Challenging Common Software Design Principles: Do they always make for better software?

Christine Miyachi
Systems Engineer and Architect
Xerox Corporation
SDM, 2000
Webinar Outline

• Brief History of some Common Software Design Principles
• Iterative Development
• “Form Follows Function”
• The SOLID principles
  – Single Responsibility
  – Open / Closed
  – Liskov Substitution
  – Interface Segregation
  – Dependency Inversion
• Software Craftsmanship
Some History

- In February 2001, a group of software professionals designed the Agile Manifesto
- **Individuals and interactions** over Processes and tools
- **Working software** over Comprehensive documentation
- **Customer collaboration** over Contract negotiation
- **Responding to change** over Following a plan
Each Iteration is Better

• "A process of repeating a set of operations until a specific result is achieved".
• With software we can typically build something quickly for stakeholders to review.
• From that feedback, we can revise the design.
• Stakeholders often don't know what they want, so iterating will help figure that out.
Why Iterating May Fail

- All stakeholders must have a clear vision of the final product
  - This is rarely the case
- Each cycle should narrow the possibilities of change
  - Sometimes the possibilities of change open up with each iteration
- Convergence may not happen
  - The project runs out of time and what ever is complete is what is delivered
Design Iteration – Denmark Furniture from the 1950s
So What To Do?

• Iteration is still a great way to break down a large project into smaller chunks
• A design that meets all its goals perfectly will not always lead to the simplest design.
  – Sometimes meeting the top few most important goals leads to the simplest design
  – The simplest software is the most beautiful software
• Stop before you made it too perfect
Form Follows Function

- Attributed to a book by Peter Blake called "Form follows Fiasco"
  - Function should be the utmost important in any design
- If function is met in the purest, most simple way, the form follows and it will be beautiful
- What is the success criteria?

From Universal Principles of Design.
"coord": {"lon": 12.4958, "lat": 41.903},
"sys": {"country": "Italy", "sunrise": 1369107818, "sunset": 1369160979},
"weather": [{
  "id": 802, "main": "Clouds", "description": "scattered clouds",
  "icon": "03d"}],
"base": "global stations",
"main": {
  "temp": 290.38,
  "humidity": 68,
  "pressure": 1015,
  "temp_min": 287.04,
  "temp_max": 293.71},
"wind": {
  "speed": 1.75,
  "deg": 290.002},
"clouds": {"all": 32},
"dt": 1369122932,
"id": 3169070,
"name": "Rome",
"cod": 200}
Form Follows Function Example in Software

From http://www.survivingwithandroid.com/2013/05/build-weather-app-json-http-android.html
Show Code Examples
Building 20

MITsdm
Form Varies With Function

- Chair made to be also a Clothes Hanger
Single Responsibility Principle (Robert Martin)

- Every class/module should have a single responsibility over a part of the function.
- The function-part should be encapsulated and the class should do that and only that.
- Example:
  - A class that compiles AND prints a report violates this principle
  - If the content of the report changes, the class must change.
  - If the format of the print must change, then the class changes.
Issues with SRP

• Needs change over time and software is repurposed
  – Examples
Open/Closed Principle

- Classes/modules should be closed for modification but open for extension
Liskov Substitution Principle

- If S is a subtype of T, then objects of type T may be replaced with objects of type S (i.e., objects of type S may substitute objects of type T) without altering any of the desirable properties of that program (correctness, task performed, etc.)
- Also called (strong) behavioral subtyping
LSP Examples

• Classic example of breaking LSP
  – Inheritance: Square is a Rectangle

• In Windows
  – Grid, StackPanels, Canvas
  – Each based on the Panel Class
    • Can contain multiple controls
  – Programmers subclass Canvas instead and create their own custom controls
  – This is a violation and will break when passing around panels.

Example from [http://www.codeproject.com/Articles/597870/Liskov-Substitution](http://www.codeproject.com/Articles/597870/Liskov-Substitution)
Interface Segregation Principle

- No client should be forced to depend on methods it does not use
- Split interfaces which are very large into smaller specific ones
  - clients will only have to know about the methods that are of interest to them
A Job doesn’t violate ISP

• Consider a Print Job, a Scan Job, a Fax Job, a Copy Job
  – They are all Jobs
• There is no Staple Job, just properties on Jobs
• Jobs can be moved through a system without the system knowing what they do
• Based on job properties, things will occur while processing (like stapling)
Dependency Inversion Principle

- 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.
DIP

- Creating dependencies is a risk
  - Handling that risk has some cost.
- What is the life of your software? Is it worth breaking things apart
- We don’t use DIP in all software
  - Think of the String class – we use it directly.
Software Craftsmanship

Not only working software,
    but also well-crafted software
Not only responding to change,
    but also steadily adding value
Not only individuals and interactions,
    but also a community of professionals
Not only customer collaboration,
    but also productive partnerships
The Software

Stage 1: Innocent
Stage 2: Exposed
Stage 3: Apprentice
Stage 4: Practitioner
Stage 5: Journeyman
Stage 6: Master
Stage 7: Researcher
Summary

• Most principles were built upon previous principles and have been around a long time
  – There is some truth to them
• The principles discussed today have some goodness to them
• Depending on the context, principles should be violated
• Adhering to principles is not free and should be used wisely
• Most teams have people that do not have the skills to follow all the principles