

What is FitNesse and should I use it?

Gojko Adzic
gojko@gojko.com
<http://gojko.net>
@gojkoadzic

Plan for today:

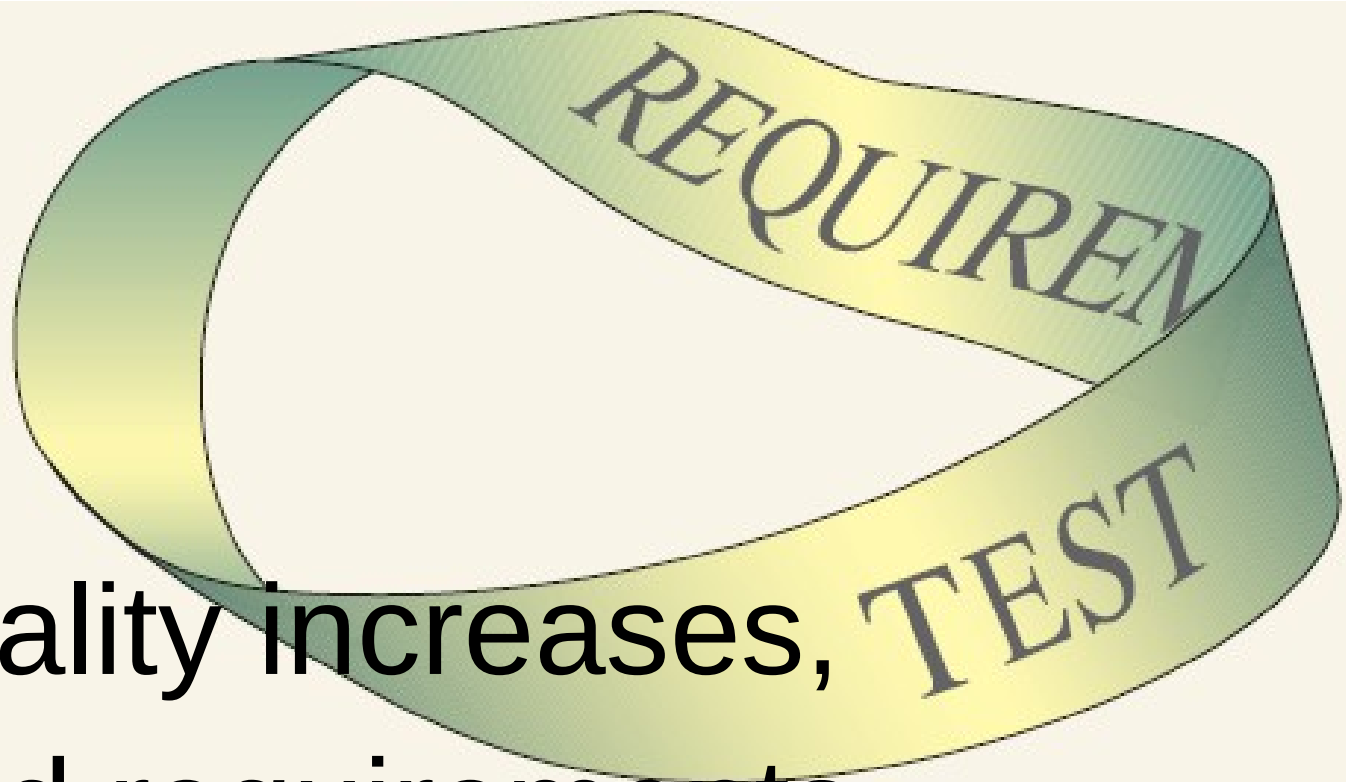
- Where FitNesse works best
- What is it anyway?
- How people misuse it
- How not to shoot yourself in the foot

Inspection to find defects is waste,
inspection to prevent defects is
essential

Mary and Tom Poppendieck
Lean Software Development

One of the most effective ways of testing requirements is with test cases very much like those for testing the completed system

