The **ROI** of **Refactoring**

vs

By Neil Green

What is this talk about?

Convincing your **Business Stakeholders** that they should give you the time to refactor your code.

We'll do this by putting things in terms they understand: **ROI (Return on Investment).**

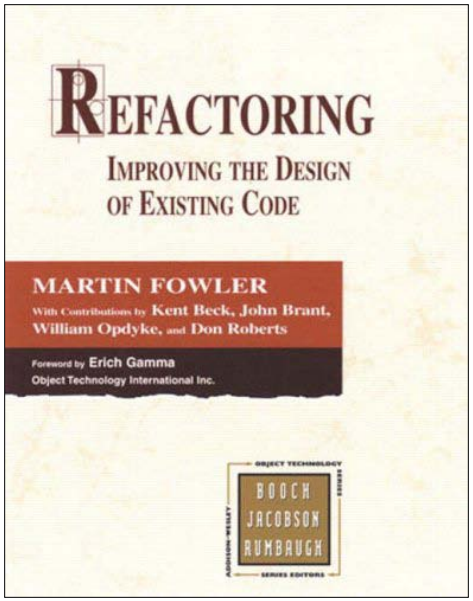
We'll ice it down using an analogy everyone can understand: **Lego and Playdoh.**

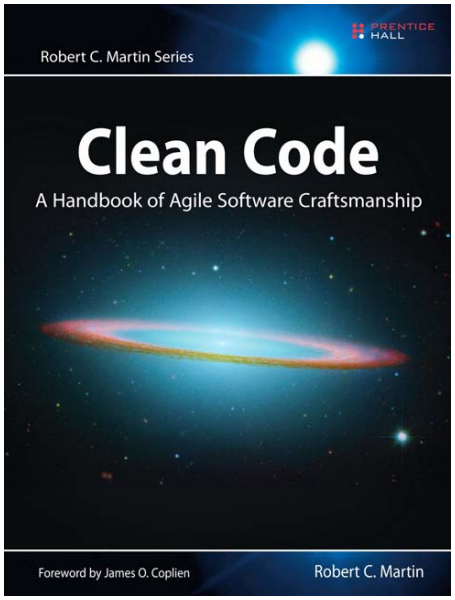
What is this talk **NOT** about?

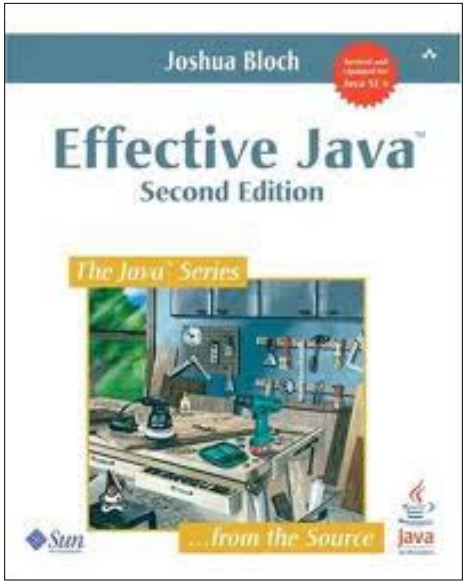
HOW to Refactor!

(This is about how to **SELL** Refactoring!)

How to Learn How to Refactor







How Engineers Define Refactoring

Improving the design of existing code without changing the functionality

How Business Defines Refactoring

A fancy way of engineers saying,“We screwed up so now can you give us time and money so we can go back and fix the code so that our work will be easier?”

Which brings us to our analogy…

VS

- Reusable Components
- Easy to Maintain
- Able to Be Built Upon
- Linear Complexity
- A Pleasure to Work In

- A Big Crappy Mess
- Brittle & Un-maintainable
- Cannot Be Built Upon
- Exponential Complexity
- A Pain in the Butt

