

Test Driven Requirements

Roland Cuellar
Senior Practice Manager, Lean-Agile Practice
CC Pace
703-251-6950 Direct
www.ccpace.com
Roland.Cuellar@ccpace.com



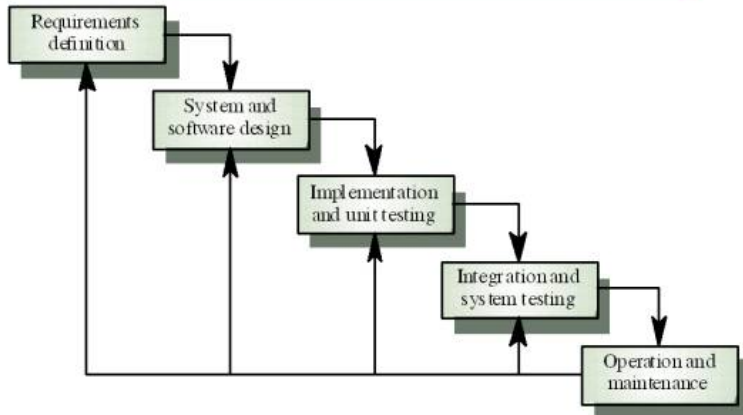
Services

- Lean-Agile Consulting
- Global Markets & Investment Banking Consulting
- Mortgage Consulting
- Retail Banking Consulting
- Learning Institute Training
- Staffing

Selected Clients

- Capital One
- The Carlyle Group
- Chase Manhattan Mortgage Corp.
- Credit Suisse First Boston (CSFB)
- Fleet Mortgage
- Freddie Mac
- Lehman Brothers

