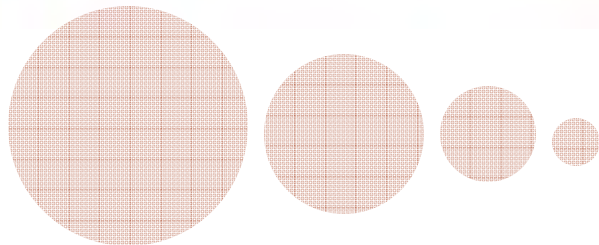


# Refactoring to Patterns

A practical look into the Agile approach on Evolutionary Design



IndicThreads.com Conference On Java Technology  
26th & 27 Oct. 2007 Pune, India



Paulo Caroli  
ThoughtWorks

# About Paulo Caroli

The ThoughtWorks logo is displayed in white text on a dark purple rectangular background with a subtle, abstract pattern.

- A technologist at ThoughtWorks US
- Master's Degree in Software Engineering
- Sun Certified J2EE Architect
- More than 13 years in SW Development.
- Agile Techniques, such as XP, TDD, Continuous Integration
- Object Oriented development practices
- More at [www.caroli.org](http://www.caroli.org)



# Agenda

- Refactoring and Agile Techniques
- Refactoring and Functional Requirements
- Refactoring and TDD
- Simple Design and Code Smells
- Refactoring to Patterns
- Case Study
- Conclusion
- Q & A



