



codemanship
c o d e m a n s h i p

Coding Dojo World Tour (of London)

@jasongorman



June 14th, Bletchley Park
www.softwarecraftsmanship.org.uk

#sc2012



The screenshot shows a browser window with the address bar containing 'www.codemanship.co.uk'. The page features the Codemanship logo, a navigation menu with links for 'home', 'about', 'training', 'blog', and 'contact', and a quote: "It's not how fast we deliver that decides the winners & losers. It's how fast we learn from what we deliver". Below the quote is a photo of people working at computers. The main content area is divided into two columns. The left column has the heading "Learning. Caring. Practicing. Sharing" and "The Four Pillars Of Software Craftmanship", followed by a list of services: Test-driven Development, Refactoring, Continuous Integration & Build Automation, and OO, Component-based & Service-Oriented. The right column features a badge for "software craftsmanship 2012" with the text "Proud organisers of the original Software Craftmanship conference". Below this are sections for "Coding Dojo Tour" (mentioning Jason Gorman's visit in March) and "Slow & Dirty" (mentioning Jason Gorman's presentation at SPA2011). The bottom of the right column has the heading "Dependable Dependencies".



"It's not how fast we deliver that decides the winners & losers. It's how fast we learn from what we deliver"

home | about | training | blog | contact



"Learning. Caring. Practicing. Sharing"
The Four Pillars Of Software Craftmanship

We train, coach and consult in:

- Test-driven Development
- Refactoring
- Continuous Integration & Build Automation
- OO, Component-based & Service-Oriented



Proud organisers of the original [Software Craftmanship](#) conference

Coding Dojo Tour

Jason Gorman will be visiting organisations throughout the week of March 19-23 to run free coding dojos [More...](#)

Slow & Dirty

Jason Gorman's presentation from SPA2011 makes a strong case for raising quality if teams want to go faster [Download Slides \(PDF\)](#)

Dependable Dependencies



youtube.com/user/parlezuml

Firefox

parlezuml - YouTube

www.youtube.com/user/parlezuml

very simple code

Codemanship's Test-driven Developm...
parlezuml 3,674 views 1 year ago

Jason Gorman demonstrates the basic practices of Test-driven Development in Eclipse, using JUnitMax to run the tests in the background. Visit <http://www.codemanship.com>

Codemanship's Bonus Code Smell - L...
parlezuml 776 views 1 year ago

Jason Gorman quickly illustrates how to apply the Collapse Hierarchy refactoring to eliminate a lazy subclass. Download the source code from <http://bit.ly/ddfnmf> For t...

Codemanship's Code Smell Of The we...
parlezuml 958 views 1 year ago

Lazy classes add little value for the maintainance they incur. In this example, Jason Gorman illustrates



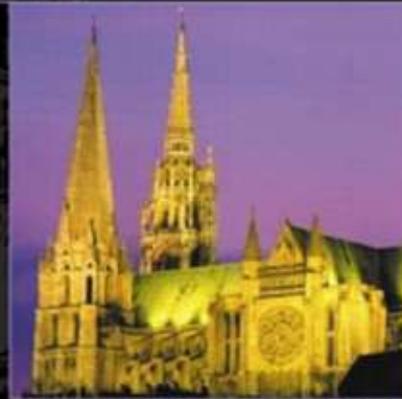
codemanship
codemanship

The Addison-Wesley Signature Series

TEST-DRIVEN DEVELOPMENT

BY EXAMPLE

KENT BECK



codemanship
codewarehouse

The Addison-Wesley Signature Series

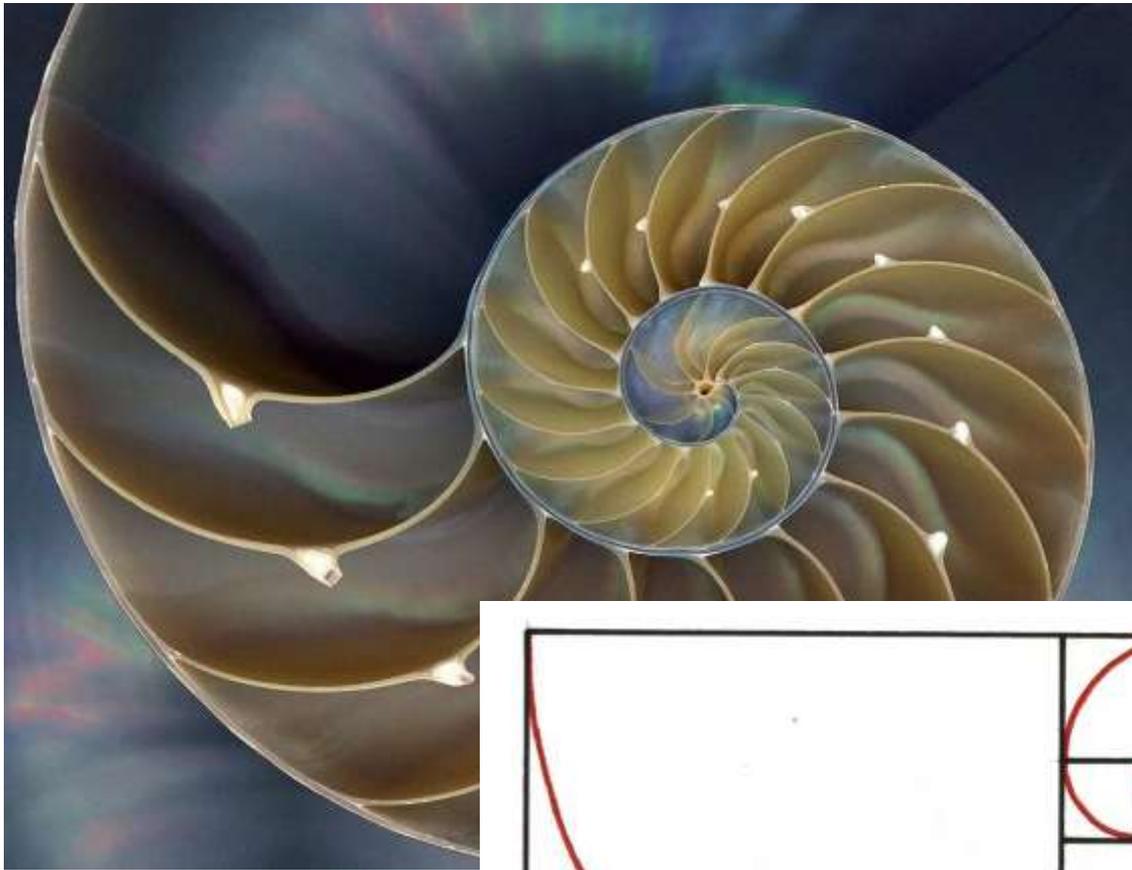
BOOK A KENT BECK SIGNATURE
Kent Beck
SIGNATURE

