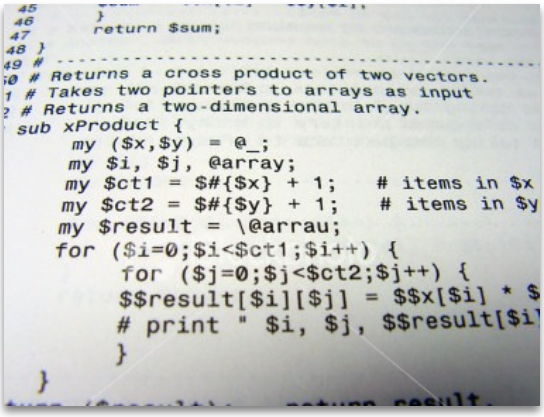
UiO : University of Oslo

Agile Software Development:
what can we learn as researchers?

Pierre Lison,
Language Technology Group (LTG)
Department of Informatics

LTG Tech Lunch
October 31 2012

Introduction



```
      }
      return $sum;
}
# ---------------------------------------------------
# Returns a cross product of two vectors.
# Takes two pointers to arrays as input
# Returns a two-dimensional array.
sub xProduct {
    my ($x,$y) = @_;
    my $i, $j, @array;
    my $ct1 = $#{$x} + 1;    # items in $x
    my $ct2 = $#{$y} + 1;    # items in $y
    my $result = \@arrau;
    for ($i=0;$i<$ct1;$i++) {
        for ($j=0;$j<$ct2;$j++) {
        $$result[$i][$j] = $$x[$i] * $
        # print " $i, $j, $$result[$i
        }
    }
```

As NLP researchers, we spend a lot of our time dealing with code

(Reading/designing/writing/debugging/testing, etc.)

Introduction

- In many ways, the *quality of our code* has a decisive impact on the *quality of our research*

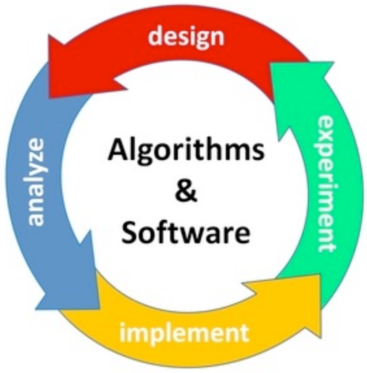
1. **Good code** → (often) better empirical results
2. **Good code** often helps us get a better understanding of our research problem (concepts, limitations, etc.)
3. **Good code** is easier to extend, reuse and refactor in several experiments or projects
4. **Good code** makes it easier for other researchers to understand our work, and adopt it in their own research

Introduction

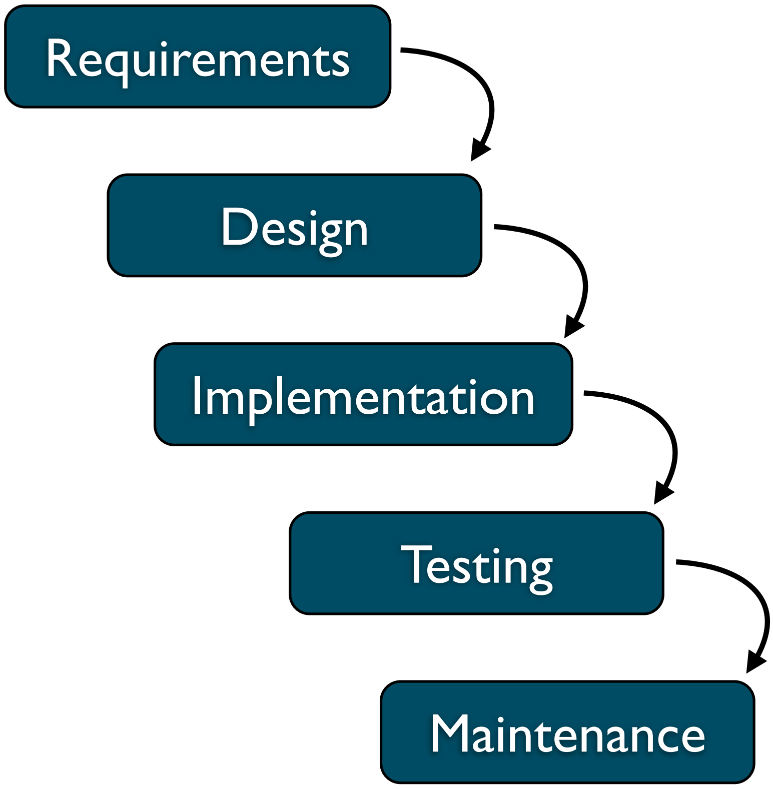
- ... but strangely enough, we (researchers) rarely *reflect* on the adequacy of our development methods
    - Are our development methods optimal?
    - Do we focus on the right (=high priority) aspects?
    - Do we control the *quality* of our code?
    - How do we deal with unexpected events (e.g. unforeseen problems, change in approach)?
    - Do our methods promote or hinder *collaboration*?