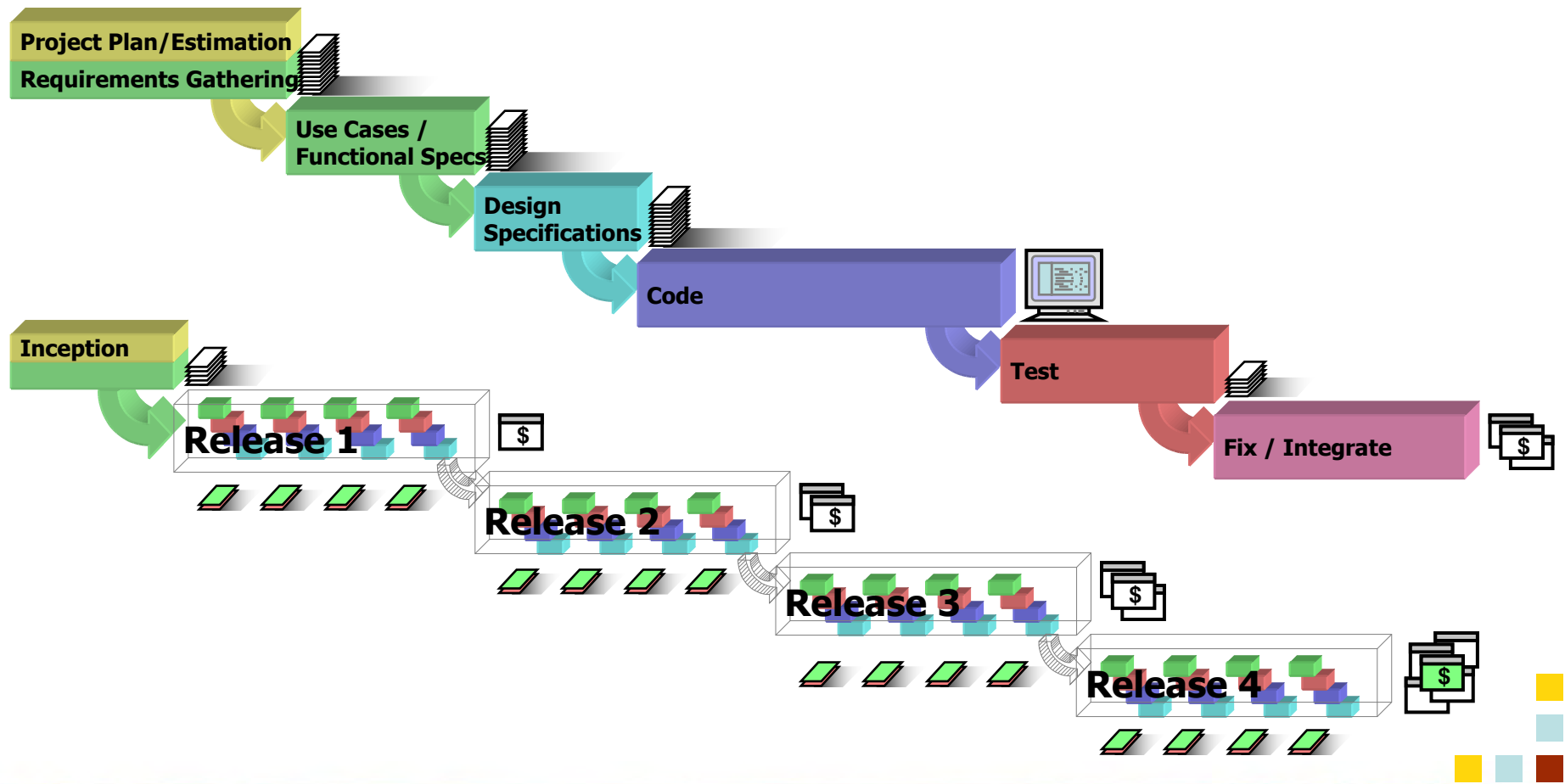
# Refactoring and Agile Techniques



# Development Styles



# Development Styles



# Agile History



# 2000

- XP | Extreme Programming (Kent Beck)
- DSDM | Dynamic System Development Method (Dane Paulkner)
- FDD | Feature Driven Development (Jeff DeLuca)
- SCRUM (Ken Schwaber)
- Crystal (Alistair Cockburn)
- Adaptive Software Development (Jim Highsmith)
- Lean Software Development (Mary Poppendieck)
- "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
- **Individuals and interactions** over processes and tools.
  - **Working software** over comprehensive documentation.
  - **Customer collaboration** over contract negotiation.
  - **Responding to change** over following a plan.

© 2001 Agile Alliance. <http://www.agilemanifesto.org>

## Agile manifesto



# Agile Design Principles



# Agile Design principles

- Do the **simplest** thing that could possibly **work**
  - A simple design **takes less time** to finish than a complex one