Disclaimer

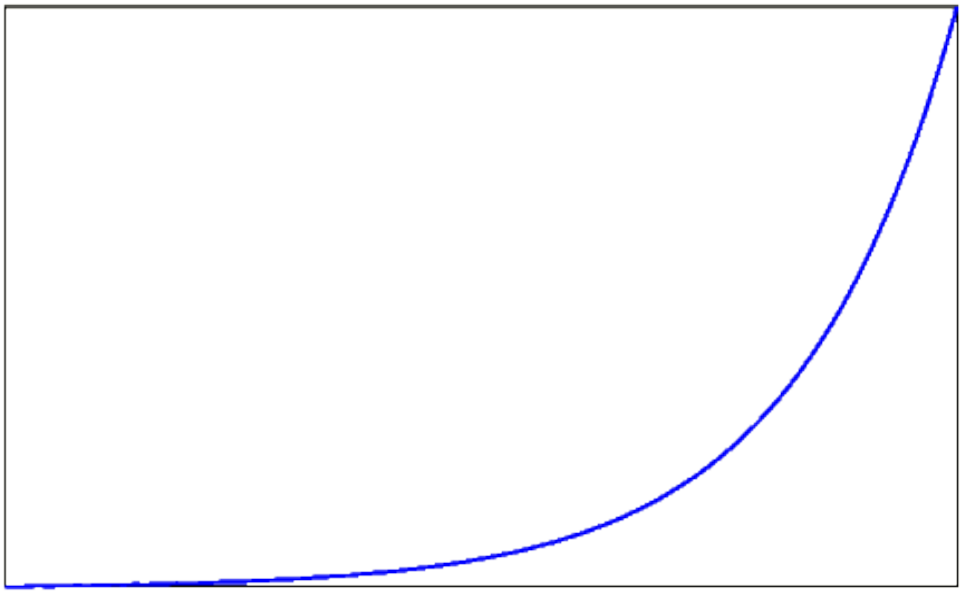
!=

- Classes
- Objects
- Types
- Interfaces
- Inheritance
- Encapsulation
- Design Patterns
- Frameworks
- Language Features
- etc…

Why should we invest time and money in improving the code?

Because of the Hockey Stick!

Refactoring ROI



Product Life Span

(This is completely useless. Don’t try this or you’ll be laughed out of the room)

Benefits of Refactored Code

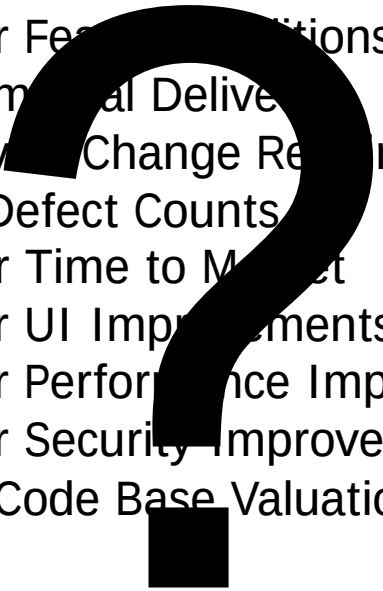
Benefits for Engineers

- Easily Maintained Code Base
- High Code Reusability
- Consistent Code Standards
- No Need for Overtime
- Working Efficiently Together
- Ease of Automated Testing
- Highly Capable Frameworks
- High Encapsulation, Low Coupling
- Less Fragile, More Robust Code

Business Doesn't Care

Benefits for Business

- Easier Feature Additions
- Incremental Delivery
- Ability to Change Requirements
- Low Defect Counts
- Faster Time to Market
- Easier UI Improvements
- Easier Performance Improvement
- Easier Security Improvements
- High Code Base Valuation



