# SOLID is a guideline not the goal

Or

# Use your brain and ignore principles

## Thoughts

* Show the benefit of SOLID
* Explain where SOLID came from
* Show how uncle bob makes a lot of money from SOLID
* Show where each principle came from and the goal it intended to solve
* Use that guy who used my tweet to write a blog post about “there are no best practices”
* Mention that this is all opinions and subject – what I’m saying and Software development
* People think they are doing their best by being “SOLID”, and we shouldn’t criticise them for that
* Context matters
* Other useful metrics
  + How long will it take
  + How deep is the object graph – how many classes and methods will I have to step through just to work out the bigger picture?
* Start the session by talking in terms of what the attendee is going to gain
  + Write “better” code
  + Create simpler code bases
  + Be more alert to marketing and hype in software development
  + Teach them analytical skills
* Useful techniques
  + Allow code to accumulate and grow a bit larger than Uncle Bob would allow
  + Wait until you can see a good abstraction that fits
    - Pretty sure the GOOS book uses this approach
* Don’t use a SOLID principle as an excuse for the way you are writing a piece of code unless you can explain the business value of it
* If you can’t explain why you are using a principle you are losing sight of the bigger picture
* Simple solutions probably just need simple code
* “There are no best practices” only what people have found works for them in certain situations

## Presentation Plan

1. show intro slides and discuss them
   1. Get people to raise hands and agree that most often they are here to deliver business value
2. World before SRP
   1. Show an example of a massive class
3. Please don’t do this
   1. Discuss this blog post <http://blog.gauffin.org/2011/07/single-responsibility-prinicple/>
   2. Refer to my blog post for the reasons
   3. In some cases, he might be right to separate that logic – but his example, there is no contextual need to. His architecture, patterns, and framework don’t guide him down that path
   4. Also look at this… <http://lostechies.com/gabrielschenker/2009/01/21/real-swiss-don-t-need-srp-do-they/>
      1. With all those classes and interfaces anyone would think those 20 lines of code did something important ☺
   5. Show my other blog post: <http://www.ntcoding.blogspot.co.uk/2012/05/contentious-controller.html>
   6. Walk through the example codes and ask which one is easiest to understand
4. OCP saves the day
   1. Explain a bit of Bob’s paper https://docs.google.com/a/7digital.com/file/d/0BwhCYaYDn8EgOTViYjJhYzMtMzYxMC00MzFjLWJjMzYtOGJiMDc5N2JkYmJi/edit?hl=en
5. OCP is as old as the dinosaurs
   1. Show both V1 baskets – highlight the increased complexity to be fully OCP

## Slides Plan

1. What you will get out of this session
   1. Why do you care about the quality of the code you create?
   2. Why do you follow principles and practices like SOLID?
      1. You are motivated to do the best job possible
      2. You are told that is how to create good software
2. What is the goal of creating software
   1. Usually to create business value
   2. Not to be SOLID
   3. SOLID can bring business value though, but makes no guarantee
   4. Maybe define some business value considerations
   5. Get everyone to raise their hands and agree business value is the most important thing
3. This session will use SOLID as any example, but applies to any “best practice”
   1. You should always be asking yourself “does this make sense”.. “does it contribute to my ultimate goal”
   2. E.g. TDD, more tests, more layers of abstraction, IoC
4. Uncle Bob and others have made a lot of money from the marketing aspects of “SOLID” and “agile”
   1. Show some examples
5. Developers have the best intentions when they are applying SOLID – because they are led to believe it is the holy grail
   1. We shouldn’t criticise devs
6. Context is king
   1. You need to look at the situation you find yourself in
      1. What are the business or ultimate goals?
      2. What is the cost of being SOLID against not being solid
7. SRP – use my blog post example, or maybe take it further and show proof of the dangers
   1. A world before SRP…..
   2. Reason for SRP (context is was made in)
   3. SRP taken out of context – mapping is not a responsibility
   4. SRP taken too far
   5. Find some real-world code examples?
   6. Could mention the coupling issues that were present when I started the media-delivery team
   7. Considerations
      1. Make db access more explicit at the expense of solid sometimes?
         1. NHibernate example
      2. Make network calls more explicit?
      3. Show the “latency every programmer should know”
8. Open Closed
   1. Reason for OCP (context is was made in) 1988!
   2. Want s you to use inheritance !! http://en.wikipedia.org/wiki/Open/closed\_principle
   3. Take Uncle Bob’s quote about too much Open-Closed takes too long and is too hard to manage
   4. Show some examples?
9. Liskov
   1. Rarely do I disagree with Liskov substitution
   2. Doing something completely un-expected in a subclass is usually not a smart thing to do
   3. But if you’re not using your classes polymorphically, then there is no danger at all
   4. The context that needed Liskov <http://en.wikipedia.org/wiki/Liskov_substitution_principle>
   5. Generally a pretty good idea not to violate Liskov, because it is really dangerous
   6. But it relies on polymorphism
   7. ?? Not sure you should abuse it
   8. Jon Skeet disregared it
   9. State machine can cause Liskov substitutions
   10. Even Barbera Liskov saying it does not apply all the time <http://c2.com/cgi/wiki?LiskovSubstitutionPrinciple> – Alistair Cockburn and others disagree with Uncle Bob
10. Interface Segregation Principle
    1. Where ISP comes from
    2. The context it was created
    3. Big massive interfaces are a bad thing (usually) – show example
    4. But taken too far (and arguably out of context) you are going to end up with LOTS of interfaces
       1. Less cohesive system
       2. More concepts to understand
       3. Thus harder to understand
       4. Get you into the habbit of creating an interface for everything
    5. Confusion from object mentor about what ISP is though <http://ayende.com/blog/153985/limiting-your-abstractions-reflections-on-the-interface-segregation-principle>
    6. Just use your brain – what makes sense in your context
11. Dependency inversion principle
    1. Where it came from
    2. Context it was made in
    3. What it saves you from – tight coupling
       1. So now you can’t test things
    4. On a \*few\* occasions that coupling has no real cost, thus the principle is not needed
       1. Use of the logger in indexers
       2. Can save you passing objects all the way down the object graph, or mocking them
12. If not SOLID then what?
    1. Other useful metrics
    2. Ways of thinking about problems
       1. If I violate SOLID, how can it hurt me?