Introduction

- Software engineering changed a lot in the last 10 years
- *Agile* development methods increasingly popular
- I would like to talk about some of these new ideas
  - And most importantly, how they can help us do better research



Waterfall model


Requirements

Design

Implementation

Testing

Maintenance

Traditional way of building software

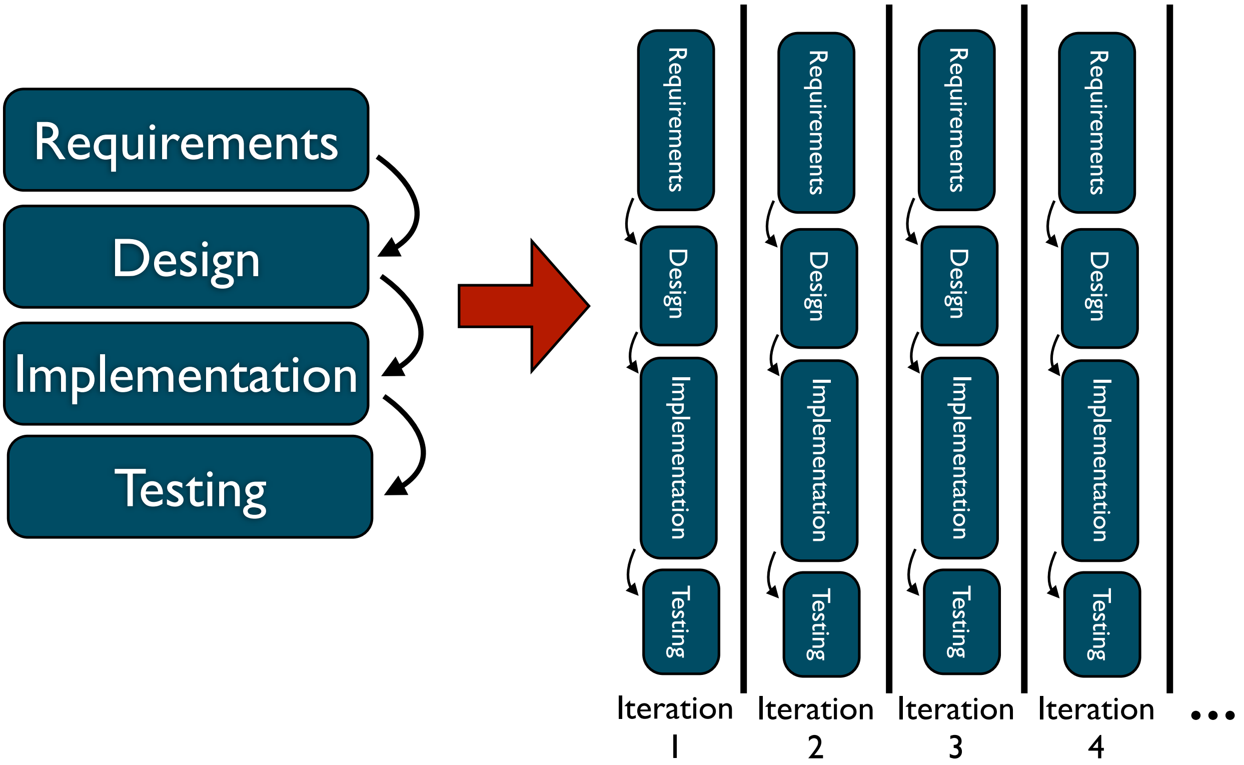
«Big Design Up Front»

