

The background features dynamic, flowing waves in shades of red, orange, and yellow, creating a sense of movement and energy. The waves are layered, with some appearing more prominent than others, and they curve across the frame.

# ABSTRACT ART

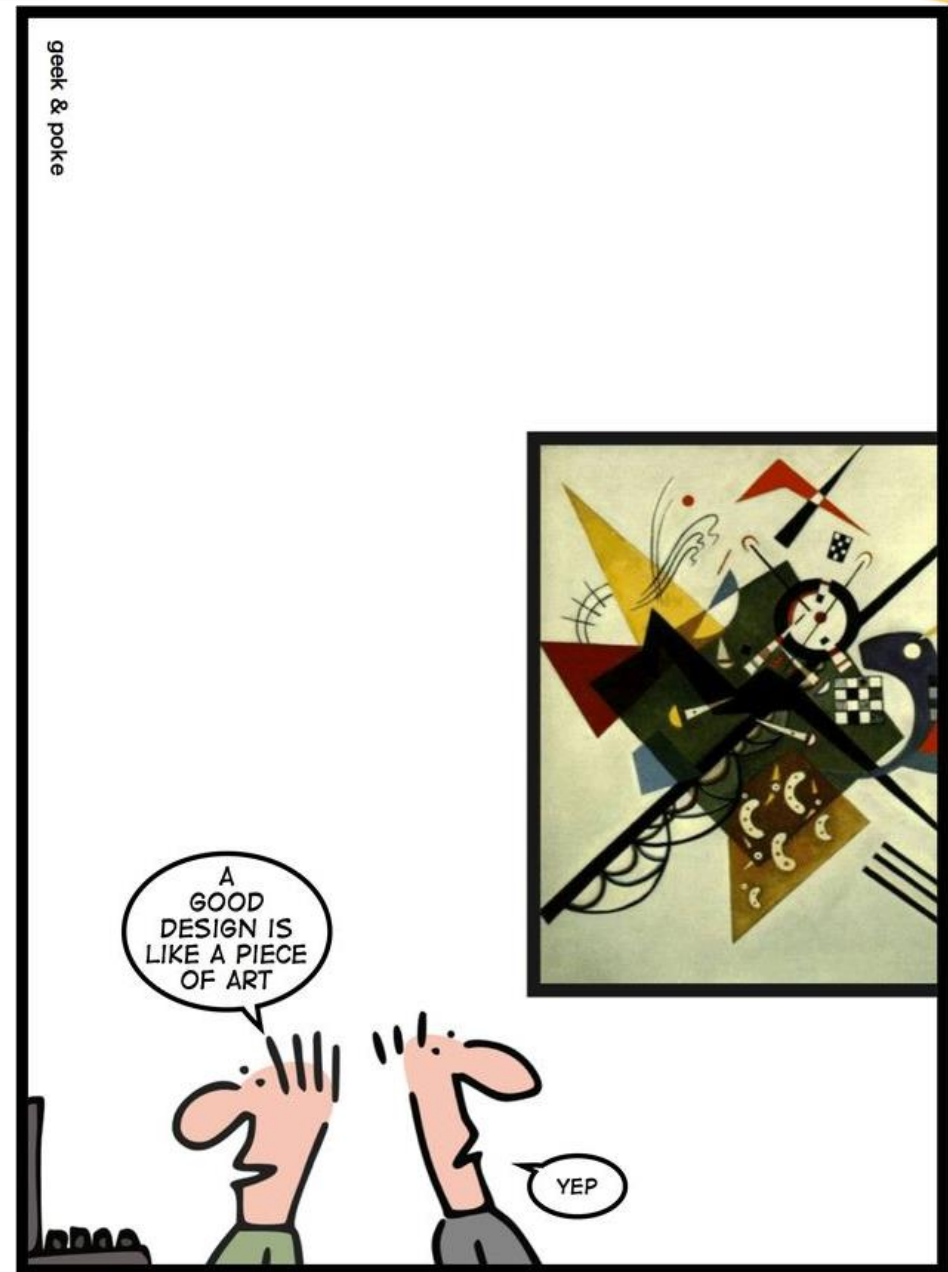
GETTING ABSTRACTION “JUST RIGHT”

