Secure Coding and OOAD

Software Design, Review, and Maintenance

Presented for OWASP March 21st 2012 by
Chris Arnold: Chris.Arnold@ventyx.abb.com; mysteryproducer@gmail.com

OWASP Top 10 Code Flaws

https://www.owasp.org/index.php/OWASP_Source_Code_Flaws_Top_10_Project_Index

- #1: Design Weaknesses
- #2: Architectural Weaknesses
- #8: Inappropriate API usage
- #9: Inadequate Documentation
- #10: everything else listed in…

OWASP Secure Coding Practices

- Quick Reference:
  https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf

Avoiding Security Flaws

- Threat analysis
- Requiring threat mitigation
- Satisfying requirements in code
- Code review
- Unit testing
- QA

Detection

- Code review
- Unit tests

Maintenance begins in the initial development cycle. Many security flaws (and most other defects) should never make it to QA.

OO Principles

- Inheritance
- Abstraction
- Polymorphism
- Encapsulation

This is a list of capabilities rather than a list of design or coding guidelines.

Robert C Martin's 5 Principles

http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

- Open-Closed Principle
- Single Responsibility Principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Liskov Substitution Principle

SOLID

Open-Closed Principle (OCP)

A module should be open for extension but closed for modification.

It is difficult to predict change requests, but the remaining 4 principles will help.

SOLID

Single Responsibility Principle (SRP)

http://www.objectmentor.com/resources/articles/srp.pdf

There should never be more than one reason for a class to change.

If you can state a method or class's function with a single clause, you pass.

Keep methods as simple as possible, and minimise inputs (arguments). This makes review and testing easier.

SOLID

Dependency Inversion Principle (DIP)

Depend upon abstractions. Do not depend upon concretions.

Use references to interfaces (preferably) or abstract classes.

SOLID

Liskov Substitution Principle (LSP)

Subclasses should be substitutable for their base classes.

This means adhering to <u>all</u> of your base class's pre- and post-conditions. Therefore, all unit tests written for the superclass must pass when run against the subclass.

…and preferably, you'll have 100% test coverage.

SOLID

Interface Segregation Principle (ISP)

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

Every time an interface is changed, all of its clients need to be recompiled and retested.

Other Principles to Keep in Mind

- Command-Query Separation

  May be thought of as a special case of SRP

- Law of Demeter

  you().might.want.to(rethink).this(); avoid adding unnecessary dependency trees.

- DRY (Don’t Repeat Yourself)

  You’re just making it harder for yourself. Twice.

SRP and DIP

Some of the advantages of applying Single Responsibility and Dependency Inversion:

- Code is easier to review.
- Unit testing is possible. If you need a database, you've written an integration test.
- Unit tests are easier to write. Every time you add an if(…) condition, the set of inputs to test is doubled.
- Conflicting requirements may be satisfied without affecting production code.
- Requirement satisfaction is traceable through all project and design artefacts.

Trade-offs

You will be trading off these principles against each other. In any particular circumstance, the right principle to apply is a matter of judgement, and the main engineering criteria to be considered are:

- Correctness
- Testability
- Maintainability