COM3005J: Agile Processes
Chapter 1 : **Context**
Agile Principles

Dr. Anca Jurcut
E-mail: anca.jurcut@ucd.ie

School of Computer Science and Informatics
University College Dublin

Beijing-Dublin International College

Agile Principles

1: The enemy: Big Upfront Anything

2: Organizational principles

3: More organizational principles

4: Technical principles

5: A few method-specific principles

Agile Principles

Organizational

- **1** Put the customer at the center
- **2** Accept change
- **3** Let the team self-organize
- **4** Maintain a sustainable pace
- **5** Produce minimal software:
  - 5.1 Produce minimal functionality
  - 5.2 Produce only the product requested
  - 5.3 Develop only code and tests

Technical

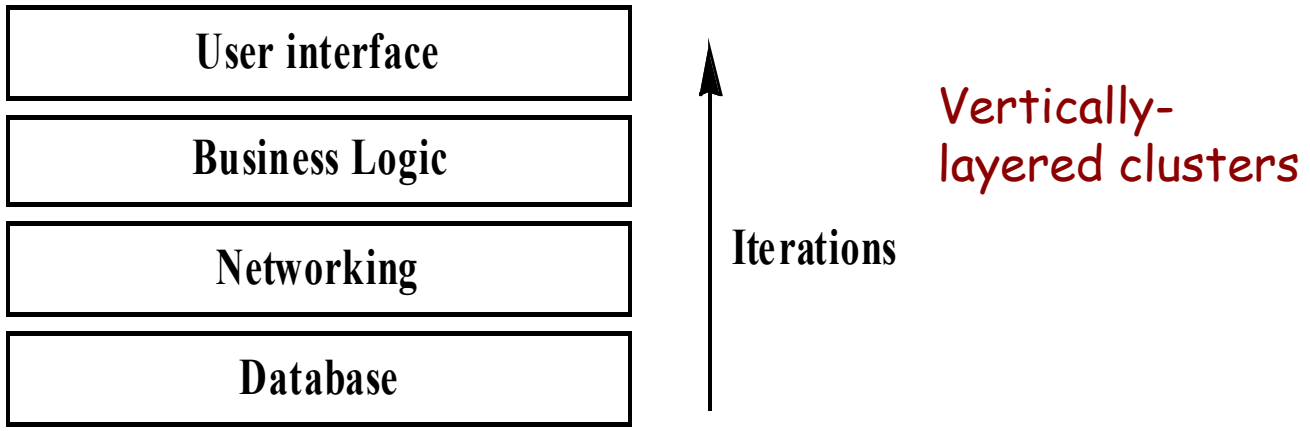
- **6** Develop iteratively
  - 6.1 Produce frequent working iterations
  - 7.2 Freeze requirements during iterations
- **7** Treat tests as a key resource:
  - 7.1 Do not start any new development until all tests pass
  - 7.2 Test first
- **8** Express requirements through scenarios

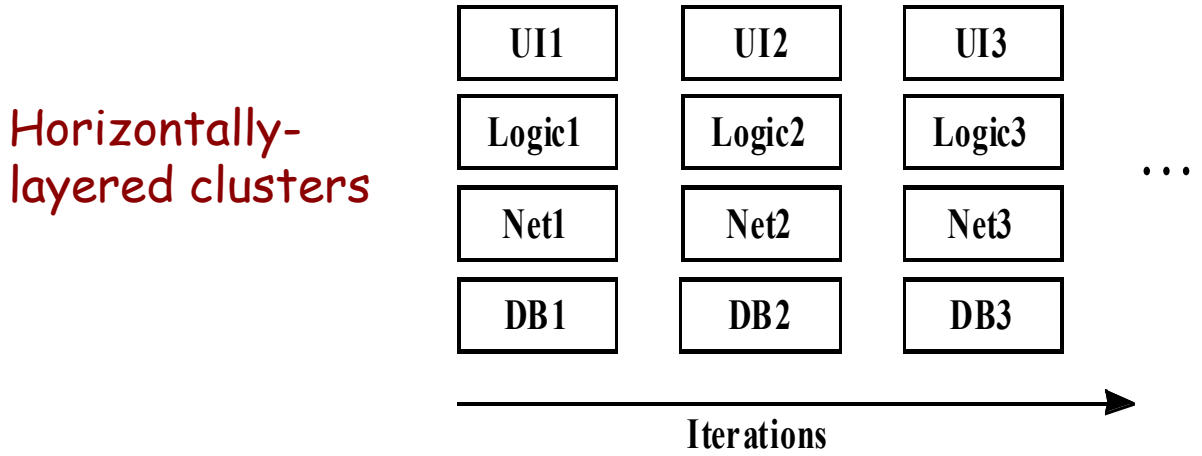
6 Develop Iteratively

Iterativeness:

- 6.1 Frequent working iterations
- 6.2 Freeze requirements during iteration (Closed-Window Rule)

6.1 Iterativeness: frequent working iterations





6.2 Iterativeness: freeze requirements during iteration

The Closed-Window Rule:

During an iteration, no one may add functionality

(or: the sprint is cancelled)

Scrum

7.1 Tests: do not move on until all tests pass

Excellent principle, to be reconciled with practical constraints

Need to classify severity

7.2 Test first

See discussion of practices

8 Express requirements through scenarios

User stories

User story

XP

Scrum

Source: Cohn

"A user story is simply something a user wants"

"Stories are more than just text written on an index card but for our purposes here, just think of user story as a bit of text saying something like

- Paginate the monthly sales report
- Change tax calculations on invoices.

Many teams have learned the benefits of writing user stories in the form of "As a … I … so that …"

Standard form for user stories

Scrum

"As a <*user_or_role*>
I want <*business_functionality*>
so that <*business_justification*>"

Example:

"***As a*** *customer,*
I want to *see a list of my recent orders,*
so that *I can track my purchases with a company."*

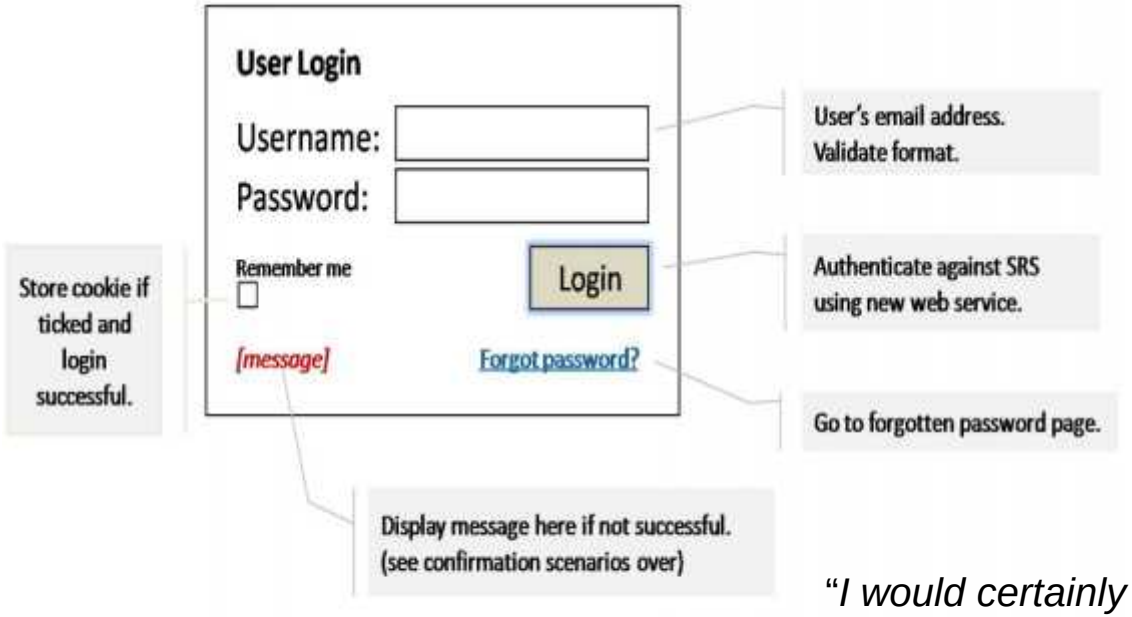
Example user story

Source: Waters

#0001 **USER LOGIN** Fibonacci Size # 3

As a **[registered user]**, I want to **[log in]**, so I can **[access subscriber content]**.

