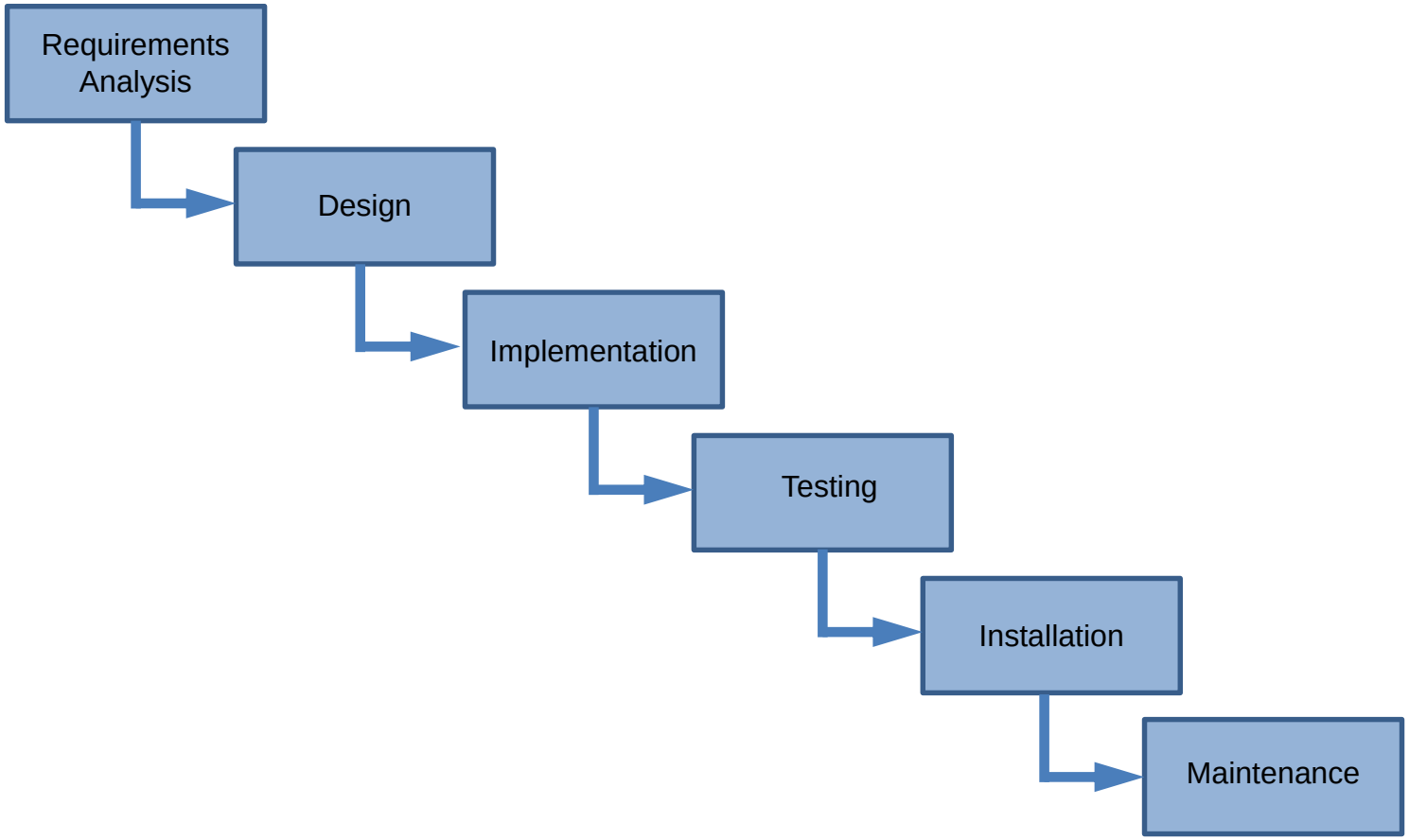
UNIVERSITY OF LINCOLN

Agile Software Engineering

Dr John C Murray

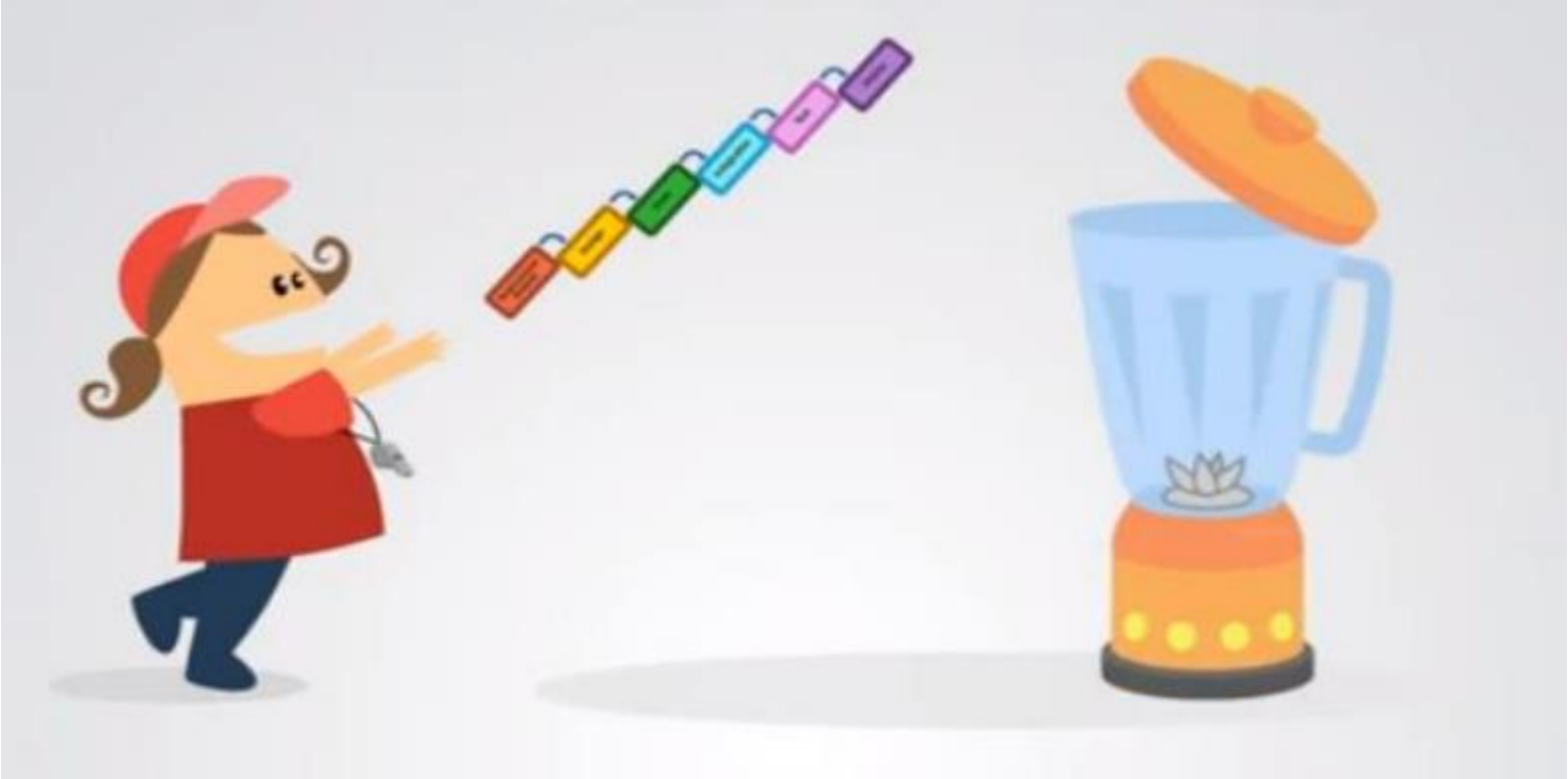
14th October 2015

Waterfall


Requirements Analysis