Problems with the waterfall

- Drawbacks of the waterfall model:
  - The software requirements often vague & volatile
  - Many design issues only become apparent at implementation time
  - Working software only available at the latest stages
  - Inability to adapt to unforeseen events
  - Typically leads to rigid division of labour

Alternative: develop software in a more *incremental* & *iterative* fashion

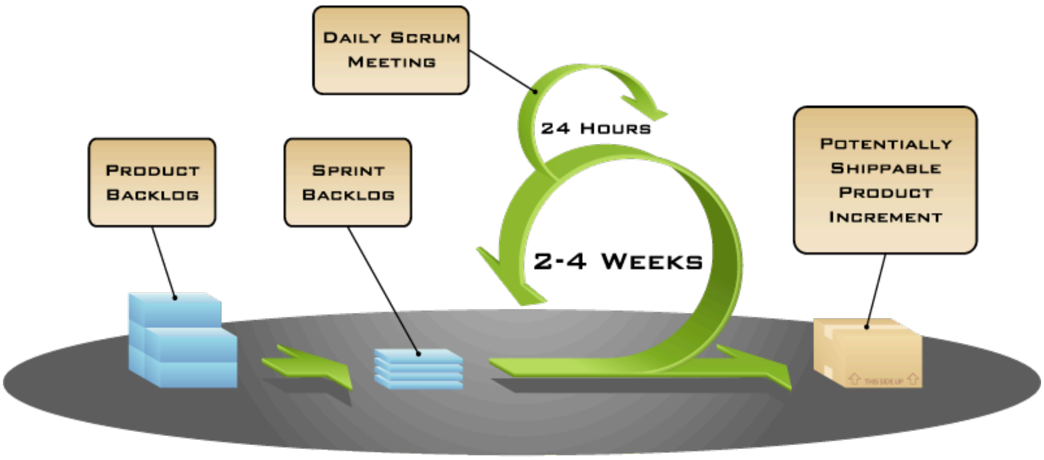
Incremental development



Incremental development

- **Idea**: construct the software via a sequence of several *short* iterations
    - Iteration purpose is to integrate a new *functionality* (the one with the highest priority at the moment)
    - Each iteration includes some basic requirements analysis, design, implementation and testing
    - At the end of each iteration, we have a *working system*, extended with the given function