GROWING OBJECT-ORIENTED SOFTWARE, GUIDED BY TESTS

STEVE FREEMAN
NAT PRYCE



codemanship
codemanship

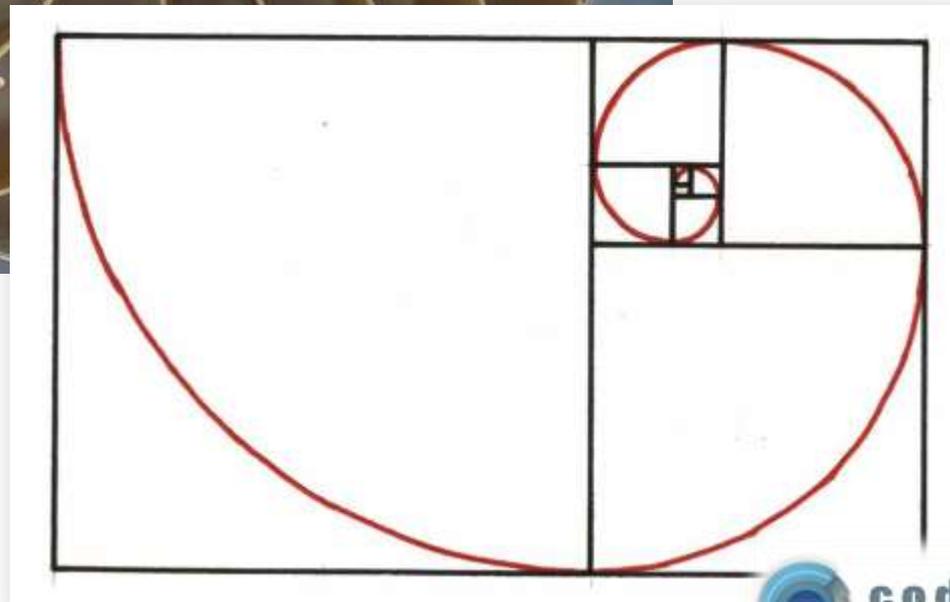


Fibonacci Sequence Generator

$$F_n = F_{n-1} + F_{n-2},$$

seed values

$$F_0 = 0, F_1 = 1.$$





FizzBuzz

Write some code that will generate a string of integers, starting at 1 and going up to 100, all separated by commas.

Substitute any integer which is divisible by 3 with “Fizz”, and any integer which is divisible by 5 with “Buzz”, and any integer divisible by 3 and 5 with “FizzBuzz”

1,2,Fizz,4,Buzz,Fizz,7,8,Fizz,Buzz,11,Fizz,13,14,FizzBuzz... etc



By the Book

Perform Fibonacci Generator or FizzBuzz without breaking ANY of these rules:

1. Start by writing a failing test
2. Write the simplest code to pass the test
3. Refactor to remove any duplication
4. Write the assertion first and work backwards
5. Run the test to see the assertion fail
6. Don't refactor with failing tests
7. Write meaningful tests
8. Keep your test and model code separate
9. Organise your test code to make it easy to find
10. Isolate your tests so they can pass when run in any order
11. Triangulate interesting behaviour
12. Write tests that only have one reason to fail



From Hell

Break ALL these rules in the fewest number of passing tests:

1. Start by writing a failing test
2. Write the simplest code to pass the test
3. Refactor to remove any duplication
4. Write the assertion first and work backwards
5. Run the test to see the assertion fail
6. Don't refactor with failing tests
7. Write meaningful tests
8. Keep your test and model code separate
9. Organise your test code to make it easy to find
10. Isolate your tests so they can pass when run in any order
11. Triangulate interesting behaviour
12. Write tests that only have one reason to fail



As If You Meant It

Perform Fibonacci Generator or FizzBuzz, writing the code to pass the tests INSIDE the tests, and discover the design entirely through refactoring

(first run by Keith Braithwaite at Software Craftsmanship 2009)