Waterfall model



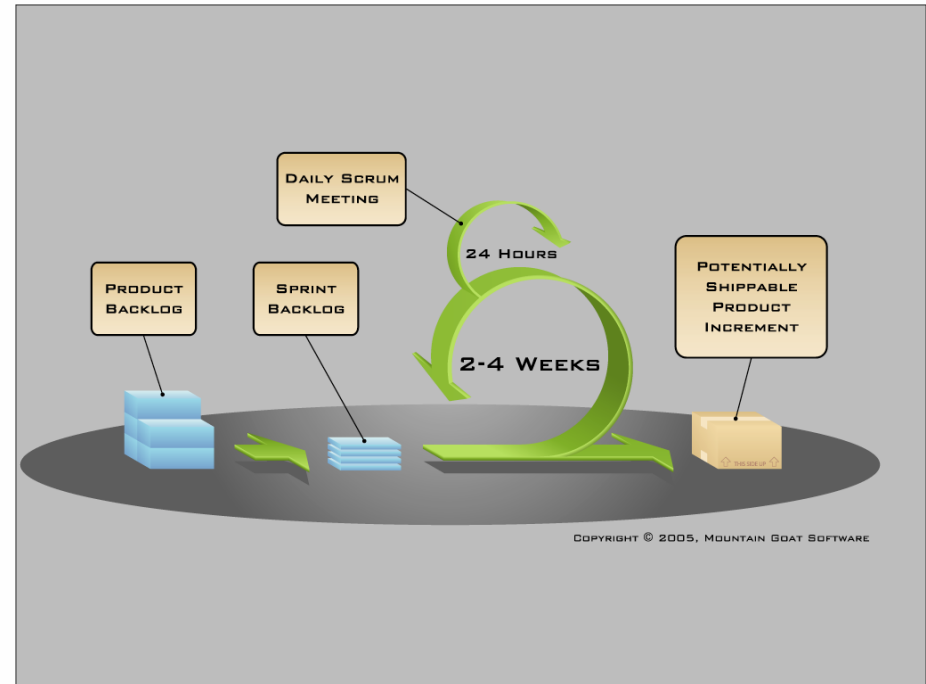
©Ian Sommerville 1995

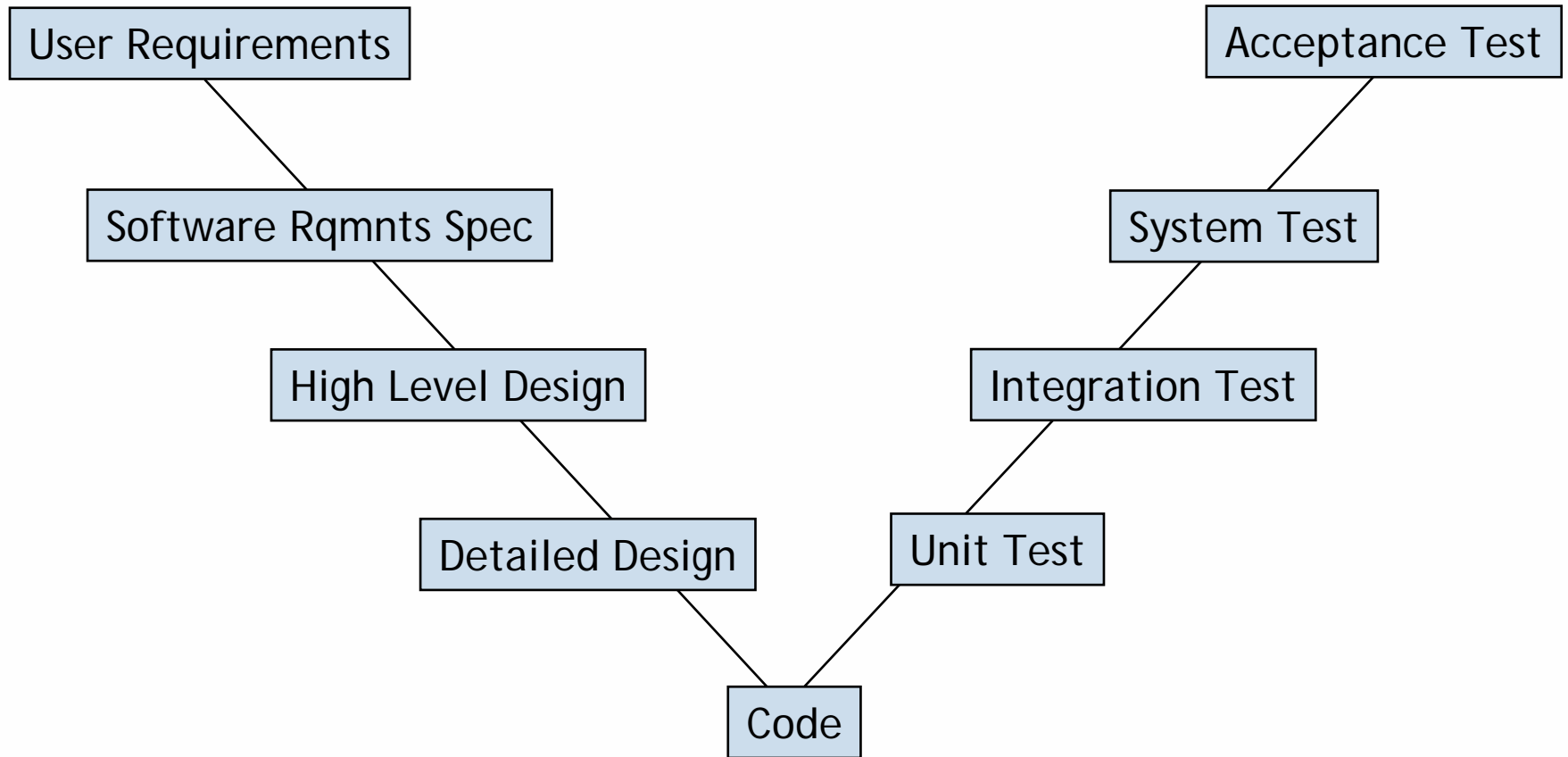
Software Engineering, 5th edition, Chapter 1

511

- Perform one phase at a time
- For all requirements
- Single Delivery at the End

- Perform all phases at a time
- For a few requirements
- Many interim deliveries