Design
Implementation
Testing
Installation
Maintenance

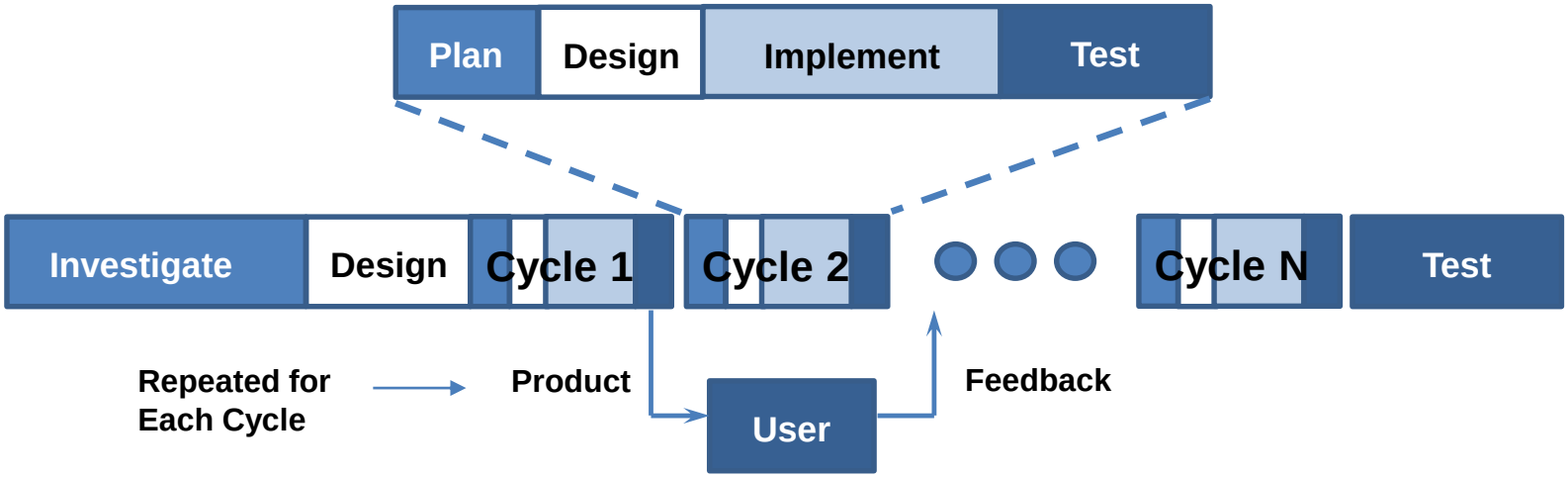
Requirements

- Due to changes in Soft. Eng. it is often impossible to arrive at a **stable**, **consistent** set of requirements.



Evolutionary development

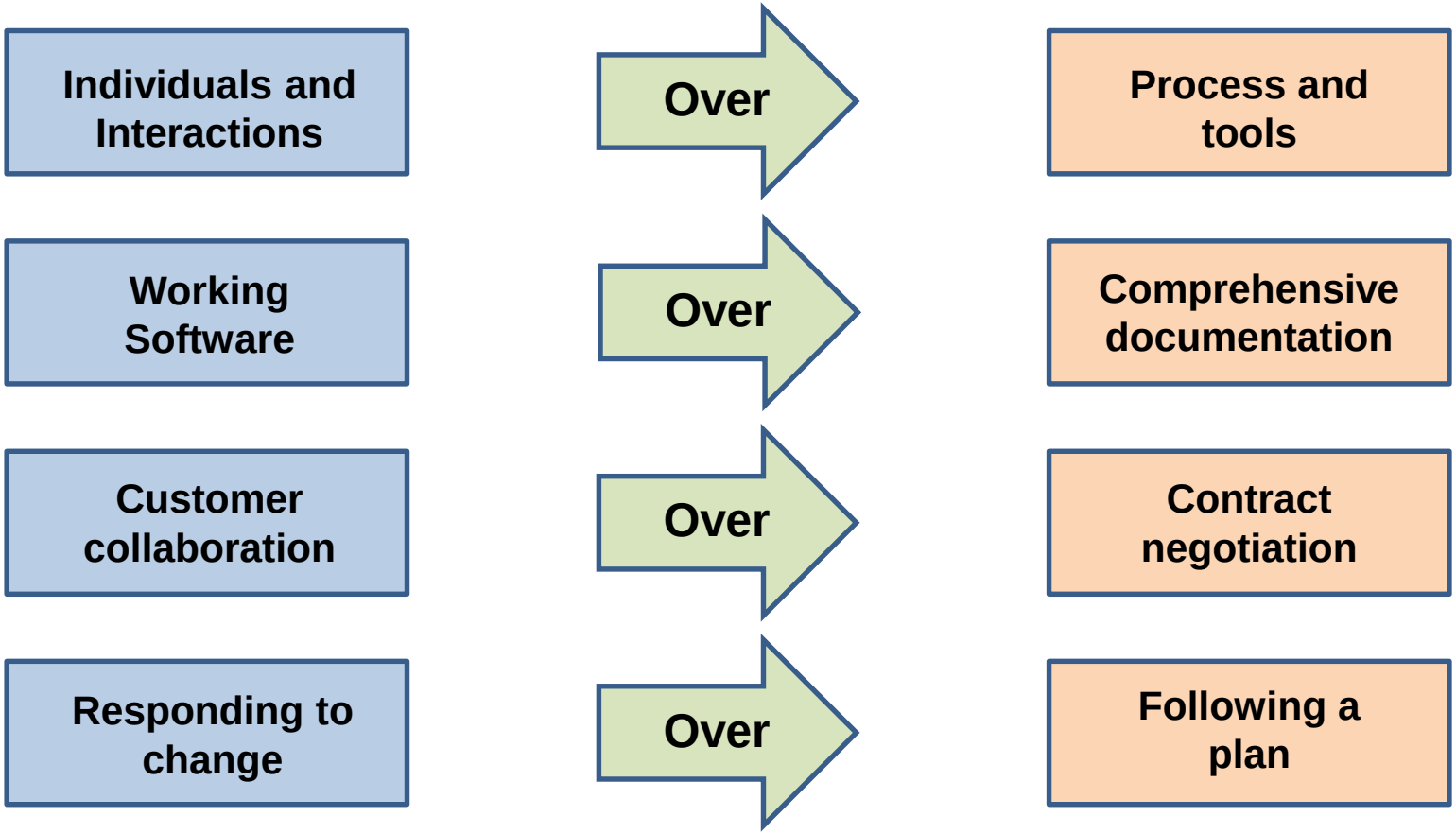
- An iterative approach to specification and delivery in order to deliver software quickly.