Business Doesn't Know

Faster Time to Market

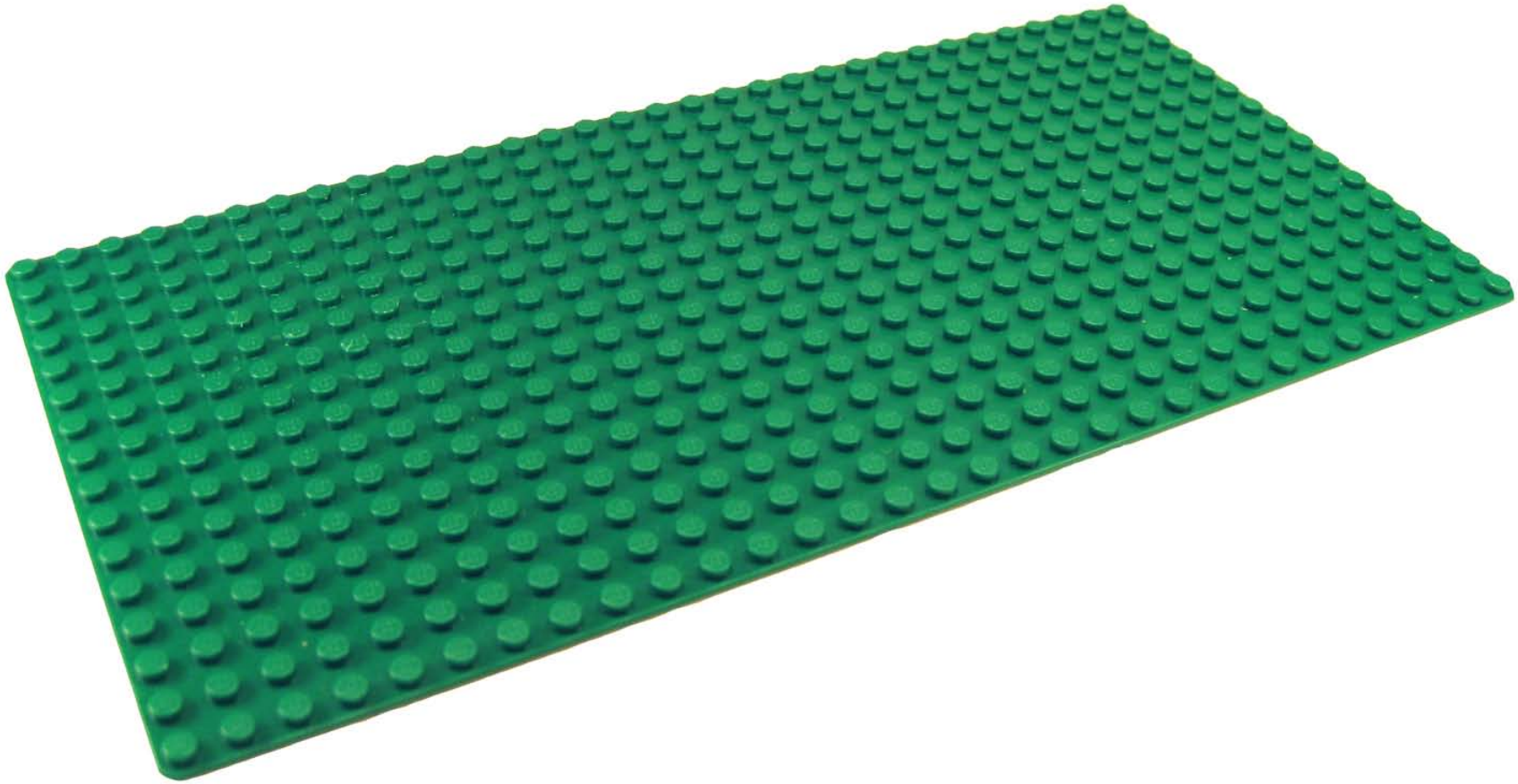
Building upon pre-existing functionality is highly efficient.

Building everything from scratch is time consuming.

Incremental Delivery

Features can be delivered one at a time.