Presented by Jeremy Clark  
[www.jeremybytes.com](http://www.jeremybytes.com)

# A Good Design is like A Piece of Art

Geek & Poke – <http://goo.gl/ifd53l>

@jeremybytes



ABSTRACTION

# ABSTRACTION IS AWESOME!



**Maintain**



**Extend**



**Test**

# ABSTRACTION IS AWFUL!



Complexity



Confusion



Debugging  
Difficulty

**Frustration**

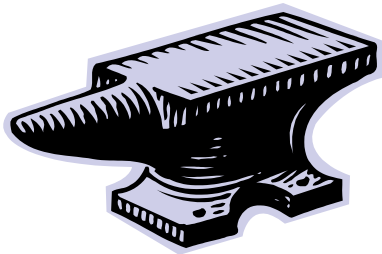




[https://archive.org/details/goldilocks\\_and\\_the\\_three\\_bears](https://archive.org/details/goldilocks_and_the_three_bears)

# GOLDBLOCKS THE DEVELOPER

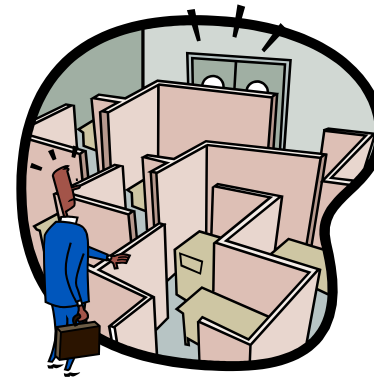
**Too Little**  
Abstraction



**Just Right**



**Too Much**  
Abstraction



# TWO TYPES OF DEVELOPERS

Over-Abstractor

The diagram consists of two horizontal rows. The top row features a teal rounded rectangle with the text 'Over-Abstractor' in white. To its left and right are thin teal lines that extend to the edges of a larger, empty teal rectangular frame. The bottom row features a blue rounded rectangle with the text 'Under-Abstractor' in white. To its left and right are thin blue lines that extend to the edges of a larger, empty blue rectangular frame.

Under-Abstractor



## Over-Abstractor

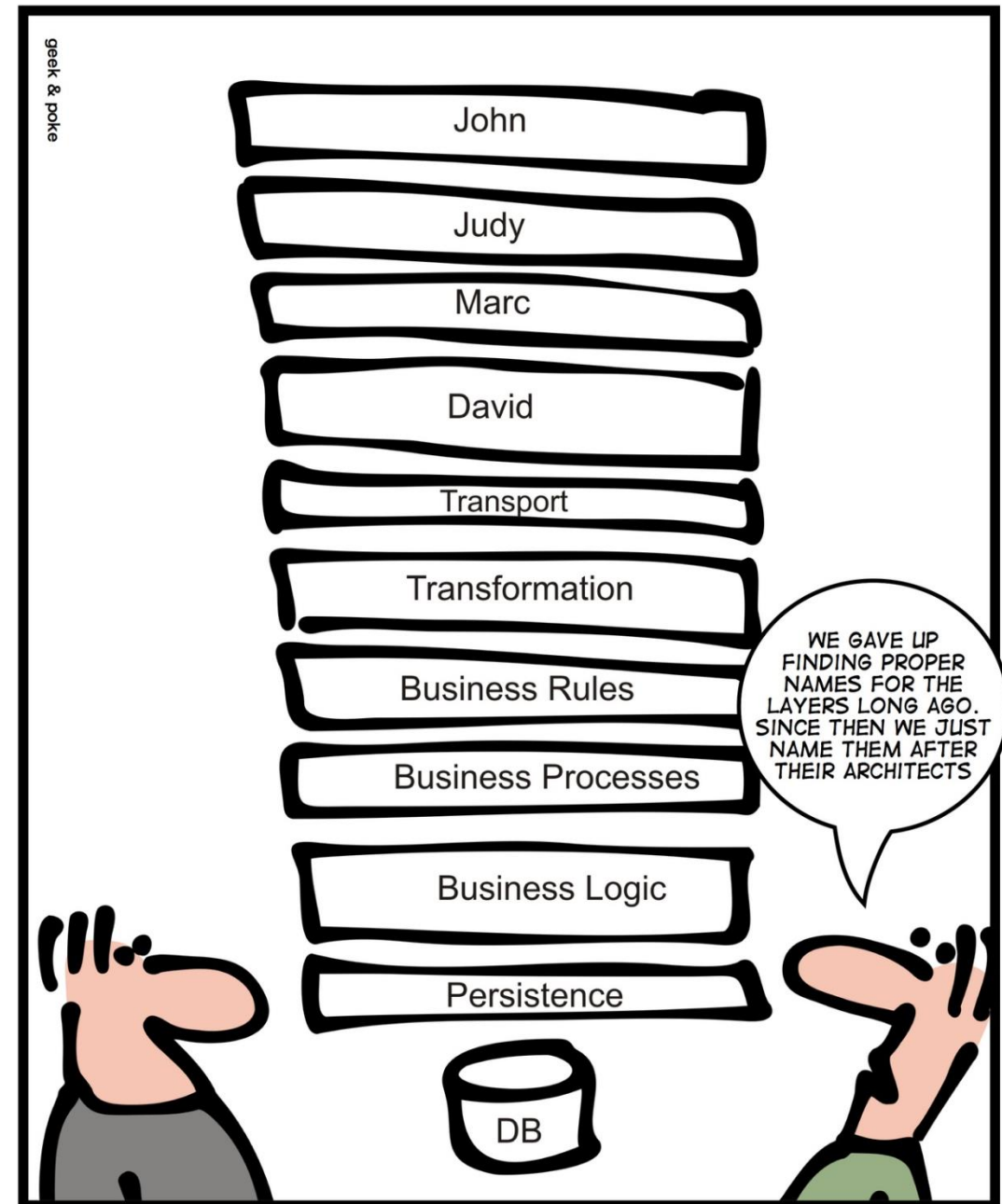
- “We’ll have a good use for this in the future.”
- Overly Complex
- Difficult to Maintain



