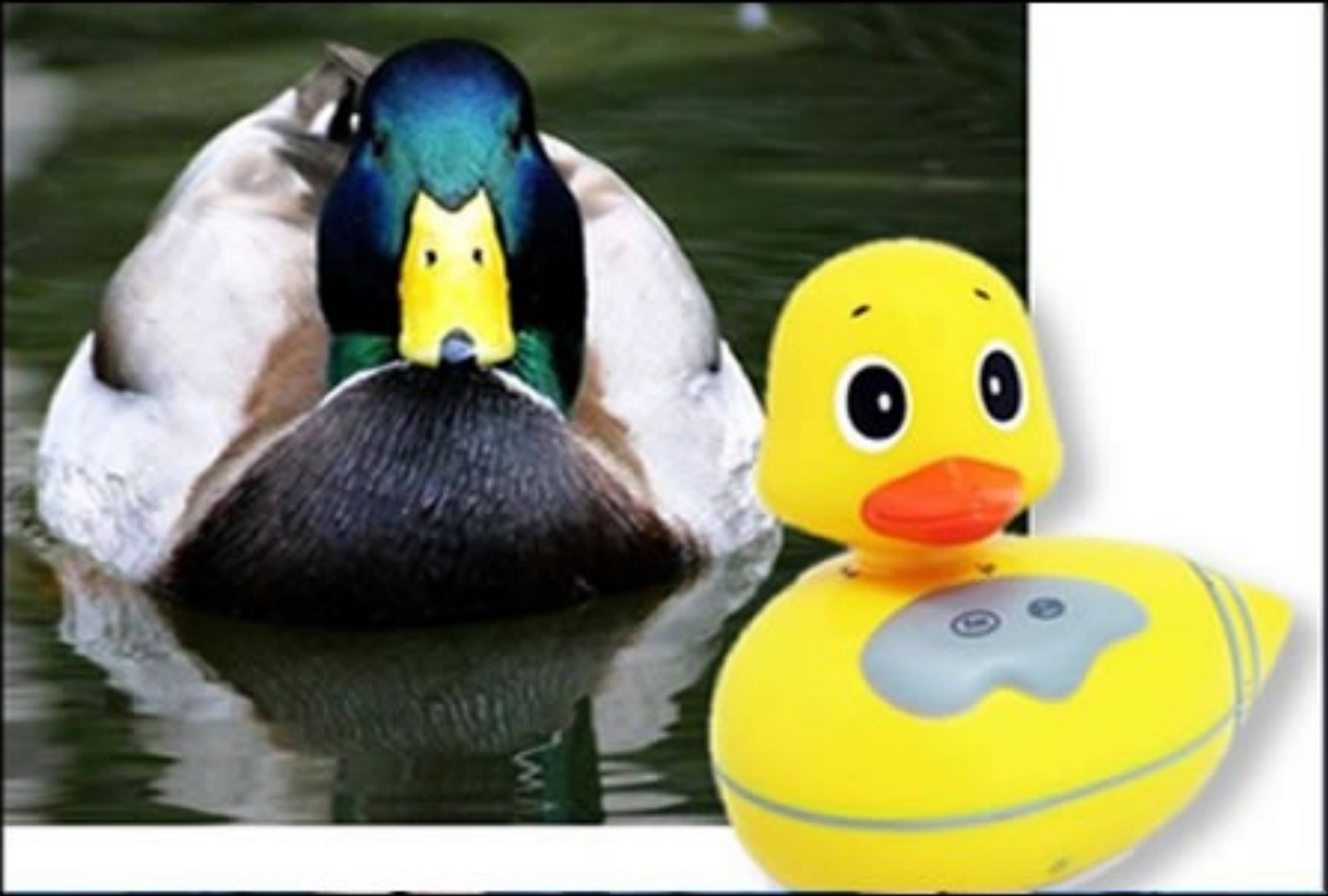
Open-Closed Principle

Open-chest surgery isn't needed when putting on a coat.

“O” in SOLID - Open-Closed Principle

Software entities – such as classes, modules, or functions should be open for extension but closed for modification.

Better to make changes to classes by adding to or building on them (using mechanisms like subclassing or polymorphism) rather than modifying their code.



Liskov Substitution Principle

If it looks like a duck and quacks like a duck but needs batteries, you probably have the wrong abstraction.

“L” in SOLID - Liskov Substitution Principle

Subclases should be substitutable for the classes from which they were derived.

For example, if ASCIITableConsole is a subclass of Console, you should be able to replace Console with ASCIITableConsole without any significant side effects.

Interface Segregation Principle

You want me to plug this in *where?*

“I” in SOLID - Interface Segregation Principle

Many client specific interfaces are better than one general purpose interface.

Make fine grained interfaces that are client specific.

DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

“D” in SOLID - Dependency Inversion Principle

High-level modules shouldn’t depend on low-level modules, but both should depend on shared abstractions.

In addition, abstractions should not depend on details – instead, details should depend on abstractions.

Depend on abstractions, not on concretions.

SOLID

Software development is not a Jenga game.