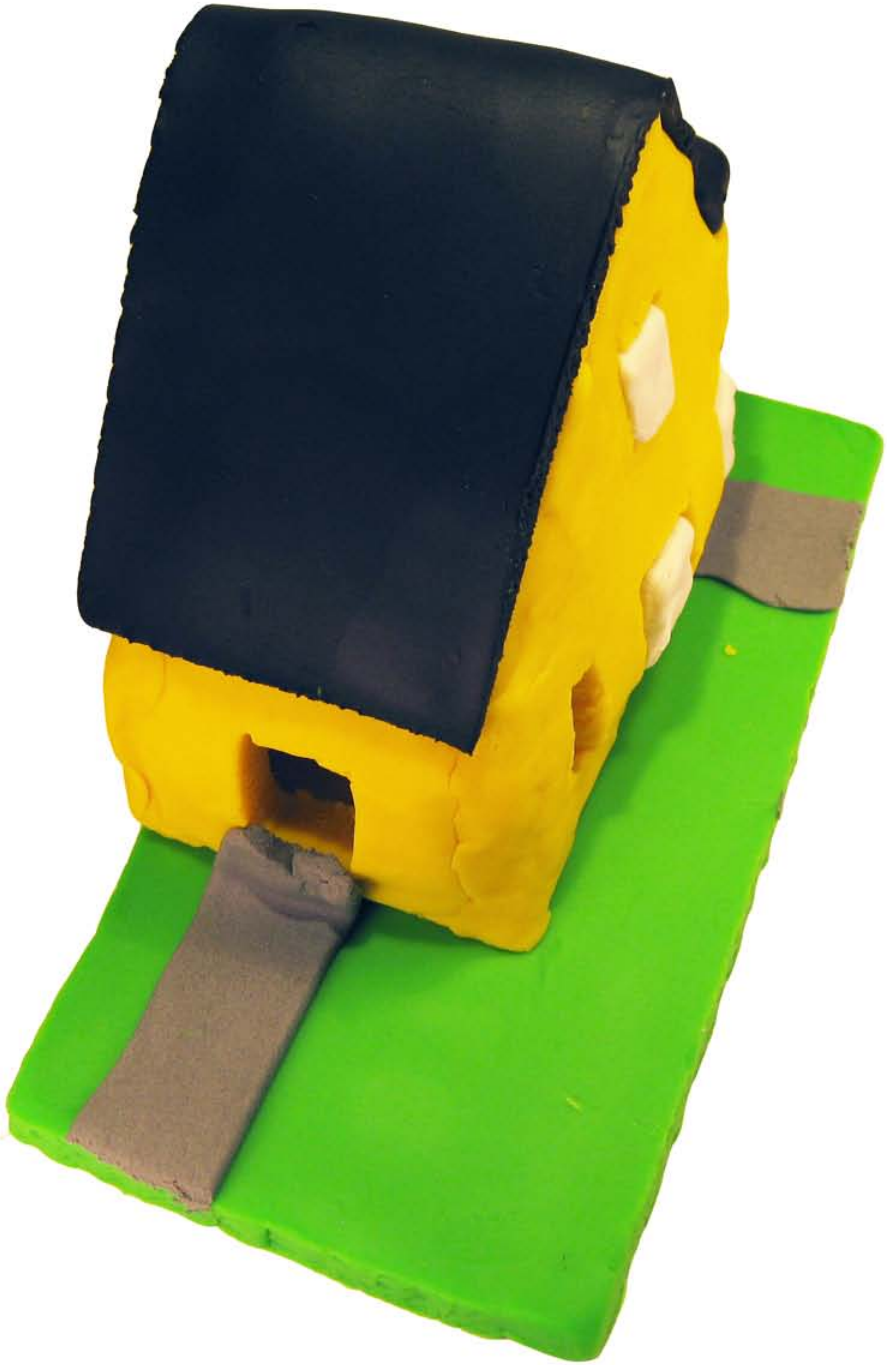
No way to break up the work, so product must be delivered in full.