# A Good Architect Leaves A Footprint

Geek & Poke: <http://goo.gl/B4uXa3>

@jeremybytes



A GOOD ARCHITECT LEAVES A FOOTPRINT



## Under-Abstractor

- “Let’s keep things simple.”
- Rigid
- Difficult to Maintain

# COMMON PROBLEM

## Over-Abstractor

- “We’ll have a good use for this in the future.”
- Overly Complex
- Difficult to Maintain

## Under-Abstractor

- “Let’s keep things simple.”
- Rigid
- Difficult to Maintain



# **The Default State Quiz**

## Who Are You?

Let's build a plug-in architecture...

Awesome!  
Let's do it.

Maybe we  
should look at  
compile-time  
options.



We need to share a value between modules...

I'll create an object state manager.

Let's use a global variable.

# How should we do the UI?

Here's a new  
JavaScript  
framework.

Let's use the  
same framework  
we did last time.

# Pull data from a database...

ORMs are  
awesome!

```
SELECT *  
FROM Customers  
WHERE ID = [@id]
```

We need an object instance...

```
var logger =  
    DIContainer  
    .Resolve<ILogger>()
```

```
var logger =  
    new FileLogger()
```



“  
Neither answer is right or wrong. The  
correct response is “It depends.”  
”

—Jeremy's Standard Response



Let's build a plug-in architecture...

Awesome!  
Let's do it.

Maybe we  
should look at  
compile-time  
options.

We need to share a value between modules...

I'll create an object state manager.

Let's use a global variable.

# How should we do the UI?

Here's a new  
JavaScript  
framework.

Let's use the  
same framework  
we did last time.

# Pull data from a database...

ORMs are  
awesome!

```
SELECT *  
FROM Customers  
WHERE ID = [@id]
```

We need an object instance...