Incremental development



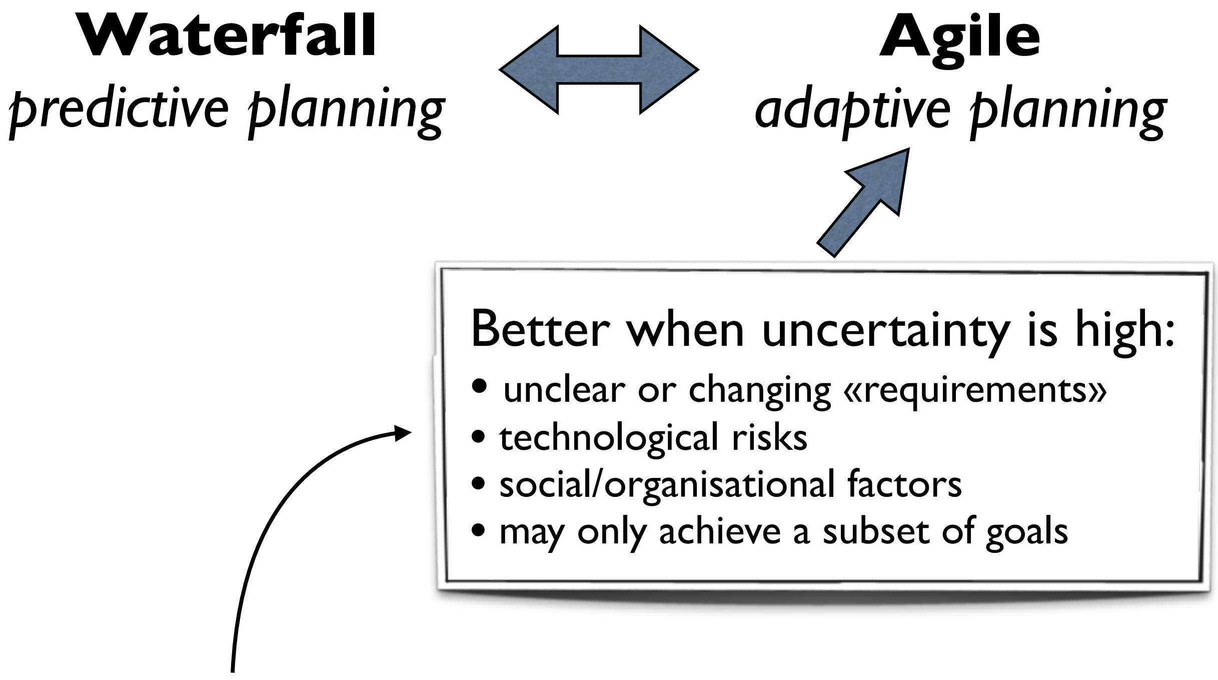
Copyright © 2005, Mountain Goat Software

One particular type of Agile method: *Scrum*

Incremental development

- Advantages of incremental & iterative development methods:
  - Fast delivery of a *working system*, even though it may be imperfect or incomplete
  - *Gradual refinement and extension* of the software requirements and system design
  - Greater *adaptivity* to unforeseen changes (implementation problems, external events)

Incremental development



Waterfall

predictive planning

Agile

adaptive planning

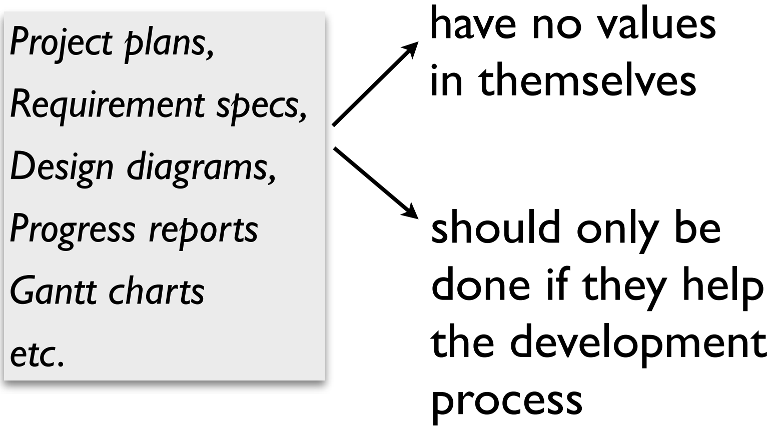
Better when uncertainty is high:

- unclear or changing «requirements»
- technological risks
- social/organisational factors
- may only achieve a subset of goals

High-uncertainty is the *norm* in academic research

Agile: a lightweight methodology

Agile put the emphasis on *working software* as the core development objective



Project plans,
Requirement specs,
Design diagrams,
Progress reports
Gantt charts
etc.