- Never add functionality **before** it is scheduled
  - Concentrate on what is **scheduled** for today only
- Agile designs are **emergent**, they're not defined up front



# Agile Design principles

- The **whole team** is responsible of the design of the system
- Design is so important you should **do it every day**
- The **Unit Tests** form much of your detailed design **documentation**
- Document complicated things
  - If it is complicated, then document it thoroughly.
  - Better yet, invest the time to **refactor** it so it is simple



# Agile Umbrella



# Agile Umbrella

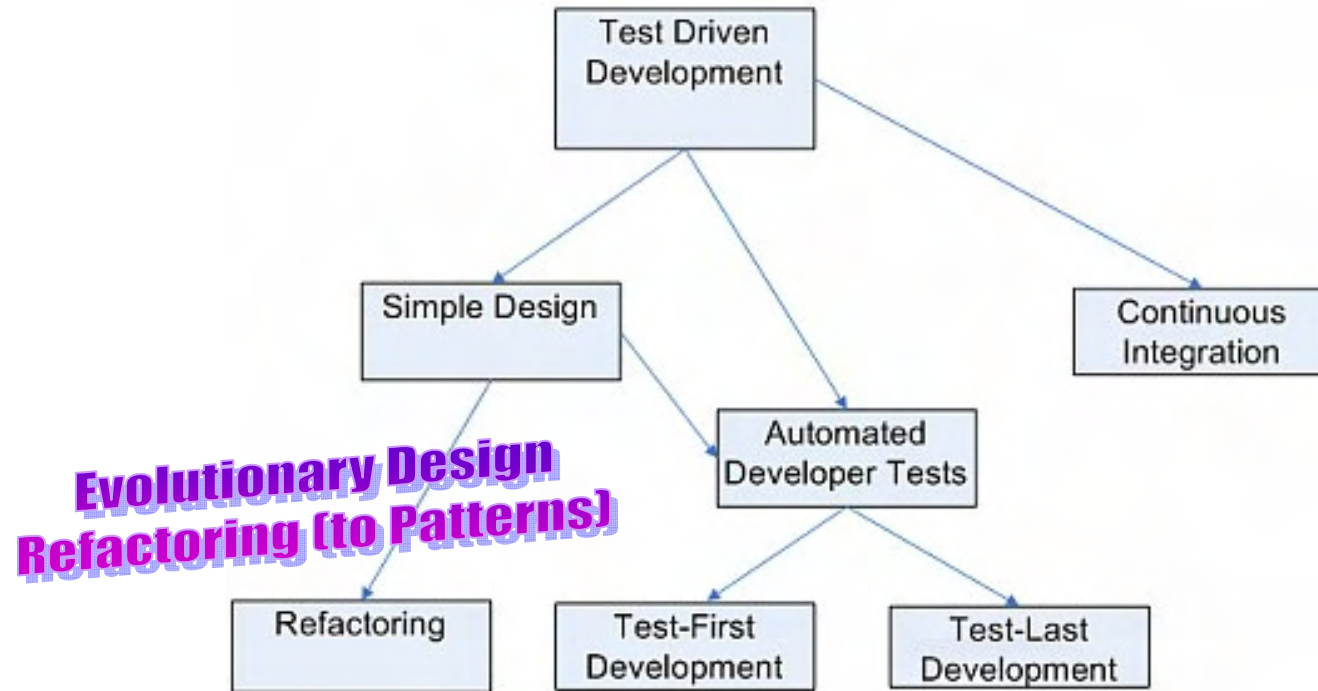
Just the right amount of documentation  
Collective Code Ownership    Pair Programming  
Test Driven Development  
Refactoring            Automated testing  
Continuous Integration  
Simple and Evolutionary Design  
Iterative Development  
Frequent Short Releases  
Collocation                            Stand ups  
Sustainable Pace                            Spikes