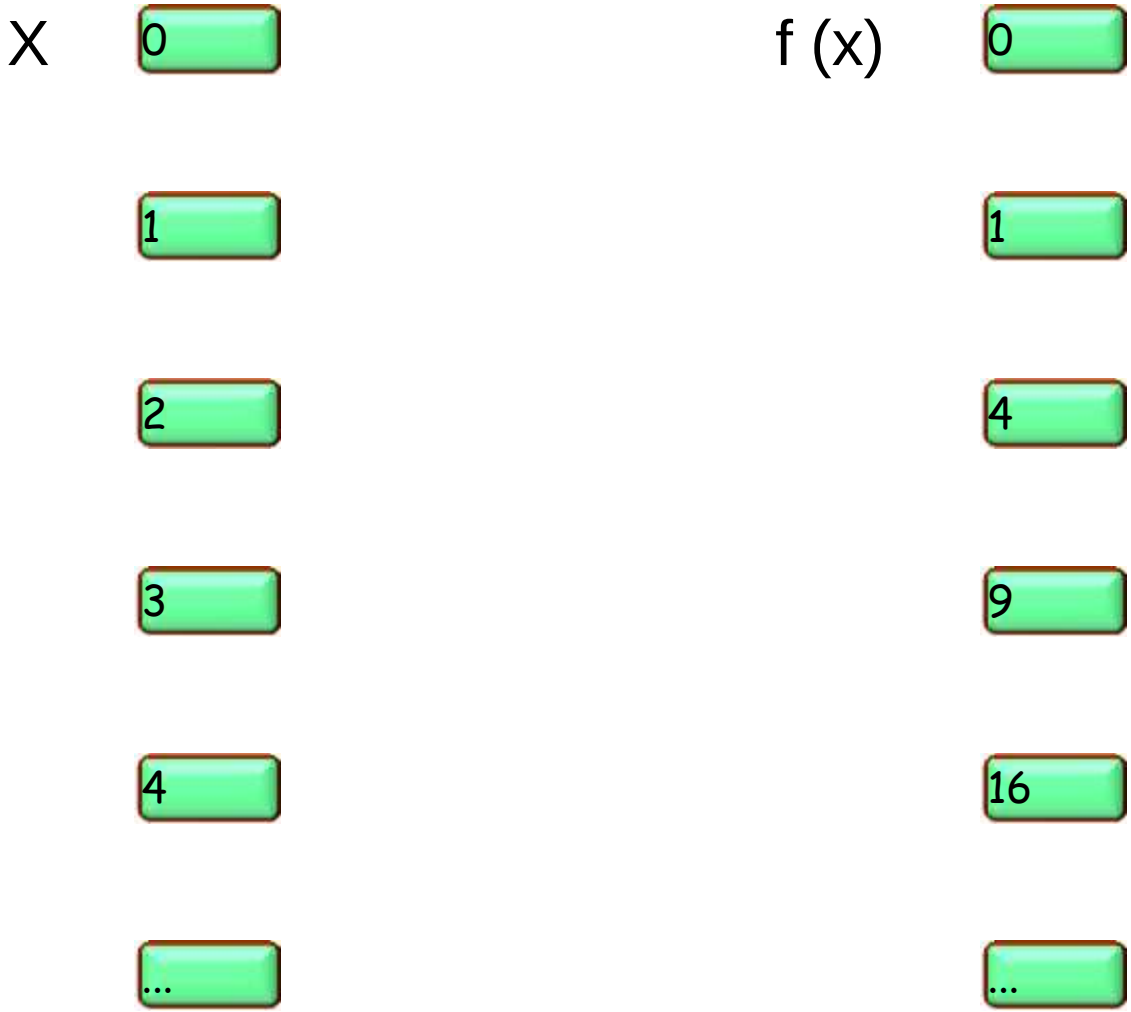
For new features, annotated wireframe. For bugs, steps to reproduce with screenshot. For non-functional stories, explain scope/standards.



Further information is attached to this story on VSTS Product Backlog.

"*I would certainly argue it is more easily digestible than a lengthy specification, especially for business colleagues*"

User stories



See https://bertrandmeyer.com/2012/10/14/a-fundamental-duality-of-software-engineering/

or http://tinyurl.com/qzyxlal

Analysis

User stories help requirement elicitation but not a fundamental requirement technique. They cannot define the requirements:

- Not abstract enough
- Too specific
- Describe current processes
- Do not support evolution

User stories are to requirements what tests are to software specification and design

Major application: for validating requirements

Principles

1. ..
2. ..
3. ..
4. **Technical Principles**

What we have seen:

Iterative development

The fundamental role of tests (more to come)

User stories as the source of requirements…

… and the limits of this approach

Agile Principles

1: The enemy: Big Upfront Anything

2: Organizational principles

3: More organizational principles

4: Technical principles

5: A few method-specific principles

Agile principles

Organizational

- **1** Put the customer at the center
- **2** Accept change
- **3** Let the team self-organize
- **4** Maintain a sustainable pace
- **5** Produce minimal software:
  - 5.1 Produce minimal functionality
  - 5.2 Produce only the product requested
  - 5.3 Develop only code and tests