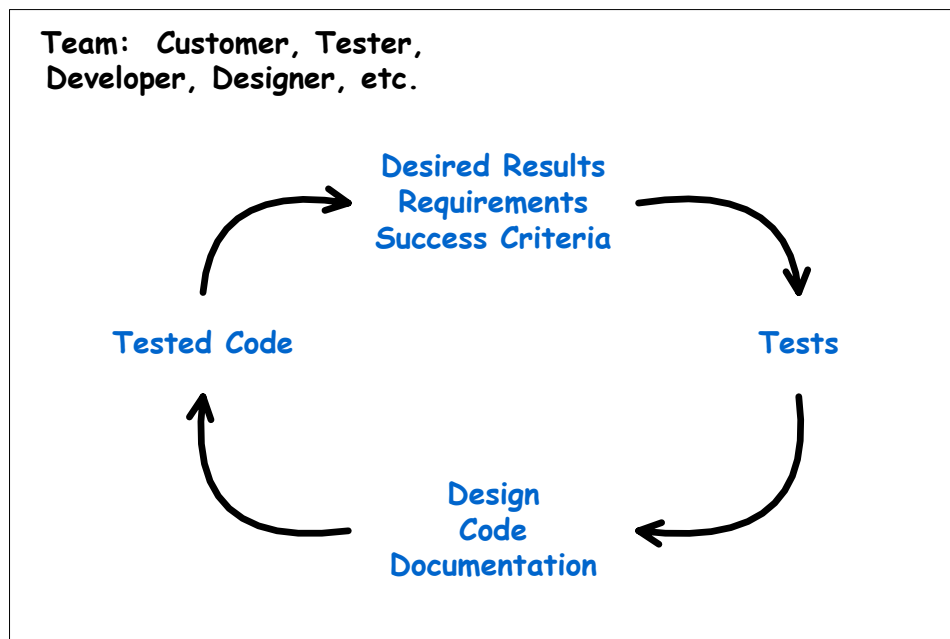


On a waterfall project, when do we validate that the system meets the user's requirements?

- The BA makes certain assumptions about business needs and creates requirements
- The Developer reads the requirements and makes a different set of assumptions to create code. *(Unit tests always pass because they test against the developers assumptions!)*
- The Tester reads the requirements and makes a third set of assumptions to create test plans
- We don't sync up assumptions until later in the process
- We write code that does not solve the customer's problem and we do not write code that does
- We may write tests that do not test for the solution in the eyes of the customer and write tests that don't

- Let's synchronize our assumptions
 - Agree up front on business rules
 - Agree up front on test cases and data variants
 - Agree up front on desired results
 - Let's document this stuff up as a test plan
 - Let's code to the test plan
 - Brilliant!

*TD: Test Driven
Requirements and
Development* →



- **It's not about testing! It's about ...**
 - Forcing conversations earlier
 - Getting into the details earlier
 - Understanding what the customer needs before we actually do it
 - Avoiding last-minute surprises
 - Writing the code that needs to be written
 - Not writing the code that does not
 - Taking the customer to the next level of detail
 - Discovering the solution, often through iteration

- **TDD also impacts**
 - Design
 - Testability
 - Estimation

- Have a document that makes sense to them
- Know what is going to be on the test and can code to that
- Have a greater chance of covering the cases
- Build a catalog of regression tests
- Avoid last-minute surprises of divergent assumptions
- Can do more meaningful unit testing

“If you can’t write a test for what you are about to code, then you shouldn’t be coding it”

- 2003 study by Laurie Williams & Boby George at NC State
 - Studied 24 pair programmers in two groups
 - Group A performed TDD at the unit test level
 - Group B did typical back-end unit test (*almost*)