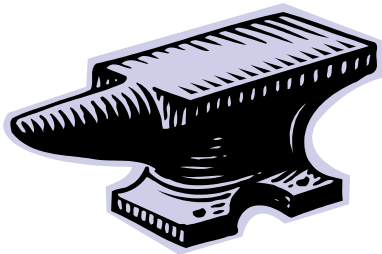
```
var logger =  
    DIContainer  
    .Resolve<ILogger>()
```

```
var logger =  
    new FileLogger()
```



# BE HONEST WITH YOURSELF

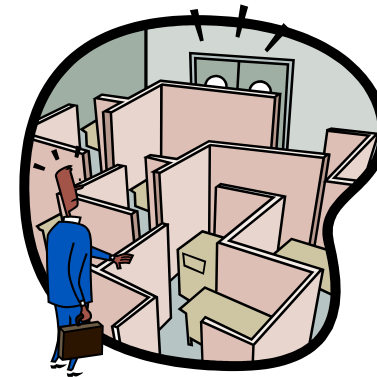
**Too Little  
Abstraction**



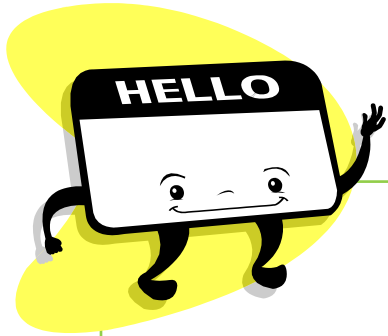
**Just Right**



**Too Much  
Abstraction**



# WHO AM I?



## Under-Abstractor

- Hello. My name is Jeremy,  
and I'm an Under-Abstractor.

“Keep Things Obvious”

“Don't Be Tricky”

# REPORTING APPLICATION

The image displays a reporting application interface with three overlapping windows. The background window is titled "Report Viewer" and features a sidebar with three buttons: "Report List", "Add Report", and "Update Report", each with a right-pointing arrow icon. The main content area of this window shows a table with two columns: "Report Title" and "Category". The visible rows are:

Report Title	Category
Quarterly Financials	
Weekly Sales	
Inventory	
Pending Shipments	
Incoming Shipments	
Advertising Sales	

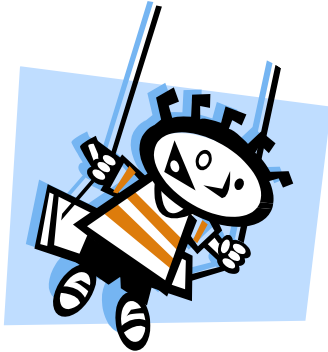
The middle window, also titled "Report Viewer", is titled "Pending Shipments". It contains fields for "Start Date" (8/24/2014), "End Date" (8/24/2014), and "Additional Info". To the right of these fields is a calendar for August 2014.

August 2014						
Su	Mo	Tu	We	Th	Fr	Sa
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16

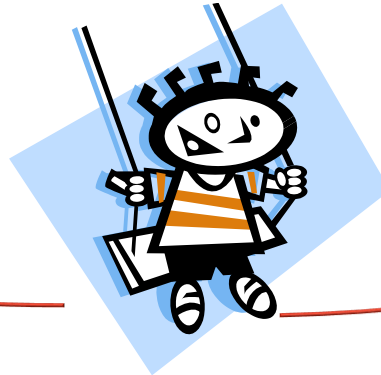
The foreground window is the main application interface. It has a top navigation bar with three tabs: "New Report File", "Edit Report", and "Edit Parameters". Below this is a secondary bar with four tabs: "Splash", "Report List", "Parameter Values", and "Report Viewer". The "Report Viewer" tab is currently active, showing a form with two input fields: "Report Title" and "Category". A "Run" button is located at the bottom right of this window.

# THE PENDULUM EFFECT

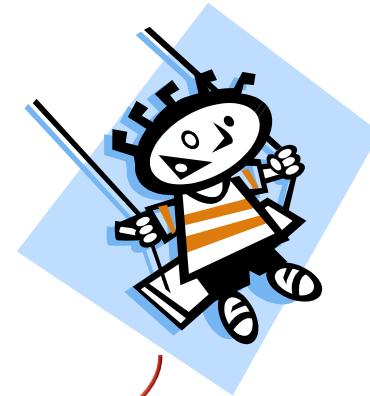
**Under-  
Abstraction**



**Just Right**



**Over-  
Abstraction**



# THOSE AROUND YOU

## Over-Abstractor

- Jeff loved to build components.
- He liked to create code for re-use.
- He thought of all possible scenarios.





