Good?/Best? programming practices

Based on a lecture from software-carpentry.com

Anna Simon

September 4, 2015

Intro

- Many scientists write code regularly but few have formally been trained to do so
- Best practices can make a lot of difference
- Development methodologies are established in the software engineering industry
- We can learn a lot from them to improve our coding skills

Common Scenarios:

- Lone student/scientist
- Small team of scientists, working on a common library. Speed of development more important than execution speed
- Often need to try out different ideas quickly:
  - rapid prototyping of a proposed algorithm
  - re-use/modify existing code

Development methodology

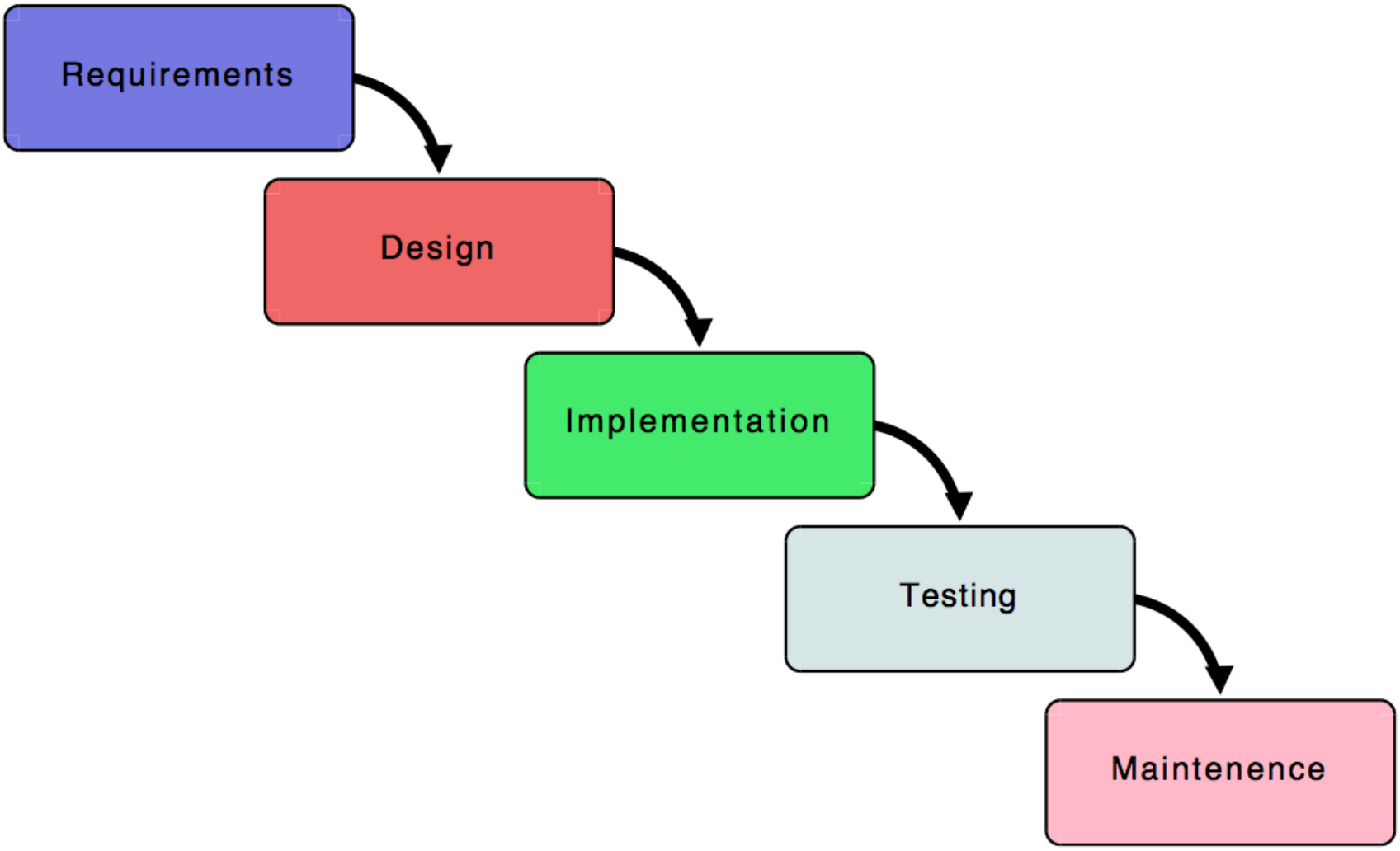
Consists of:

- A philosophy that governs the style and approach towards development
- A set of tools and models to support the particular approach

Helps answer the following questions:

- How far ahead should I plan?
- What should I prioritize?
- When do I write tests and documentation?