# Refactoring (to Patterns)



# Refactoring (to Patterns)



**Evolutionary Design  
Refactoring (to Patterns)**



# Refactoring



# Refactoring

---

---

- Refactoring is a development practice for **restructuring** an existing code, altering its internal structure without changing its external behavior.



# Refactoring

---

---

- Refactoring is a process of improvement to an existing software artifact.



# Refactoring

---

---

- New requirement, bug fix, behavior change and architecture change are **NOT** Refactoring.



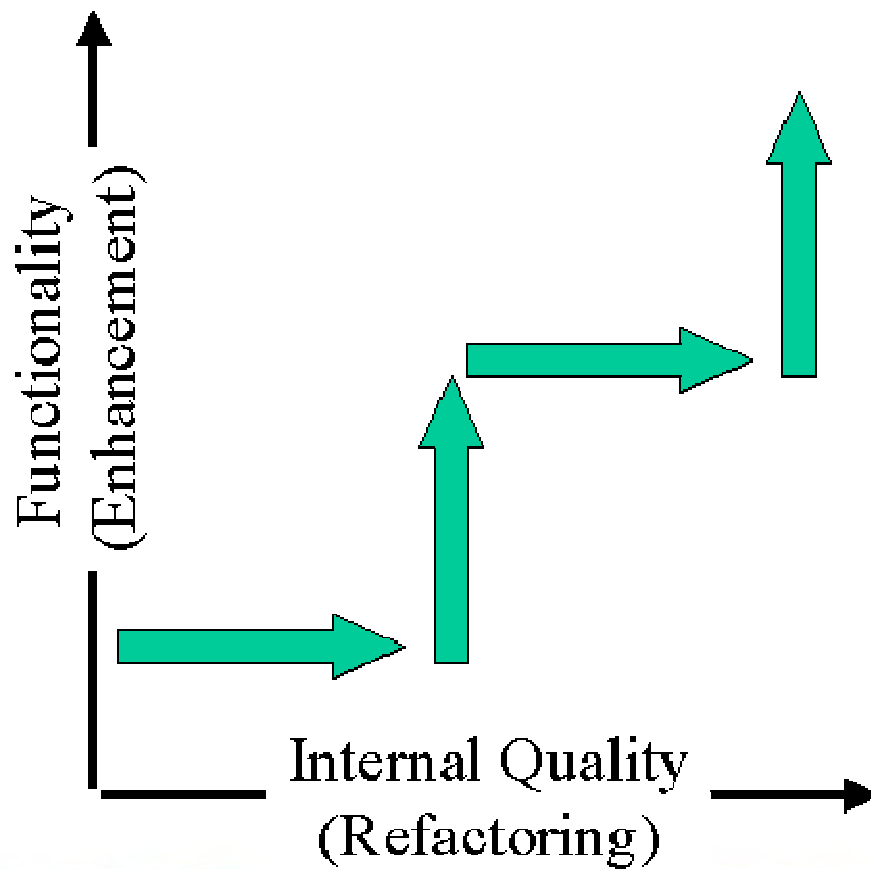
---

---

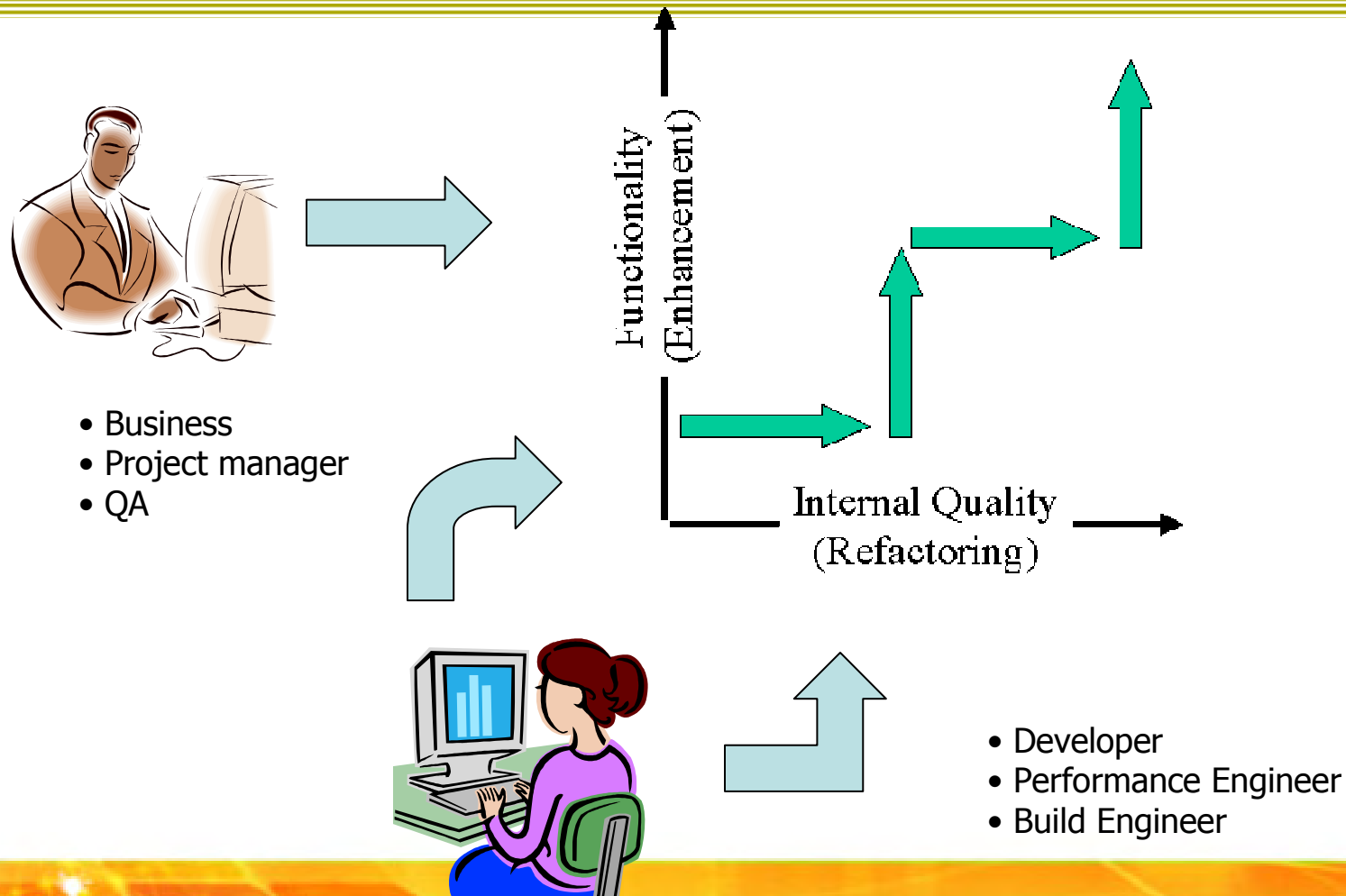
# Refactoring and Functional Requirements