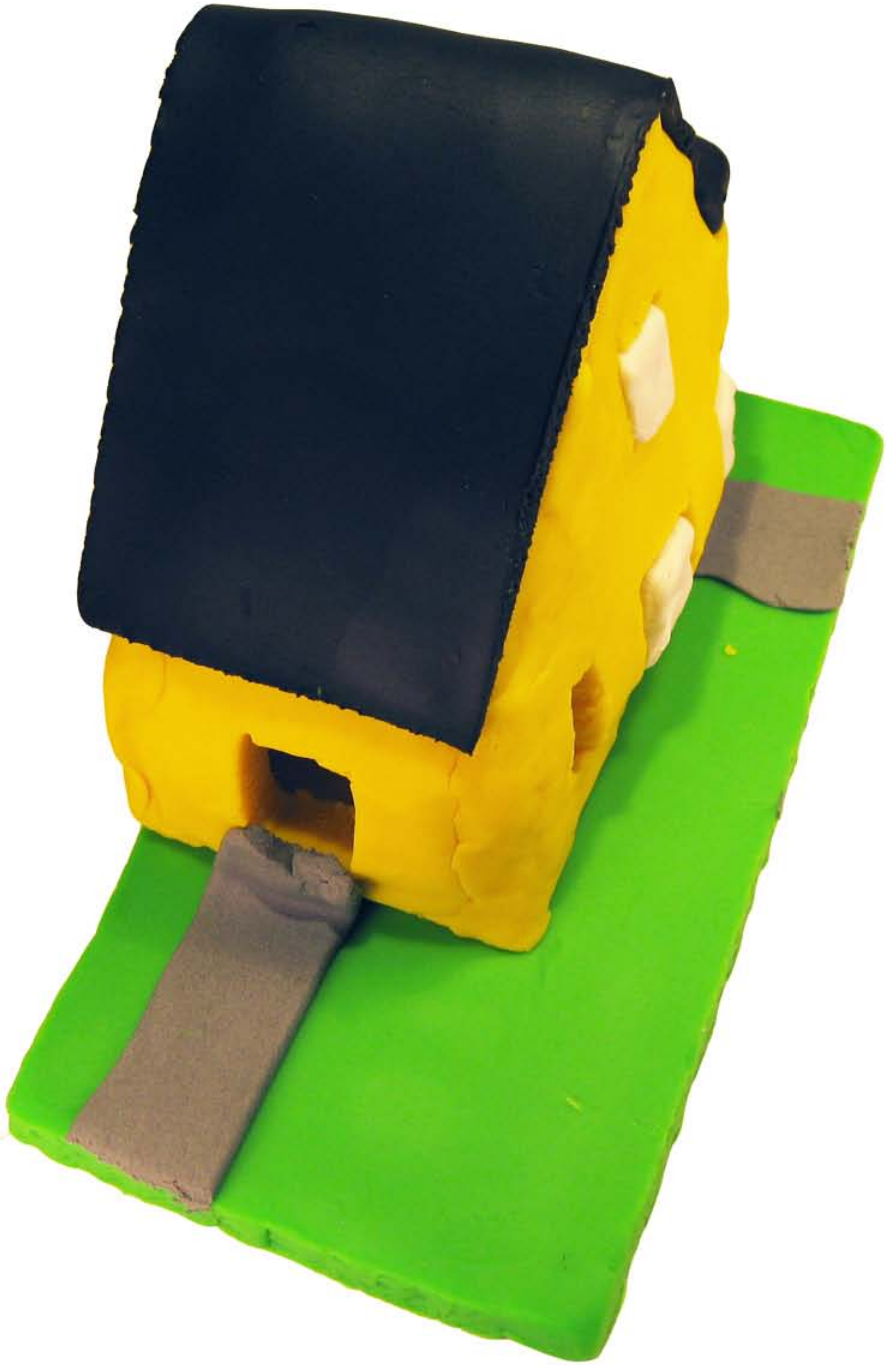


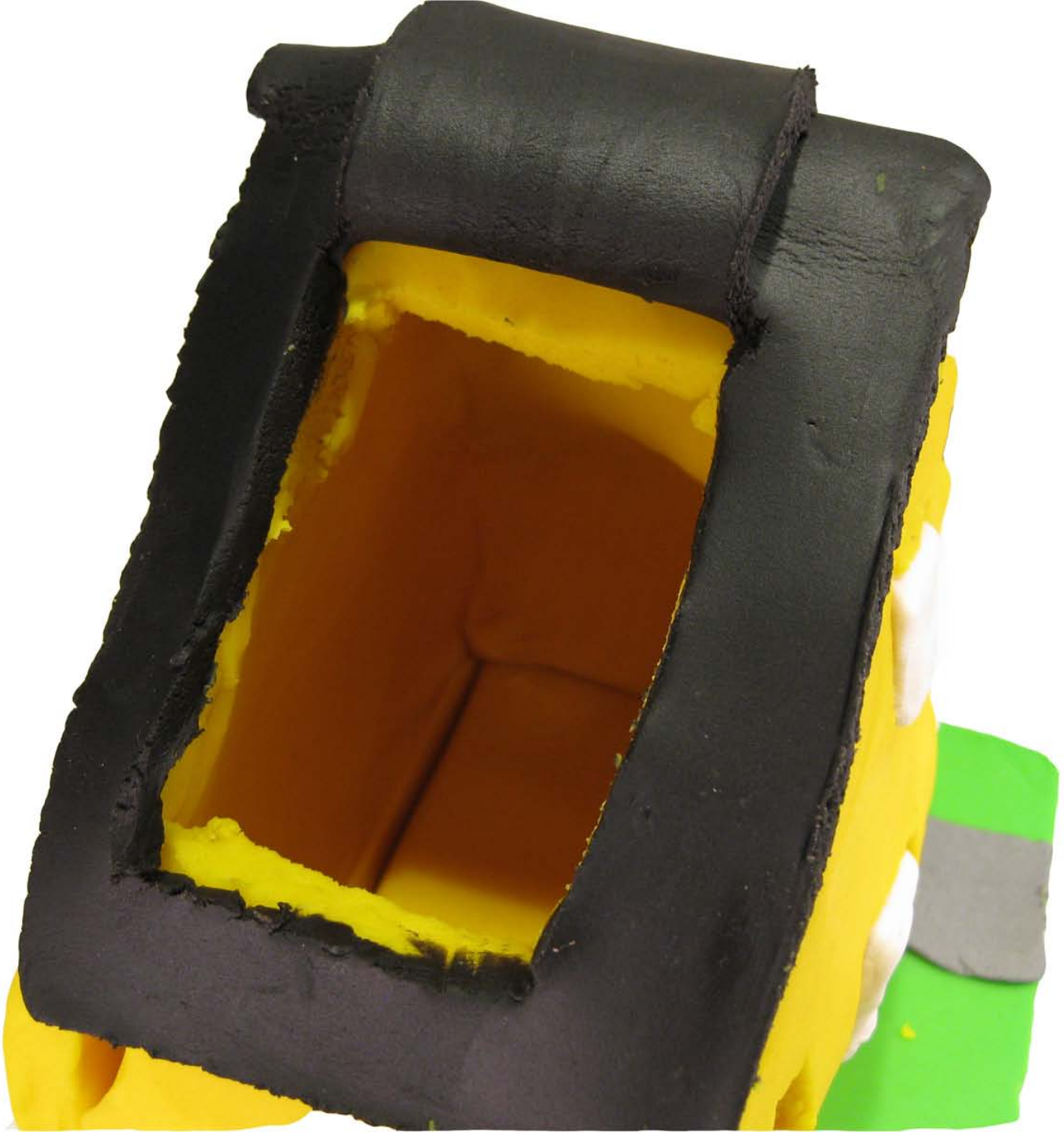
Easier Feature Addition

Can build new components on top of existing components.