Donald Gause and Gerald Weinberg
Exploring Requirements - **1989!**

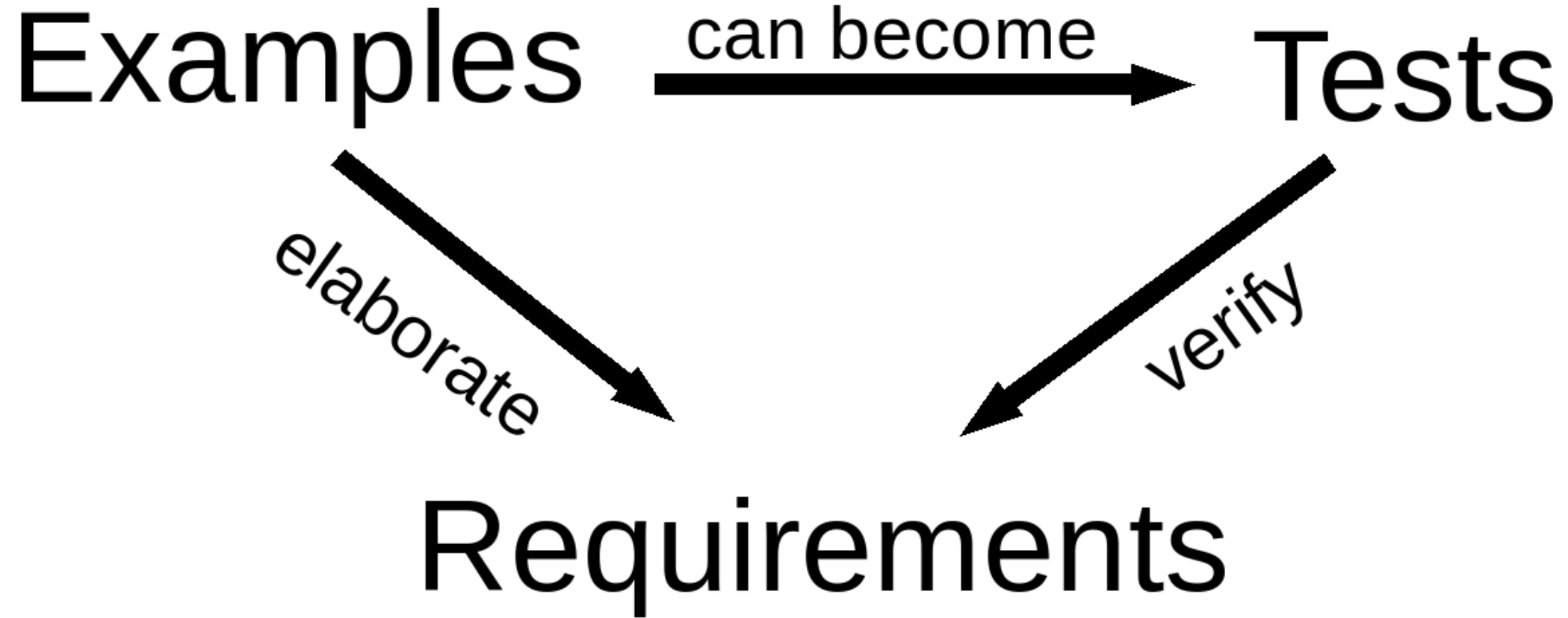


As formality increases,
tests and requirements
become indistinguishable.