Agile Software Development

- What is Agile Development?
  - An incremental/iterative approach to software **specification** and **development**
  - Support business where systems requirements change rapidly
  - A collection of Software Methodologies and processes
    - XP Programming (Beck, 1999), SCRUM (2001), DSDM, Adaptive Software Development (2000)

Agile Manifesto



- http://agilemanifesto.org/principles.html

Principles of Agile Methodology

- Customer Involvement
- Incremental Delivery
- People not process
- Embrace change
- Maintain simplicity

Agile

- Advantages
  - Fast Delivery (Iterations)
  - Allows for lots of client feedback
  - Do not tackle the **whole** problem right away
- Disadvantages
  - Lots of time with clients (and not developing)
  - No long term planning (what's happening in 6 months?)
  - Prioritising changes can be difficult
  - Contract!

Extreme Programming (XP)

- An ‘agile’ process
  - The focus is on code rather than design or documentation
  - XP takes an ‘extreme’ approach to iterative development
    - New versions can be built several times per day
    - Increments are delivered to customers every 2 weeks
    - All tests must be run for every build
      - Build is only accepted if tests run successfully

Extreme Programming (XP)

- In XP, requirements are expressed as ‘stories’
- These are written on cards
  - Dev team break into tasks
  - Tasks are the basis of schedule and cost estimates
- Customer chooses the stories for inclusion in the next release based on priorities and schedule estimates

User Stories

- Serve the same purpose as Use Cases
  - But are not the same
  - Used to create time estimates for release planning
- Used instead of a large requirements document
- Written by the **customer**!!
  - Things the system needs to do for them
  - They are written in about 3 sentences of text

User Stories

- Should only provide **enough** detail to make a low risk estimate of how long it will take to implement
  - Different from Requirements Specifications
    - These need enough detail to **implement**

- When it is time to implement
  - Developers go back to customer for a **detailed** description of requirements

User Stories

- Example User Stories:-
  - Students can purchase monthly parking passes online.
  - Parking passes can be paid via credit cards.
  - Parking passes can be paid via PayPal.

  - Lecturers can input student marks.
  - Students can obtain their current lecture schedule.
  - Students can order official transcripts.
  - Students can only enrol in seminars for which they have prerequisites.

User Stories

1. Stakeholders write them:-
   - Simple and straightforward and the domain expect should write them
2. Use the simplest tool:-
   - Write them on index cards
3. Indicate the estimated size:-
   - Estimate the effort or team size that is needed to implement
4. Indicate the Priority:-
   - What is the priority of the story?
5. Use a unique identifier:-
   - Allows for traceability

173. Students can purchase parking passes.

Priority: 8
Estimate: 4

173

As a student I want to purchase a parking pass so that I can drive to school

Priority : Should
Estimate: 4

Detailing a User Story

Front of Card

173

As a student I want to purchase a parking pass so that I can drive to school

Priority: Should
Estimate: 4

Back of Card

Confirmations:

~~The student must pay the correct amount~~

One pass for one month is issued at a time

The student will not recieve a pass if the payment isn't sufficient

The person buying the pass must be a currently enrolled student.