## A SYMBIOTIC RELATIONSHIP

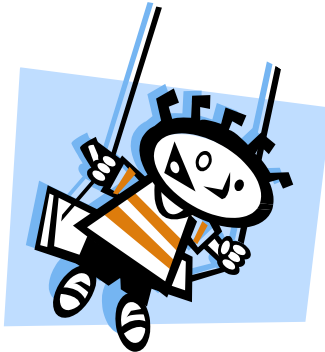
The  
Over-Abstractor  
helps the  
Under-Abstractor  
get things  
Just Right

The  
Under-Abstractor  
helps the  
Over-Abstractor  
get things  
Just Right

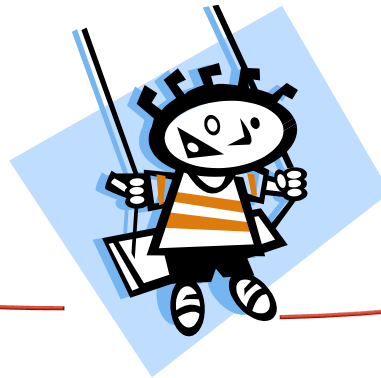


# THE PENDULUM EFFECT

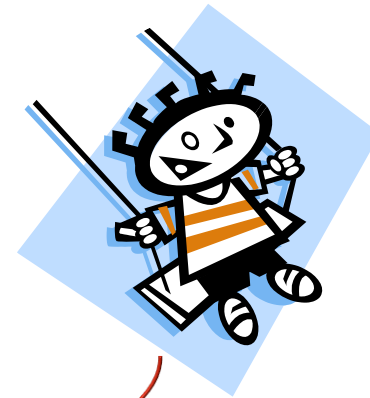
**Under-  
Abstraction**



**Just Right**



**Over-  
Abstraction**



# VARIOUS DATA SOURCES

Microsoft SQL Server

MongoDB

CSV

WebAPI

Oracle

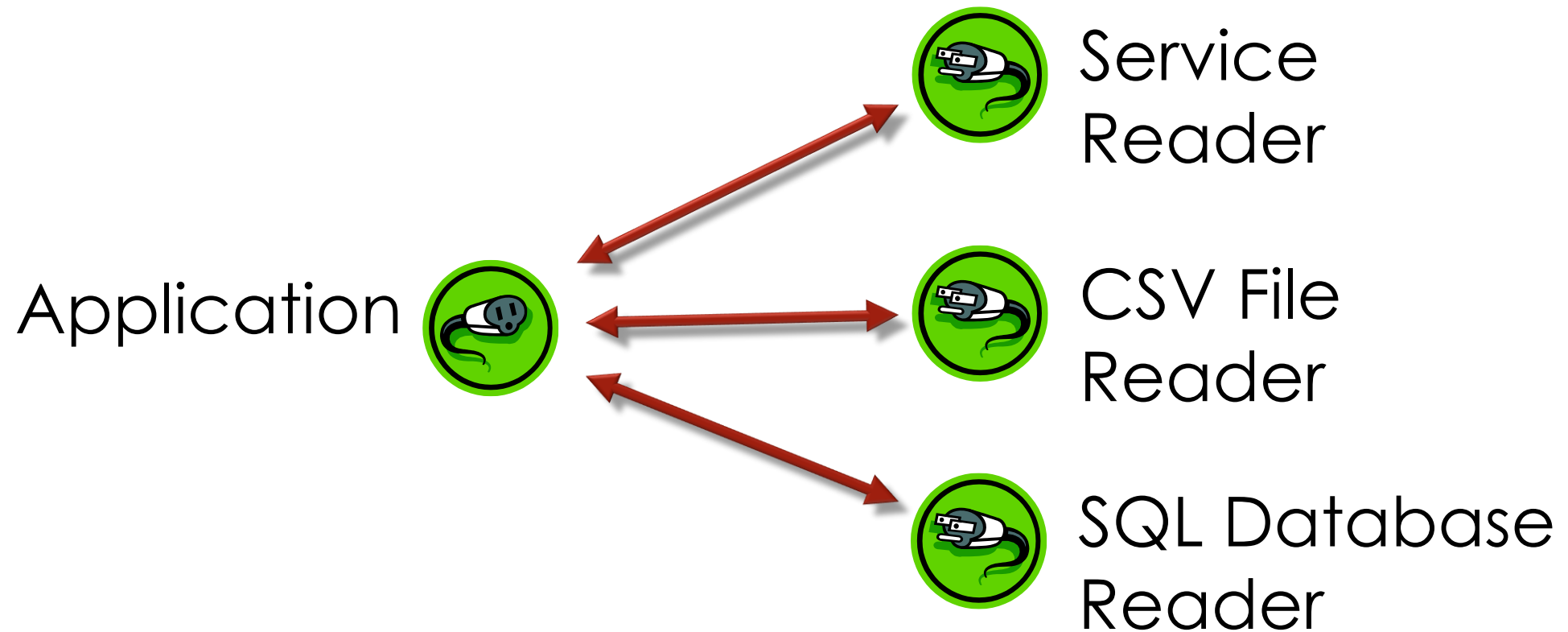
Amazon RDS

JSON

Azure Cosmos DB

Hadoop

# PLUGGABLE DATA READERS







# DRY

- Don't Repeat Yourself

Under-Abstractor



# DON'T REPEAT YOURSELF

Consolidate  
Similar  
Code

Avoid  
Copy/Paste

Copy/Pasta

Spaghetti  
Code

# SoC

- Separation of Concerns

Under-Abstractor

# SINGLE RESPONSIBILITY PRINCIPLE

Complements  
Separation of  
Concerns

The “S” in  
S.O.L.I.D.

A class should  
have only one  
reason to  
change

A class should  
do one thing  
(and do it well)



# YAGNI

- You Ain't Gonna Need It
- (You Aren't Going to Need It)

Over-Abstractor



# MORAL OF YAGNI

- Code for the features you have now
- Add abstraction as you need it
- Don't add abstraction based on speculation