Robert C. Martin and Grigori Melnik

Tests and Requirements, Requirements and Tests: a Möbius Strip

IEEE Software January/February Issue 2008



Unit test = target for development:

```
@Test
public void testCanDoubleDown() {
    HandScore handScore = new HandScore(2, 13, 3);
    Assert.assertTrue(ANY.canDoubleDown(handScore));
    Assert.assertFalse(NONE.canDoubleDown(handScore));
    Assert.assertTrue(SOME_SOFT_AND_HARD.canDoubleDown(handScore));
    Assert.assertFalse(SOME_HARD.canDoubleDown(handScore));

    handScore = new HandScore(2, 14, 4);
    Assert.assertTrue(ANY.canDoubleDown(handScore));
    Assert.assertFalse(NONE.canDoubleDown(handScore));
    Assert.assertFalse(SOME_SOFT_AND_HARD.canDoubleDown(handScore));
    Assert.assertFalse(SOME_HARD.canDoubleDown(handScore));

    handScore = new HandScore(2, 12, 20);
    Assert.assertTrue(ANY.canDoubleDown(handScore));
    Assert.assertFalse(NONE.canDoubleDown(handScore));
    Assert.assertTrue(SOME_SOFT_AND_HARD.canDoubleDown(handScore));
    Assert.assertFalse(SOME_HARD.canDoubleDown(handScore));
}
```

A test can be our light at the end of the tunnel



But what if it's actually a train?



But they aren't really readable

```
@Test
public void testPlayersGetNewStatus() throws GameException {
    expectGameStart();
    Player p1 = new Player(BigDecimal.valueOf(17),BigDecimal.valueOf(17),"");
    Player p2 = new Player(BigDecimal.valueOf(18),BigDecimal.valueOf(18),"");
    EasyMock.expect(gameRules.getPlayers(gameStatus1)).andReturn(new Player[]{p1, p2}
    //ExecutionContext context = new ExecutionContext(GAME_ID, gameStatus1, v, "A", Tabl
    EasyMock.expect(gameRules.getStatusForPlayer(EasyMock.eq(p1), (ExecutionContext) Easy
    EasyMock.expect(gameRules.getStatusForPlayer(EasyMock.eq(p2), (ExecutionContext) Easy
    EasyMock.expect(gameRules.execute(new ExecutionContext(GAME_ID, gameStatus, v, "a", t
        (GameWalletImpl)EasyMock.anyObject()), EasyMock.eq(command))).andReturn(new E
    dd.notifyPlayer(EasyMock.eq(t.getId()),
        EasyMock.eq(BigDecimal.valueOf(17)),
        EasyMock.eq(new GameStatusDocument(1L, gameStatus1, "12345"))));
    EasyMock.expectLastCall();
    dd.notifyPlayer(EasyMock.eq(t.getId()),
        EasyMock.eq(BigDecimal.valueOf(18)),
        EasyMock.eq(new GameStatusDocument(1L, gameStatus, "12345"))));
    EasyMock.expectLastCall();
    EasyMock.replay(a, dd, gameStatus, gameStatus1, gameRules);
    executeCommand();
    EasyMock.verify(dd);
}
```

Here's what I thought...

- When the order is in a “pending” state, we first check the account, and if approved move it to “confirmed” state
- When the order is manually confirmed, it moves from the “pending” to “confirmed” state even if the account does not have enough funds
- When the order is in a “pending” state for two days, we send an alert

Here is what they saw:

- ghorgh [the] [order] 'oH Daq [a] ["pending"]
[state] maH wa'DIch [check] [the] [account] 'ej
chugh [approved] vIH 'oH Daq ["confirmed"]
[state] ghorgh [the] [order] 'oH [manually]
[confirmed] 'oH vIHtaH vo' [the] ["pending"] Daq
["confirmed"] [state] 'ach chugh [the] [account]
ta'taH ghobe' ghaj yap [funds] ghorgh [the]
[order] 'oH Daq [a] ["pending"] [state] vaD cha'
jajmey maH ngeH [an] [alert]

FitNesse allows us to specify automated but human readable business tests!

The prize pool is divided among the winners using the following distribution for winning combinations (number of correct hits out of six chosen numbers).

The example below is for \$2M payout pool.

Prize Distribution for Payout Pool	2,000,000	
Winning Combination	Pool Percentage?	Prize Pool?
6	68	1,360,000
5	10	200,000
4	10	200,000
3	12	240,000

A PLAYER CAN BUY TICKETS FOR DIFFERENT DRAWS AND THEY WILL ALL SHOW UP SEPARATELY ON HIS STATEMENT.