# Refactoring and Functional Requirements



# Refactoring and Functional Requirements



---

---

# Refactoring and Test Driven Development

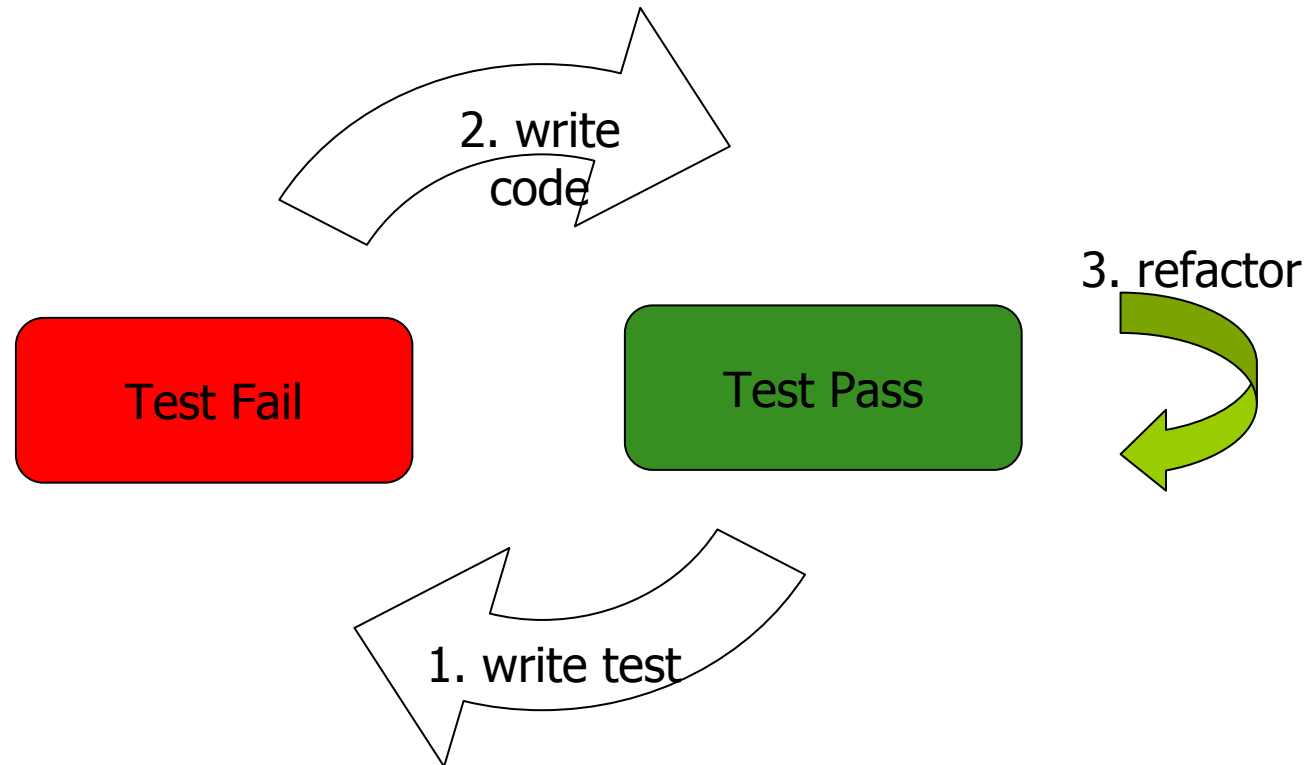


# Test Driven Development

“Test-Driven Development (TDD) is an evolutionary approach to development which instructs you to have test-first development intent. Basically, you start by writing a test and then you code to elegantly fulfill the test requirements.”