With nothing to plug into, things need to be shoehorned in



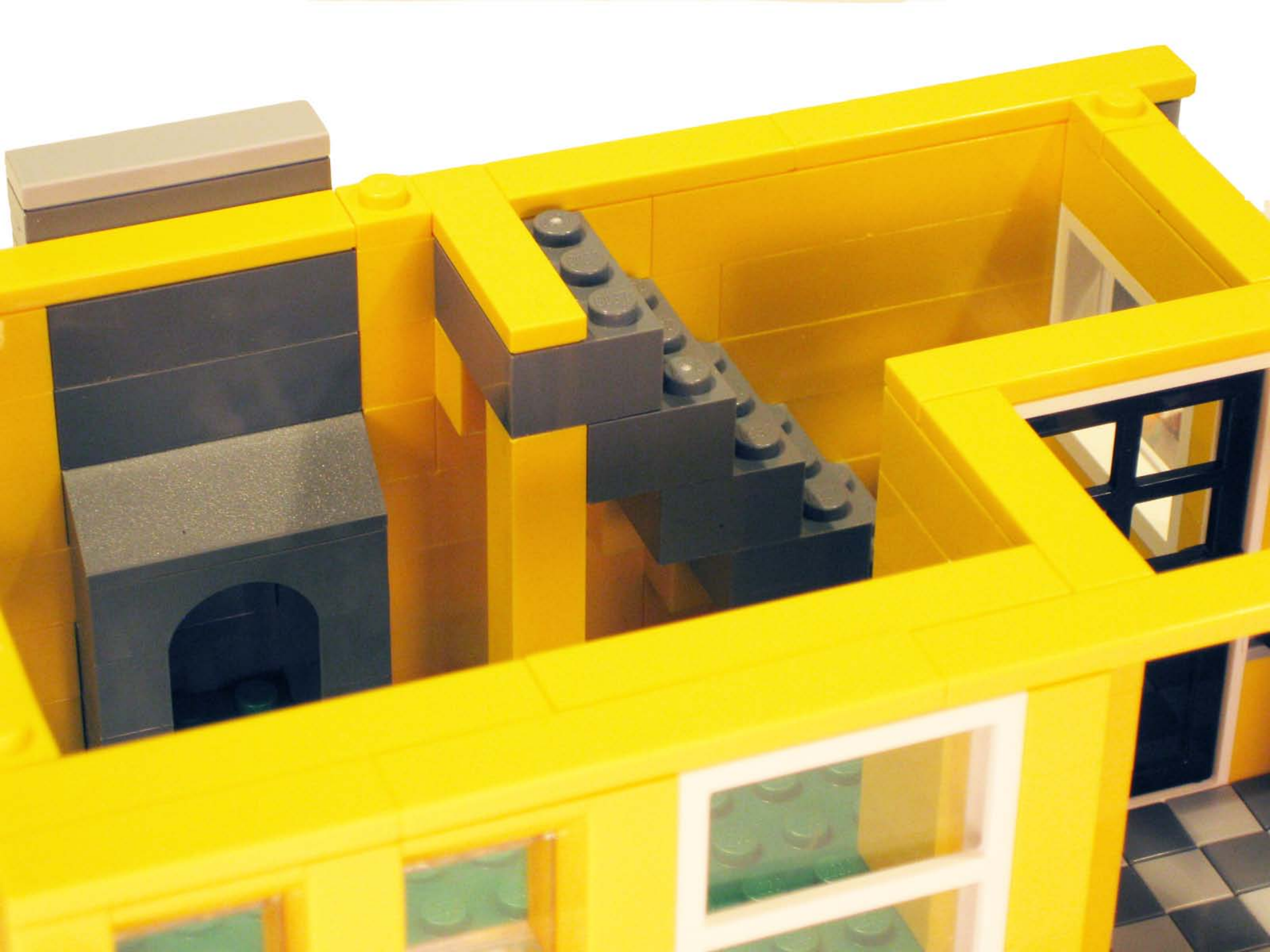




Ability to Change Requirements

Requirements are well encapsulated, so changes can be made discretely

The code was not designed to be changed, so a changed requirement is painful.



Easier Performance Improvements

Easier to identify and fix bottlenecks.

Identifying bottlenecks is hard, fixing them is harder.

Easier Security Improvements

Vulnerabilities are isolated and easier to fix.

Vulnerabilities are scattered and difficult to fix completely and reliably.

Easier UI Improvement

UI components can be improved independently from the rest of the system.

UI is deeply integrated into all parts of the system making improvements difficult.

Low Defect Counts

Defects are isolated and easy to find and resolve.

Difficult to find what's causing the defect, and nearly impossible to know if by fixing it other things are not broken.

High Code Base Valuation

Acquiring companies see more value in a well factored code base.

Code that has to be thrown away has no value.

Importance of Refactored Code

Benefit	Start Up	Enterprise	Pre-Acquisition
Easier Feature Additions	+++++	+++++	++
Incremental Delivery	++++	++	+
Ability to Change Requirements	+++++	++	+
Low Defect Counts	+	+++++	+++++
Faster Time to Market	+++++	++	+
Easier UI Improvements	+	+++	+
Easier Performance Improvement	+	++	+++
Easier Security Improvements	+	++	+++++
High Code Base Valuation	+++	+	++++++++++

Worksheet for Calculating ROI

Benefit	$ Worth	$ Invest	$ ROI
Easier Feature Additions			
Incremental Delivery			
Ability to Change Requirements			
Low Defect Counts			
Faster Time to Market			
Easier UI Improvements			
Easier Performance Improvement			
Easier Security Improvements			
High Code Base Valuation			

* Work together with Business to complete this worksheet.

Questions You May Still Be Asked

Question:

“Why wasn’t the code factored right the first time?”

Answer:

“When building complex systems, you often have to solve new problems that you didn’t know you were going to encounter.”

Questions You May Still Be Asked

Question:

“How long will It take to refactor the code?”

Answer:

“It depends on the efficiency and productivity of your team. I recommend leaving them alone to do their jobs.”

Questions You May Still Be Asked

Question:

"How do I know the code has been refactored properly?"

Answer:

"You have hired a competent engineering team. Trust them. If all else fails, you can get third party validation."

Questions You May Still Be Asked

Question:

“Will I find myself in the this place again?”

Answer:

“Not if you allow your team to constantly refactor.”

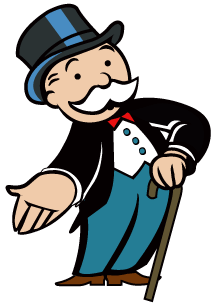
In Conclusion

Refactored Code Gives You...

- ***Fast** Time to Market*
- ***Quick** Proof of Concepts*
- ***Low Cost** Maintenance*
- ***Low Cost** Upgrades*

Win Deals and Keep Customers!!!

$$$$$



Thanks! Questions?

http://www.linkedin.com/in/neilgreen

http://www.facebook.com/robertneilgreen

http://www.twitter.com/feyn