Player buys a ticket with numbers	1,5,17,44,22,19	for the draw on	01/01/2009
-----------------------------------	-----------------	-----------------	------------

Player buys a ticket with numbers	1,5,17,44,22,19	for the draw on	02/01/2009
-----------------------------------	-----------------	-----------------	------------

Player buys a ticket with numbers	10,25,13,42,19,18	for the draw on	01/01/2009
-----------------------------------	-------------------	-----------------	------------

Player receives a statement	
-----------------------------	--

draw	numbers
01/01/2009	1,5,17,44,22,19
01/01/2009	10,25,13,42,19,18
02/01/2009	1,5,17,44,22,19

How FIT works

Test Table

The prize pool is divided among the winners using the following distribution for winning combinations (number of correct hits out of six chosen numbers).

The example below is for \$2M payout pool.

Prize Distribution for Payout Pool	2,000,000	
Winning Combination	Pool Percentage?	Prize Pool?
6	68	1,360,000
5	10	200,000
4	10	200,000
3	12	240,000

Fixture

```
public class PrizeDistributionForPayoutPool:fit.ColumnFixture {
    private WinningsCalculator wc = new WinningsCalculator();
    public int winningCombination;
    public int PoolPercentage()
    {
        return wc.GetPoolPercentage(winningCombination);
    }
    public decimal? payoutPool;
    public decimal PrizePool()
    {
        if (payoutPool == null) payoutPool = Decimal.Parse(Args[0]);
        return wc.GetPrizePool(winningCombination, payoutPool.Value);
    }
}
```

Domain code

```
public class WinningsCalculator
{
    public int GetPoolPercentage(int combination)
    {
        switch(combination) {
            case 6: return 68;
            case 5: return 10;
            case 4: return 10;
            case 3: return 12;
            default: return 0;
        }
    }
    public decimal GetPrizePool(int combination, decimal payoutPool)
    {
        return payoutPool * GetPoolPercentage(combination) / 100;
    }
}
```