The student may only buy one pass per month.

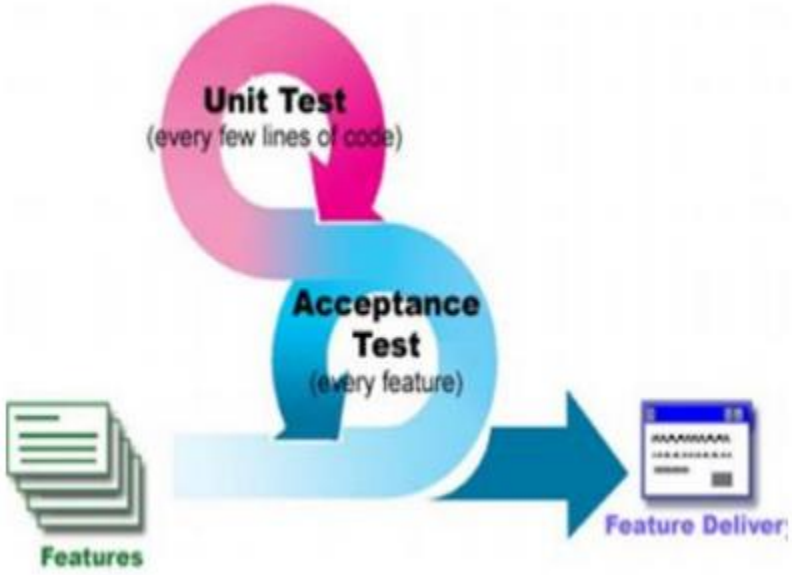
XP Customer

- Part of the team
  - On site, available at all times
  - XP Principles: communication and feedback
  - Making sure we build what the customer wants

- Actively involved in **all** stages
  - Clarify the requirements
  - Negotiate with the team, what to do next
  - Define acceptance tests
  - Constantly evaluates intermediate versions

Testing in XP

- Test-Driven Development (TDD)
  - Write tests **before** code!
  - Tests are automated
  - Must run at 100%



- Acceptance Testing
  - Written with the customer
  - Acts as a ‘contract’
  - Measure of progress

- Unit Testing
  - Automate testing of functionality as developers create it

Test-Driven Development

- Test first
  - Before we write any code, write a test for that feature
- Automated
  - Tests are ran automatically each time a release is built
- Unit Tests
  - Automate testing of functionality as developers write it
- Acceptance Tests
  - Specified by the customer to test the overall system

Pair Programming

- Two programmers work side-by-side
  - at one computer.
- Helps develop common ownership of code
  - Spreads knowledge across the team.
- Continuously collaborate on same design, algorithm, code, test, etc.
- It serves as an informal review process
  - each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.

NOT Pair Programming

https://www.youtube.com/watch?v=msX4oAXpvUE

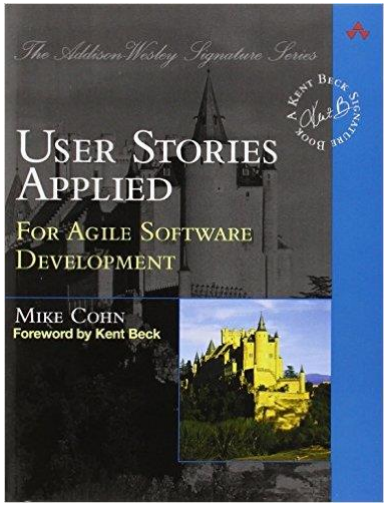
Pair Programming

Observer(Navigator)

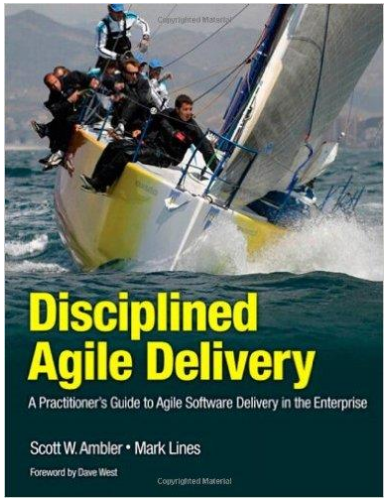
Driver

Books

- User Stories Applied: For Agile Software Development 1st Ed.
  - ISBN-13: 978-0321205681



- Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery
  - ISBN-13: 978-0132810135



Questions

?