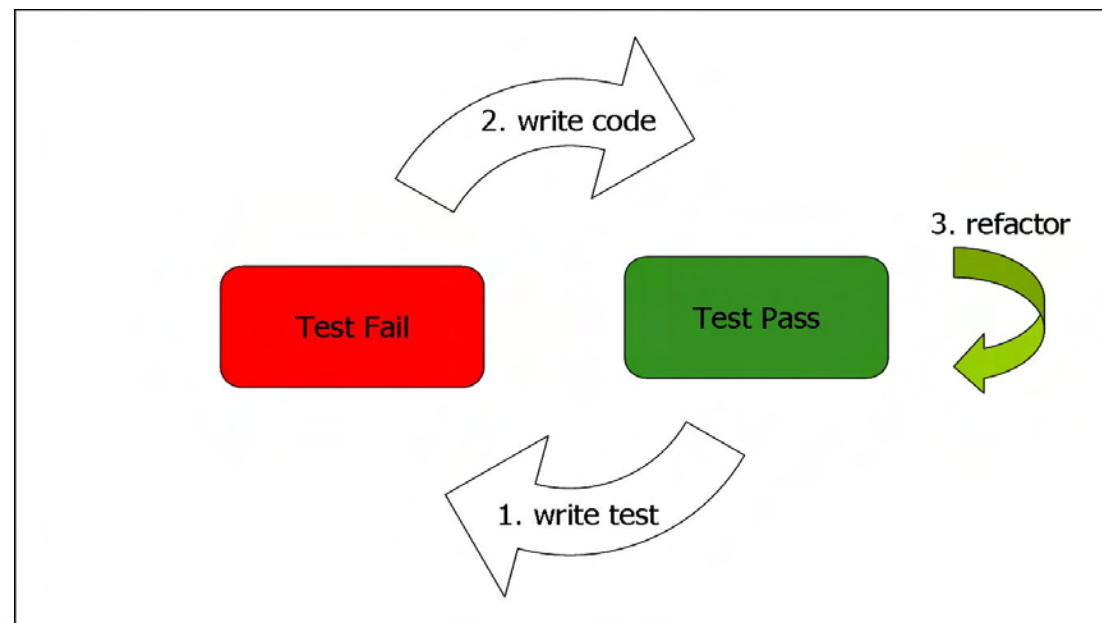


# TDD Cycle

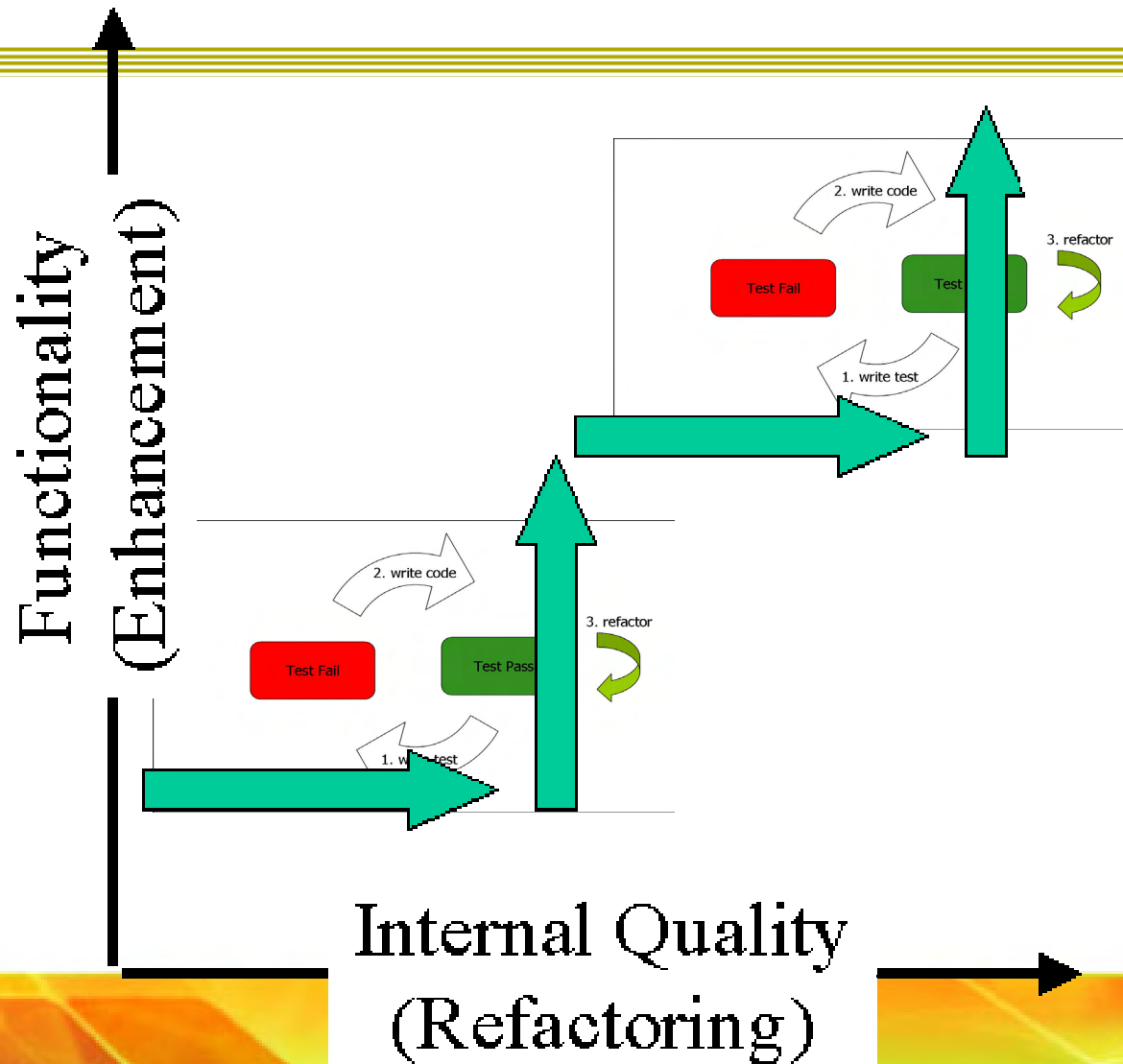


# Refactoring and TDD

- Refactor on every green bar
- Look for Code Smells to Refactor
- No Automated Developer Test -> Hard to Refactor