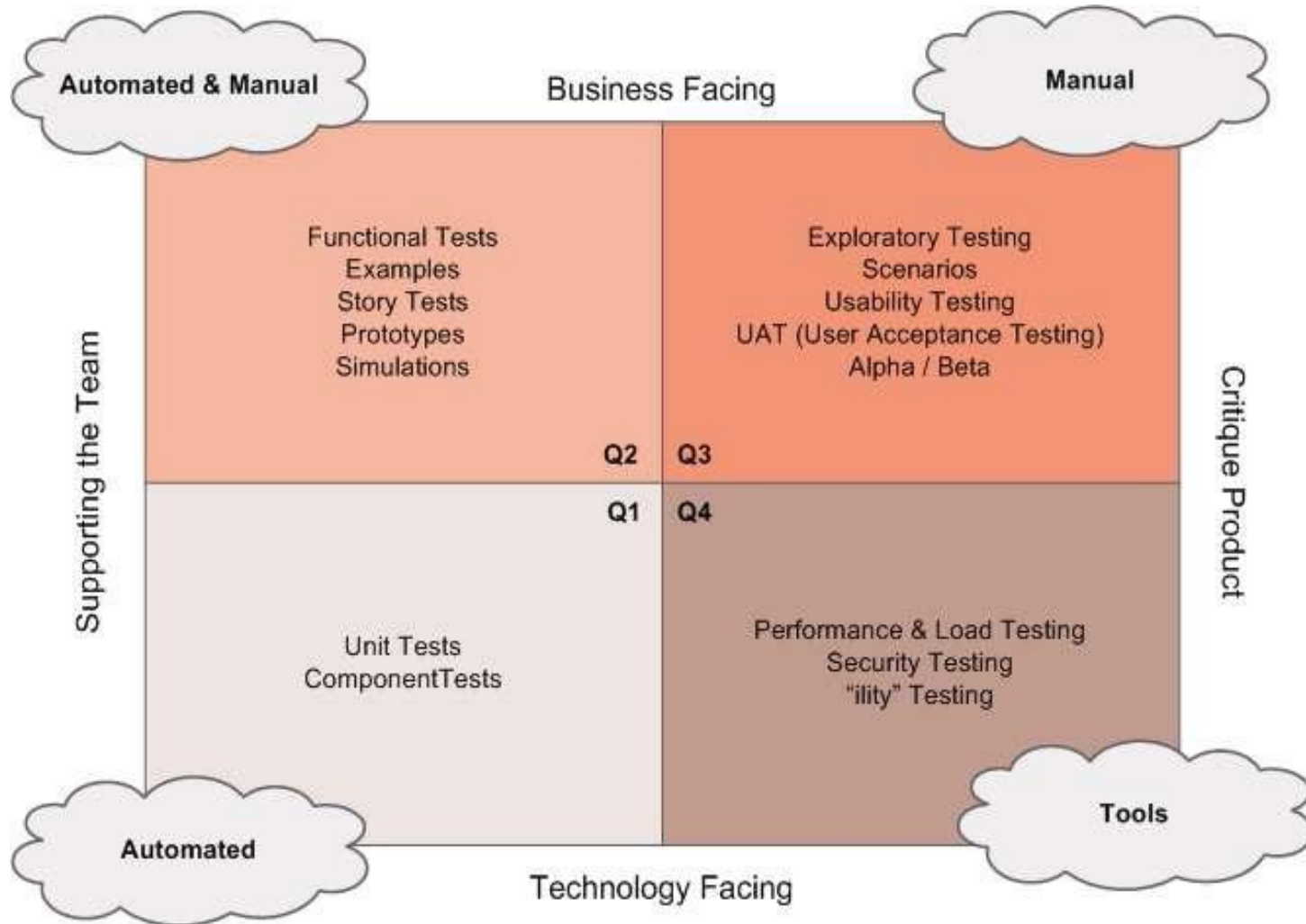
FitNesse

- Ward Cunningham wrote FIT in 2002 – no IDE/management tool
- Robert Martin and Micah Martin wrote FitNesse in 2005:
 - Wiki
 - Test suites, includes, symbolic links, markup variables...
 - Integrated reporting and execution
 - Integrates with almost anything easily

So what is it anyway?

- An opensource tool to manage executable specifications (acceptance criteria – tests) for development
- A collaboration tool for business people, developers and testers
- Works as a Web server, use it through a browser
- Supports Java, .NET, C++, Smalltalk, Python...

Where it really works



- Agile testing quadrants idea by Brian Marick, picture by Janet Gregory

It's excellent for:

- Defining and organising tests that are used as acceptance criteria for the project/iteration/piece of code
- Managing a “live” documentation of the project business functionality
- Collaborating between business people, testers and developers

It's usable (but not really good) for:

- Managing lots of different types of tests (eg ui tests, database tests)
- Integration testing
- Testers or developers working in isolation
 - Pure regression testing

It's definitely not:

- Usable for Unit testing
- Built for general purpose test management
- Suitable for performance testing
- Suitable for GUI testing
- **A REPLACEMENT FOR QTP, LOAD RUNNER, WIN RUNNER OR ANYTHING LIKE THAT!**

How to use it properly:

- focus on *what*, not on *how*
 - Distil the specification from scripts
- use the business language
- use realistic examples
- make pages self-explanatory
- SMART
- Specific, Measurable, Achievable, Relevant, Time-bound

How - script

rate change date is 23/5

due date	rate	amount
22/4	5.40%	340.2
22/5	5.40%	340.2
22/6	4.90%	308.7

rate change date is 20/5

due date	rate	amount
22/4	5.40%	340.2
22/5	5.40%	308.7
22/6	4.90%	308.7

What - specification

due date	rate changed	change reflected in period
22/5	23/5	no
22/5	20/5	yes

Often easier to spot gaps:

due date	rate changed	change reflected in period
22/5	23/5	no
22/5	20/5	yes
22/5	22/5	????