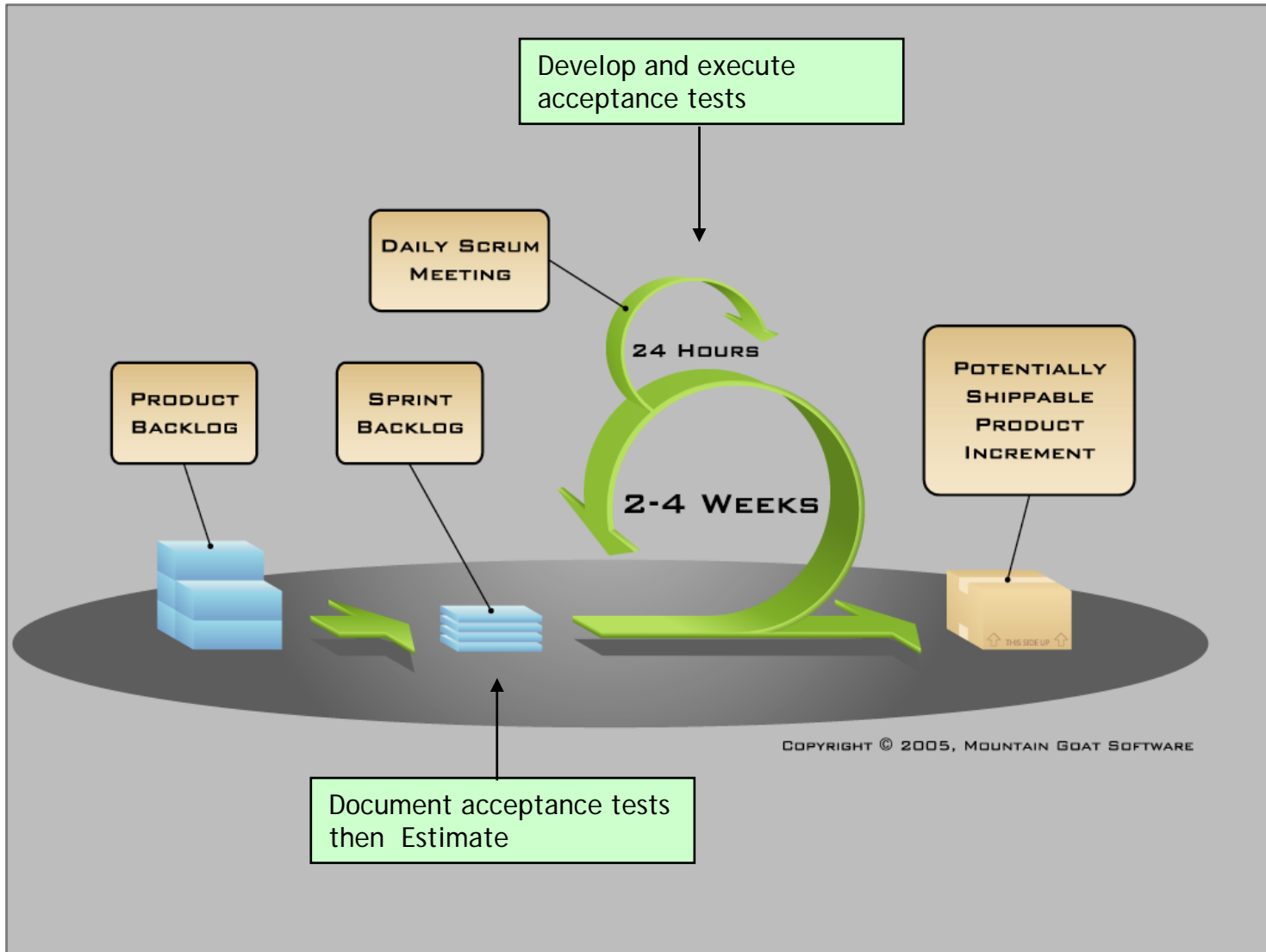
- Results
 - The TDD group passed 18% more black-box functional tests
 - And developed a suite of automated test cases
 - But took 16% longer to develop the solution
 - The traditional group was faster, but passed fewer tests, and did not actually write and execute many of the test cases

“All pairs turned in their programs when they felt it would run correctly. The TDD pairs did not feel they were done until they wrote higher quality code with a good set of automated test cases. The control group pairs felt they were done with lower quality code, primarily without any worthwhile automated test cases.”

- 87% of the developers felt that TDD led to better requirements understanding
- 95.8% felt that TDD reduced their debugging effort
- 92% believed that TDD led to higher quality code
- 79% felt that TDD promoted simpler design
- But 56% felt that getting into the TDD mindset was difficult

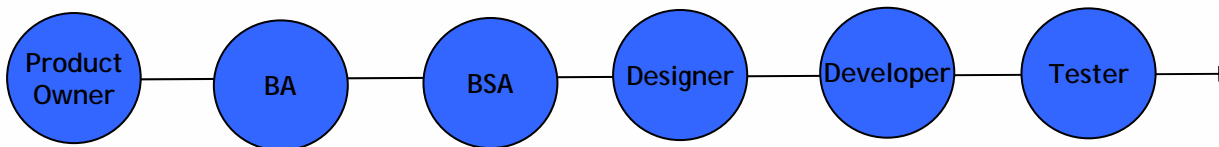
- **Agile does not expect**
 - Complete and final requirements up front
 - Requirements can be added, deleted, or changed

- **This implies that TDD on Agile projects**
 - Cannot expect complete system acceptance criteria up front
 - Acceptance criteria can be set for user stories
 - Acceptance criteria may change between iterations