Waterfall method



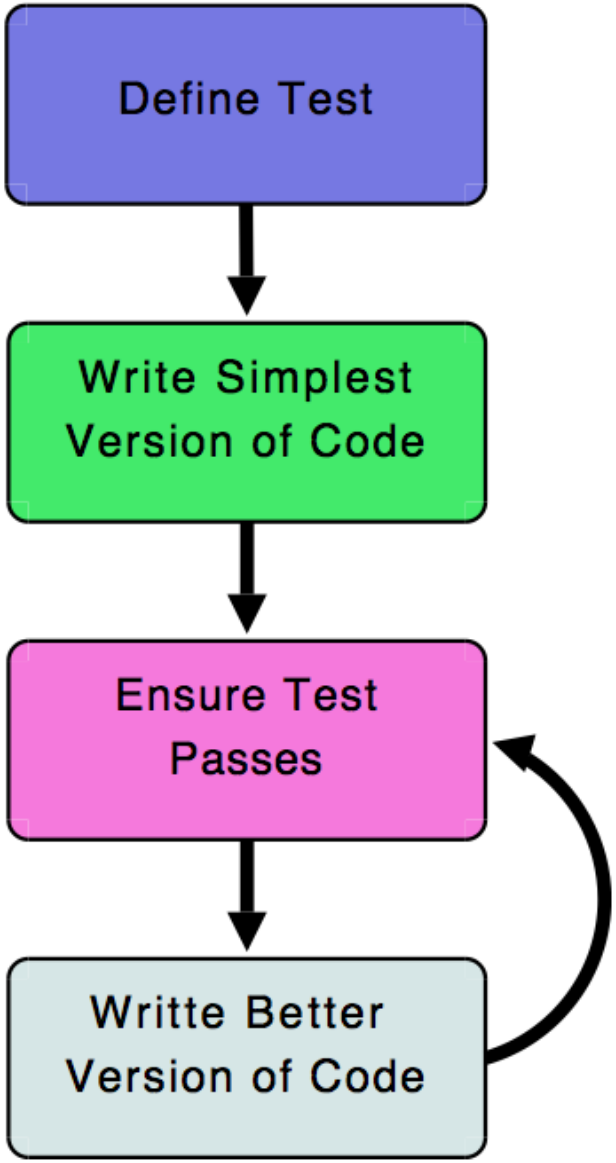
Agile method

- Agile methods emerged during the late 90's
- Generic name for set of more specific paradigms
- Set of best practices
- Particularly suited for:
  - small teams (less than 10 people)
  - unpredictable or rapidly changing requirements

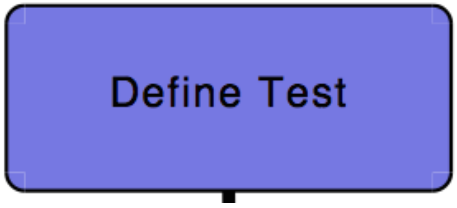
Prominent Features of Agile methods

- Minimal planning
- Small development iterations
- Rely heavily on testing
- Promote collaboration and teamwork
- Very adaptive

Agile workflow

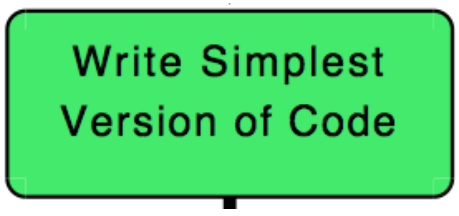


Agile workflow



The function `my_sum` has to return the sum of all the elements

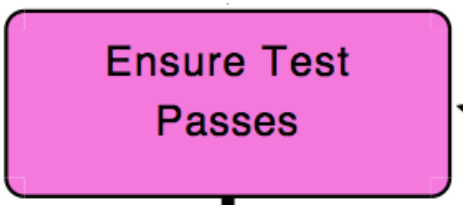
Agile workflow



Write a simple version of the function:

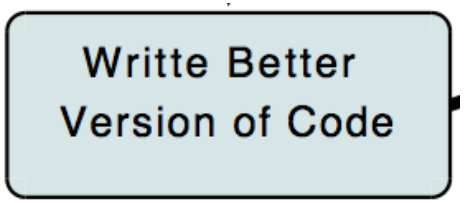
```
1 int my_sum(vector<int> list}){
2 //computes the sum of all the elements of the list
3     int sum_of_elem;
4     for(vector<int>::iterator iter=list.begin(); iter!=list.end();
          ++iter)
5           sum_of_elems += *iter;
6          return sum_of_elem;
7 }
```

Agile workflow




```
1 vector<int> test_vector(3,1);
2 if(my_sum(test_vector) != 3)
3     cout << "There is something wrong with my_sum function" << end;
```

Agile workflow




```
1 int my_sum(vector<int> list}){
2     int sum_of_elems =std::accumulate(vector.begin(),vector.end(),0)
          ;//#include <numeric>
3     return sum_of_elems;
4 }
```

Unit testing

- Unit is the smallest testable piece of code
- e.g. my_sum function

Goals of unit tests

- check code works
- catch regression (new features that break old part of the code that used to work)

Why to test?

- Easier to test the whole, if the units work
- Can modify parts, and be sure the rest still works
- Provide examples of how to use code

How to test?

- Test with simple cases, using hard coded solutions
  `my_sum([1,2,3]) == 6`
- Test special or boundary cases
  `my_sum([]) == 0`
- Test that meaningful error messages are raised upon corrupt input
  `my_sum(['1', 'a'])` $\rightarrow$ incorrect data type

What makes a good test:

- independent (of each other, and of user input)
- repeatable (i.e. deterministic)
- self-contained

Refactoring

This is what it is called when you write a better version of your code.

- Re-organisation of your code without changing its function:
  - remove duplicates by creating functions and methods
  - increase modularity by breaking large code blocks into units
  - rename and restructure code to increase readability and reveal intention
- Always refactor one step at a time, and use the unit tests to check the code still works

Introducing new features

- Split feature into units
- Use the agile workflow
- Tests drive the development. Keep the iterations small

Documenting the code

- Comment to communicate what the code should be doing. Comments should explain the "why" of the code and stop there.
- Do not write comments that explain what is going on. It basically means translating C++ to English!

```
1 *pointer++;      //increment pointer by one
```

- Write the comments for a unit of code that will be still accurate even if the implementation of that unit changes!
- Your goal should be to write Really Obvious Code that will be understandable to anyone