Technical

- **6** Develop iteratively
  - 6.1 Produce frequent working iterations
  - 7.2 Freeze requirements during iterations
- **7** Treat tests as a key resource:
  - 7.1 Do not start any new development until all tests pass
  - 7.2 Test first
- **8** Express requirements through scenarios

Eliminate waste

Lean

Source: Poppendieck

Everything not adding value to the customer is considered waste:

- Unnecessary code
- Unnecessary functionality
- Delay in process
- Unclear requirements
- Insufficient testing
- Avoidable process repetition
- Bureaucracy
- Slow internal communication
- Partially done coding
- Waiting for other activities, team, processes
- Defects, lower quality
- Managerial overhead

The Seven Wastes of Manufacturing
Overproduction
Inventory
Extra Processing Steps
Motion
Defects
Waiting
Transportation

Value stream mapping: strategy to recognize waste. Eliminate it iteratively

Amplify learning

Lean

Source: Poppendieck

Software development is a continuous learning process
The best approach for improving a software development environment is to amplify and speed up learning:

- To prevent accumulation of defects, run tests as soon as the code is written
- Instead of adding documentation or planning, try different ideas by writing and testing code and building
- Present screens to end-users and get their input
- Enforce short iteration cycles, each including refactoring and integration testing
- Set up feedback sessions with customers

Decide as late as possible

Lean

Source: Poppendieck

Delay decisions as much as possible until they can be made based on facts, not assumptions, and customers better understand their needs

The more complex a system, the more capacity for change should be built in

Use iterative approach to adapt to changes and correct mistakes, which might be very costly if discovered after system release

Planning remains, but concentrates on the different options and adapting to the current situation, as well as clarifying confusing situations by establishing patterns for rapid action

Focus

Focus on individual task, to ensure progress:

- Control flow of progress
- Deal with interruptions:
  - Two-hour period without interruption
  - Assign developer to project for at least two days before switching

Focus on direction of project

- Define goals clearly
- Prioritize goals

Deliver as fast as possible

Lean Scrum

Source: Poppendieck

It is not the biggest that survives, but the fastest

The sooner the end product is delivered, the sooner feedback can be received, and incorporated into the next iteration

For software, the Just-in-Time production ideology means presenting the needed result and letting the team organize itself to obtain it in a specific iteration

Multiple design

Lean

Source: Poppendieck

Another key idea from Toyota is set-based design. If a new brake system is needed, three teams may design solutions to the problem

If a solution is deemed unreasonable, it is cut

At period end, the surviving designs are compared and one chosen, perhaps with modifications based on learning from the others — an example of deferring commitment until the last possible moment

Software decisions could also benefit from this practice to minimize the risk brought on by big up-front design

Minimize dependencies

Scrum asserts that it is possible to remove dependencies between user stories, so that at any point any user story can be selected according to the proper criteria (maximizing business value)

Additive and multiplicative complexity

Adding features

Source: Zave*

In telecommunication software, feature interactions are a notorious source of runaway complexity, bugs, cost and schedule overruns, and unfortunate user experiences.

Bob has "call-forwarding" enabled and is forwarding calls to Carol. Carol has "do-not-disturb". Alice calls Bob, the call is forwarded to Carol, and Carol's phone rings, because "do-not-disturb" is not applied to a forwarded call.

Alice calls a sales group. A feature for the sales group selects Bob as a sales representative on duty, and forwards the call to Bob. Bob's cellphone is turned off, so his personal Voice Mail answers the call and offers to take a message.

[etc.]

*Abridged, see full citation in book

User stories (imagined)

(#1) **As an** executive, **I want** a redirection option **so that** if my phone is busy the call is redirected to my secretary

…

(#2) As a system configurator, I want to be able to specify various priorities for “busy” actions

..

(#3) As a salesperson, I want to make sure that if a prospect calls while I am in a conversation, the conversation is interrupted **so that** I can take the call immediately

…

(#4) As a considerate correspondent, I want to make sure that if a call comes while my phone is busy I get to the option of calling back as soon as the current call is over

Personal safety

- Encourage free expression of ideas
- Do not ridicule anyone because of a question or suggestion
- Build trust within the team

Humanity

Source: Beck 05

Recognize that software is developed by people

Offer developers what they expect:

- Safety
- Accomplishment
- Belonging
- Growth
- Intimacy

Agile approaches are indebted here to DeMarco's and Lister's *Peopleware* (see bibliography)

Context: Principles

1. ..
2. ..
3. ..
4. ..
5. **A few method-specific principles**

What we have seen:

Important ideas from Scrum, Lean, Crystal and XP

Minimize waste

Accumulate user stories (and the limits of that approach)

Amplify learning

Ensure personal safety

(and a few others)