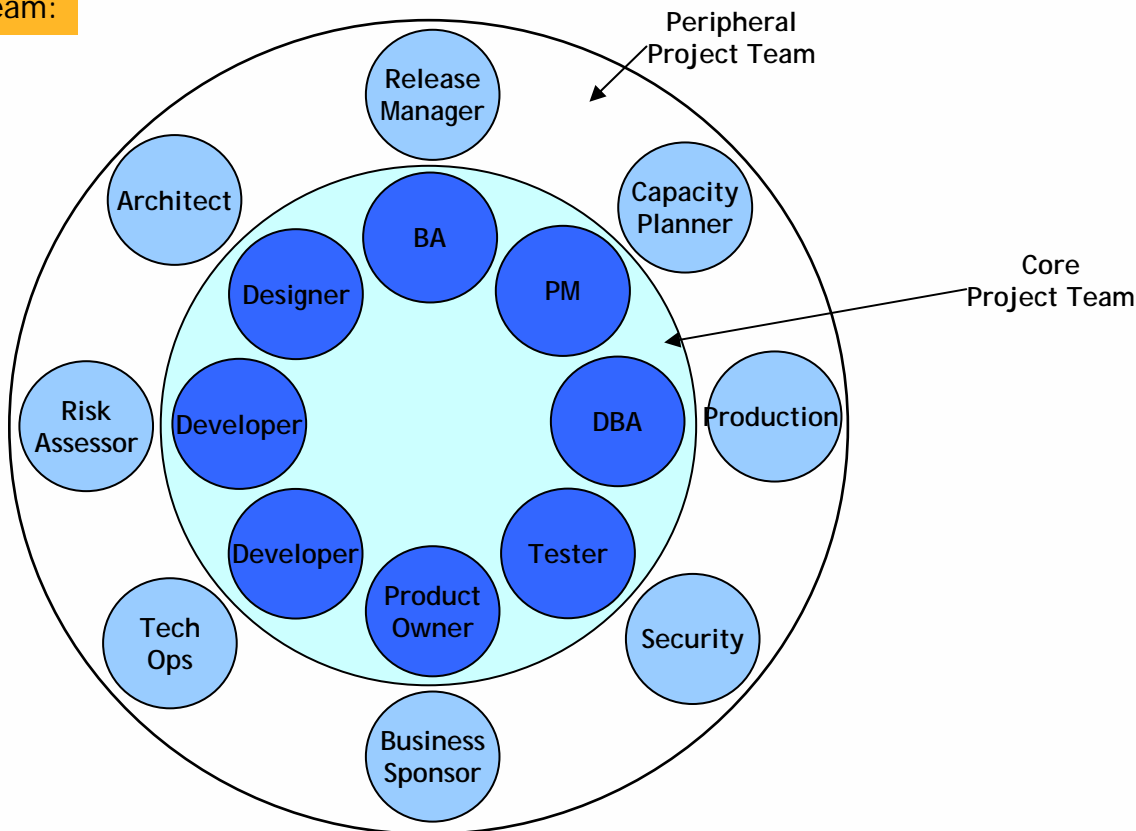


Graphic Courtesy of Mike Cohn

Traditional Silos:



Integrated Agile Team:





1. **Green** acceptance test cards for every **blue** user story
2. Acceptance test cards are created during iteration planning, before development begins
3. Acceptance tests are valuable inputs into the estimation process
4. A user story is not complete until the acceptance tests have passed

- Tests are expressed as tables of input data and expected output data.
- Often used at the “business rules level” but can be used for other forms of testing.
- Scaffolding is written into application code to execute the tests for each input and compare the results to the expected output.
- Wiki interface. The tester specifies the tests in an HTML table. Tests are executed for each row. Color coded results are returned to the screen.

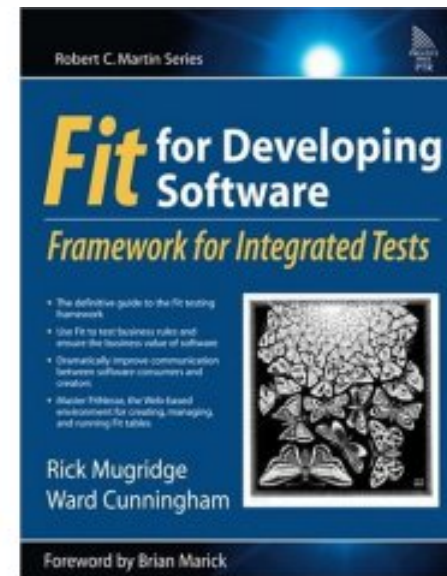
Input 1	Input 2	Result
20	0	0
4	6	24
-5	-5	25
-9	-9	-81

- Discuss acceptance criteria up front during requirements phase
- Capture acceptance criteria as first-class requirements if possible
- Discuss test data and scenarios as early as possible
- Write unit tests that test acceptance criteria
- Have regular integrated team meetings throughout the waterfall process
 - Customer (or proxy)
 - BA, BSA
 - Developers
 - Testers

Validate assumptions!

Can we use acceptance test cases as the requirements document?

- Test-Driven Development by Example by Kent Beck
- Test-Driven Development, A Practical Guide by David Astels
- An Initial Investigation of Test Driven Development in Industry by Bobby George and Laurie Williams
- www.fitnessse.org
- Fit for Developing Software by Rick Mugridge and Ward Cunningham



CC Pace

4100 Monument Corner Drive

Suite 400

Fairfax, Virginia 22030

www.ccpace.com

703/631.6600