Another example of “how”

- Mike logs on
- Mike browses to the books page
- Mike adds “Perfect Software” to the cart
- Mike adds “Agile testing” to the cart
- Mike goes to check-out
- Mike gets offered free delivery

what are we specifying here?

Free delivery

Free delivery is offered to VIP customers once they reach a certain number of books in their cart. Regular customers, and VIP customers beneath this threshold, do not receive free delivery.

Examples

Given the threshold for free delivery is 10 books or more, then we expect the following:

Type of customer	Number of books in cart	Free Delivery?
VIP	8	No
VIP	9	No
VIP	10	Yes
VIP	11	Yes
VIP	12	Yes
Regular	8	No
Regular	9	No
Regular	10	No
Regular	11	No
Regular	12	No
.....		

As a rule of thumb:

- FitNesse pages should be relatively short
 - Specification must be easy to understand
- Stuff should not repeat across pages
 - Don't copy and paste, make specification clear and focused
- Fixtures should be very thin
 - Just an adapter to your code
 - Shouldn't contain logic
 - Use them to script test workflow

Best practices to keep tests good:

- Keep in the same SCM as the code
- Keep them organised and easy to find
- Evolve the language consistently
- Clean up periodically

Common symptoms of problems

- Technical tests
 - Reflect the way the code was written
 - Use technical jargon/class names
 - Favour reuse over clarity

com.mycompany.PlayerRegistrationFixture				
username	password	accountTemplate	security code	create()
Mike	mmikke	com.mycompany.TemplateRegular	1329	OK
Tom	tom111	com.mycompany.TemplatePriority	1132	OK
Tom	tom111	com.mycompany.TemplatePriority	1111	OK
Tom	tom222	com.mycompany.TemplatePriority	1111	UniqueConstraintViolation

Common symptoms of problems

- Long/complex tests
 - Specify too much (how not what)
 - Check for every possible case
 - These tests are important, but not as a spec
 - copy/paste from other tests things that aren't really needed
- Distil the specification

Common symptoms of problems

- Lots of tests with minor differences in some values
 - copy/paste?
 - How, not what?
- Distil “what”, create a single test out of the whole group

Interdependent tests

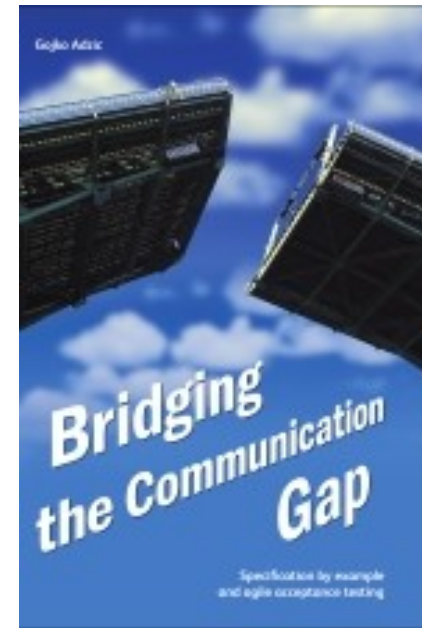
- Copy/paste?
 - Common dependencies?
 - Parts used to set up/clean up after related tests
- Extract common dependencies into test suites
- Push technical activities into fixtures

Common symptoms of problems

- Tests that fail intermittently
 - Unreliable
 - Asynchronous processes?
 - External dependencies?
 - Problems in the implementation?
 - Data dependency?
 - Random values?
- Work out what's wrong and fix it

Bridging the Communication Gap

- learn how to improve communication between business people and software implementation teams
- find out how to build a shared and consistent understanding of the domain in your team
- learn how to apply agile acceptance testing to produce software genuinely fit for purpose
- discover how agile acceptance testing affects your work whether you are a programmer, business analyst or a tester
- learn how to build in quality into software projects from the start, rather than control it later



<http://www.acceptancetesting.info>

upcoming events....

- testing...
- 27/5: Acceptance testing tools roundup
- 23/6: Testable software is good software
- .NET
- 19/5: F# and Lucene.NET
- 16/6: Rake.NET and Iron Ruby