# Refactoring and TDD



---

---

# Refactoring and Simple Design



# Refactoring and Simple Design

---

---

- Think about tomorrow, but design, test and code for Today's need
- Produce simple artifacts that could be made flexible instead of Complex flexible artifacts



# Refactoring and Simple Design

---

---

- Do the **simplest** thing that could possibly **work**
- Evolve into elaborate solutions



---

---

# The Simplest Thing That Works



# The simplest thing that works

- Don't try to solve **big problems** in one go
  - Break them down into **little problems** and solve them one at a time
- Do the **simplest thing** that meets the **current requirement**



# The simplest thing that works

- Use **interfaces** / abstractions to insulate areas of risk or unknown
- Use **TDD + refactoring + interfaces** to keep the system extensible



# Evolutionary Design



# Evolutionary Design

---

---

- The complexity of your design should support the **current requirements** at hand (being built in this Iteration) and no more.



# Evolutionary Design

---

---

- Do not design for future flexibility, let new requirements "**pull**" more complex designs - Refactor later.



# Evolutionary Design

---

---

- Evolutionary Design should not be practiced without the ability to **refactor** and evolve the design.



# Code Smell



# Code Smell

---

---

Code smell is a hint that something can be improved somewhere in your code.



# Code Smell Samples



# Code Smell Samples

---

---

- Duplicated Code
- Methods too big
- Nested “if” statements
- Classes with too many instance variables
- Classes with too much code



# Code Smell Samples

---

---

- Strikingly similar subclasses
- Too many private (or protected) methods
- Similar looking code sections
- Dependency cycles
- Passing Nulls To Constructors
- Classes with too little code



# Refactoring (to Patterns)



# Refactoring (to Patterns)

---

---

- Simple Design -> Code Smell -> Refactor
- Refactoring (to Patterns) is the ability to transform a "Code Smell" into a positive design pattern



---

---

# Design Patterns



# Design Patterns

---

---

- **Proven solution** to a problem in context
- Classic solutions to common OO design problems
- Become a **common language** among developers