We still think about the future,  
but we don't implement it yet.

# KISS

- Keep It Simple, Stupid
- (Keep It Short & Simple)
- (Keep It Simple & Straightfoward)

Over-Abstractor

# DDIY

- Don't Do It Yourself

Over-Abtractor

Under-Abtractor



## Over-Abstractor

- Over-Abstractors like to build things to solve specific problems

## Under-Abstractor

- Under-Abstractors shy away from external frameworks and libraries

# EXAMPLES

## Dependency Injection

- Ninject, Autofac, Spring.NET, Microsoft.Extensions.DependencyInjection

## Unit Testing Framework

- NUnit, xUnit.net, MSTest, Approval Tests

## Mocking

- Moq, NSubstitute, FakeItEasy, JustMock

## UI Framework

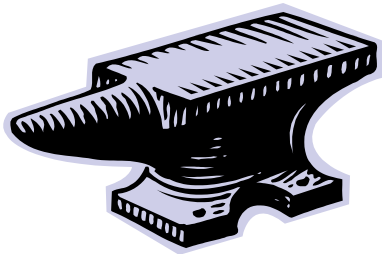
- Angular, React, Vue, Prism

# ABSTRACTION IS AWESOME & AWFUL



# THE GOLDBLOCKS PRINCIPLE

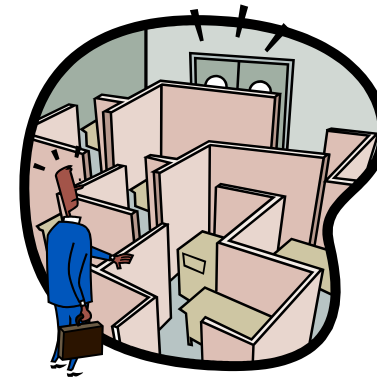
**Too Little**  
Abstraction



**Just Right**



**Too Much**  
Abstraction



# GETTING THINGS RIGHT

## DRY

- Don't Repeat Yourself

## SoC

- Separation of Concerns

## YAGNI

- You Ain't Gonna Need It

## KISS

- Keep It Short & Simple

## DDIY

- Don't Do It Yourself



THANK YOU!

Jeremy Clark

- <http://www.jeremybytes.com>
- [jeremy@jeremybytes.com](mailto:jeremy@jeremybytes.com)
- [@jeremybytes](#)