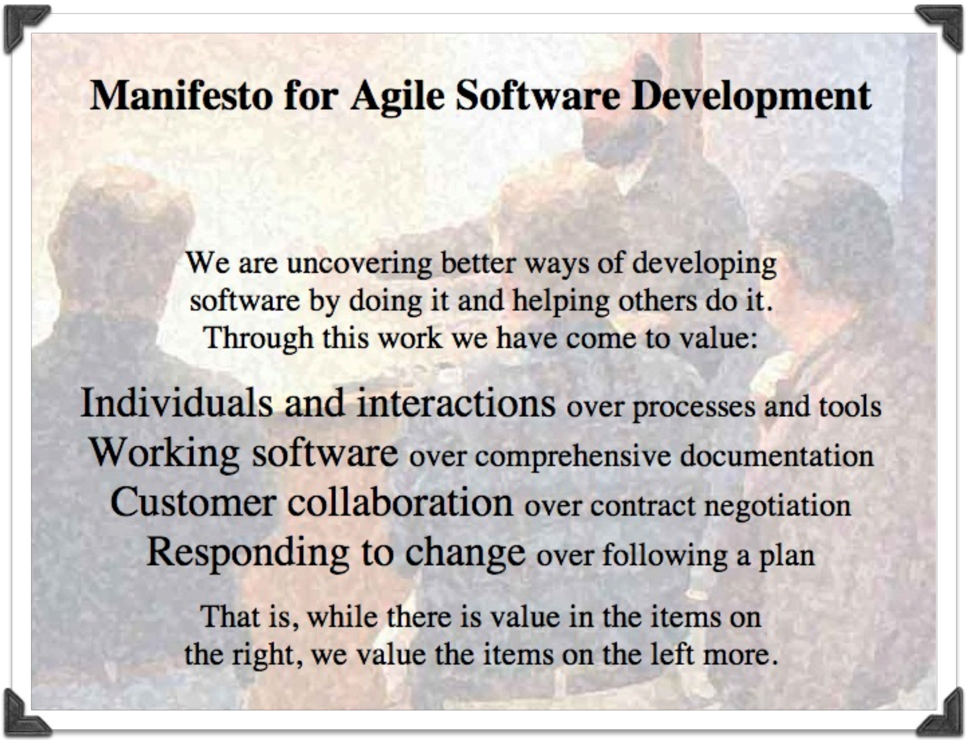
have no values in themselves

should only be done if they help the development process

Agile: a lightweight methodology

- Organisational structure is also lightweight
  - No rigid roles or hierarchy, work is largely *self-directed*
  - Users seen as *partners* directly engaged in the development process
  - Emphasis on direct, face to face *collaboration*

Agile manifesto



Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

[The Agile Manifesto: http://agilemanifesto.org/]

Agile methodologies

Scrum

Crystal

Agile Unified Process

Lean

Extreme Programming (XP)

Good practices

Four agile engineering practices:

Unit testing Write systematic test cases for every unit of code to ensure the requirements are satisfied. Run the unit tests after any code change.	**Refactoring** Modify the code's internal structure (without altering its behaviour) to follow standard patterns, increase readability and extensibility
Test-driven development Write test cases *first*, as a way to define the software requirements. Then use these tests to control the development progress	**Continuous Integration** Commit written code to repository and rebuild system as soon as possible. Automatically control for integration problems.

Conclusion

- *Agile* development methodologies can help us write better code
  - Improved quality, faster delivery, increased flexibility
  - Especially useful for research systems, which must typically face *high uncertainty*
  - Lightweight, but highly disciplined methodology!

Conclusion

- Key ideas
  - Development as a sequence of short *iterations* gradually extending or improving the system
  - *Working code* is the primary focus, not procedures or hierarchical roles
  - Adaptivity: *Embrace* change instead of trying to predict it