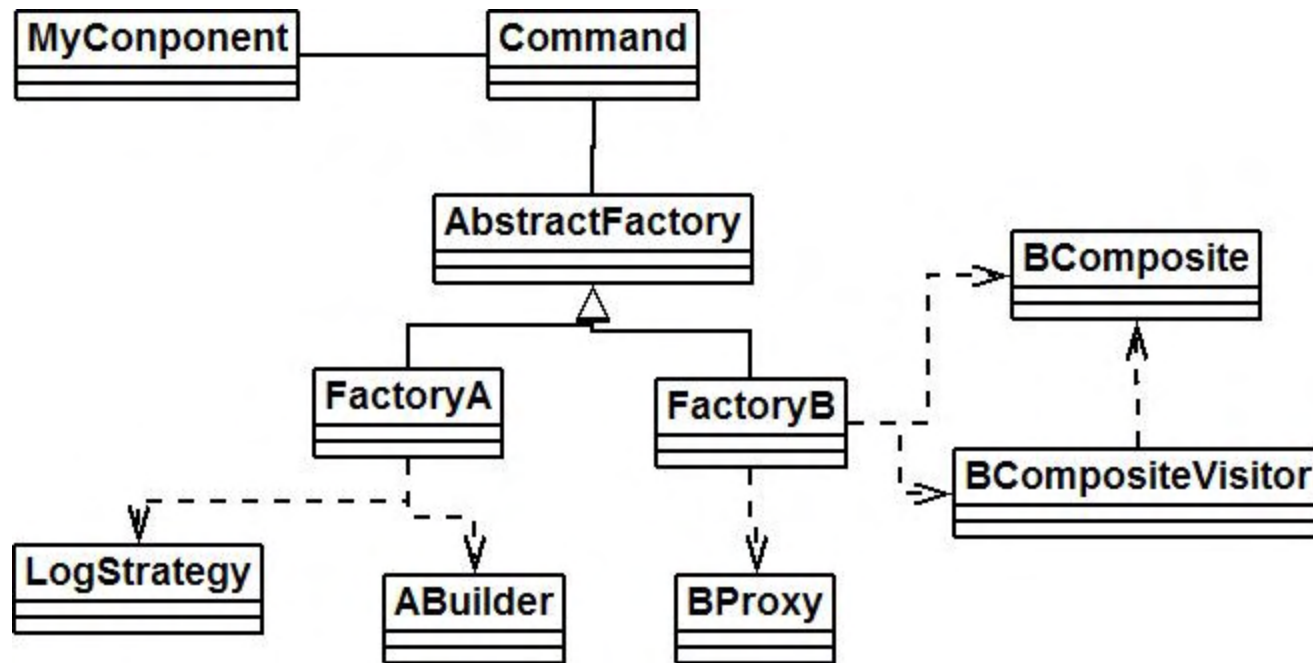
---

---

# Applying Design Patterns



# Applying Design Patterns - Upfront Design



# Applying Design Patterns - Upfront Design

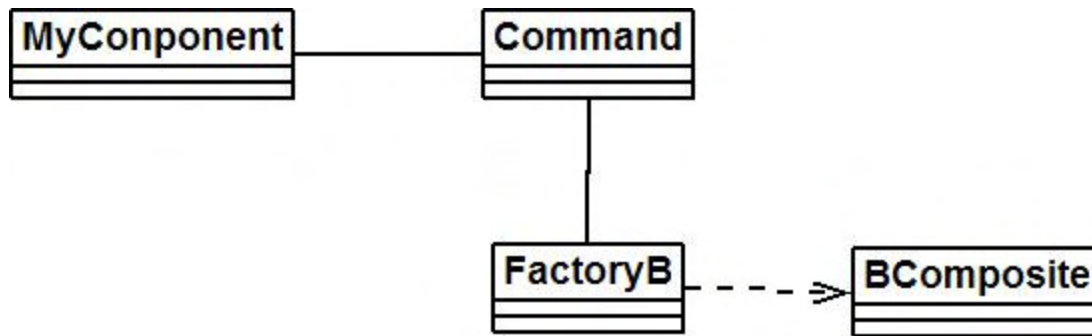
---

---

- Pre-emptive Design
- Sounds Great! But do I really need it?
- Wow, 3 weeks designing, beautiful diagrams!
- I generated lots of Pattern code (functional and test)



# Applying Design Patterns - Emergent Design



## Applying Design Patterns - Emergent Design

- Evolutionary Design
- Positive Patterns were “pulled” naturally
- Several components were just fine without Design Patterns
- Unit Test (TDD) validates the Refactoring to Patterns



# Case Study



---

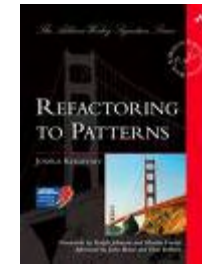
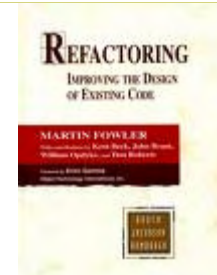
---

# Recommended Readings



# Recommended Readings

- ***Refactoring: Improving the Design of Existing Code*, Martin Fowler , Addison-Wesley, 2000.**
- **Refactoring to Patterns, Joshua Kerievsky, Addison Wesley, 2004.**
- **Design Patterns: Elements of Reusable Object-Oriented Software, Erich Gamma, Richard Helm , Ralph Johnson, John Vlissides, Addison-Wesley 1995**
- <http://www.thoughtworks.com/>
- <http://agilemanifesto.org/>



---

---

# Refactoring Tips and Suggestions



# Refactoring Tips and suggestions

---

---

- Design **every day**
- **Isolate** the **unknown** with **abstractions** / interfaces
- **Mock out** dependent systems for testing



# Refactoring Tips and suggestions

- **Refactor** code so it **communicates intent**
- **Test everything** – don't refactor without tests
- Focus on **simplifying complexity** rather than documenting it



# Session Learning's



# Session Learning's

---

---

- Understand Refactoring and TDD as Agile techniques
- Understand Agile Simplicity and Evolutionary Design Techniques



# Session Learning's

- Understand how simple design evolves into more elaborate design by means of TDD and Refactoring
- Understand what code smells are and how they can be Refactored into Design Patterns (when appropriate)



Questions?

---

---

**THANK YOU!**

