A discussion of the relevance of the waterfall and agile methodologies to the NEMO software development process

Mike Bell, Claire Levy, Julien le Sommer & Amy Young

Ocotober 2020

1. Introduction

These notes evolved from consideration of two characterisations of the software development process during the review of NEMO's approach to verification and validation. Our aim is to propose a description of the software development process which is consistent with good practice and with what the best NEMO code developers do. This is intended to be neither a revolution nor a burden on developers. Rather it is intended to reconcile different approaches to and aspects of good practice.

2. Overview of standard methodologies and terminology

For any major code system, its development process is designed to ensure that the developments submitted to it have been properly designed, reviewed and verified. It's important that the process is both followed by developers and sufficiently rigorous to avoid introducing errors that can waste a lot of time during integration or remain undetected for a long time. Intelligent use of the process is important to maintain and improve the quality of the system.

There are two well-known methodologies for code development.

The original waterfall methodology (e.g. see https://en.wikipedia.org/wiki/Waterfall_model) can be summarised as involving the following stages:

1. System and software requirements: captured in a product requirements document
2. Analysis: resulting in models, schema, and business rules
3. Design: resulting in the software architecture
4. Coding: the development, proving, and integration of software
5. Testing: the systematic discovery and debugging of defects
6. Operations: the installation, migration, support, and maintenance of complete systems

It is widely recognised that in practice some iteration of the stages is usually needed (e.g. the analysis and design are often found to be flawed and re-visited during the coding stage).

The Agile methodology (e.g. see https://en.wikipedia.org/wiki/Agile_software_development) has twelve principles:

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Deliver working software frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the primary measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams

12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

There are large literatures on these methodologies and a literature of its own comparing them. See for example https://www.guru99.com/waterfall-vs-agile.html.

3. Discussion of the use of waterfall and agile methodologies within NEMO

We consider the waterfall and agile methodologies for code development to be "theoretical" characterisations of alternative views of good practice. We discuss what use is made of similar approaches within NEMO and how we might take elements from each to agree a coherent approach to code development which is as consistent as possible with what the best NEMO developers currently do.

In order to avoid confusion we want to emphasise two points. First we only discuss how these processes are relevant to a code development that is covered by a single ticket. The coordination of priorities for development (e.g. the formulation of the NEMO Development Strategy) is a different activity that is not considered here. Second the way that most teams using Agile methodologies work might be quite different from what is envisaged here. We only consider the principles of Agile methodologies not how they are usually implemented.

It is interesting to compare the two approaches with the learning cycle. One description of the four stages of that cycle is: reflecting, concluding (theorising), planning, doing. At the risk of being simplistic, the original waterfall methodology starts with the theorising and ends with the doing, whilst the agile methodology emphasises learning by doing. Everyone has their preferred learning stage(s) or styles. Effective learning (https://www.simplypsychology.org/learning-kolb.html) normally involves a progression through the stages. The iterative nature of designing and coding reflects the learning cycle of the people developing the code. There are some similarities also with the creative process (http://members.optusnet.com.au/charles57/Creative/Brain/process.htm) as one might expect given that design is a creative process.

https://forge.ipsl.jussieu.fr/nemo/wiki/Developers/DevelopingCodeChanges describes the NEMO development process (NDP). It focuses on steps 3 to 5 of the waterfall methodology (design, coding and testing). The preview step of the design has been included in order to avoid code failing the review process. Review failures are usually due to of a lack of consistency with the NEMO coding rules or the use of an inelegant approach which is likely to make the code harder to maintain in the longer term. The NDP could be seen as a minimal specification of the waterfall methodology. One of the strengths of the waterfall methodology and the NDP is that it has a clear logic and it is relatively straightforward to describe how to follow it.

Principles 5-11 of the agile methodology match fairly well with the typical working practices of small teams collaborating on NEMO development. With a little imagination one can re-write the first four principles so that they work for NEMO too. If we replace "customers" and "business people" in principles 1 and 4 with "scientists" or "System Team members" those principles make good sense for NEMO. We also welcome changing requirements even late in development (principle 2) when the requirements have a solid foundation (e.g. we've just realised we've misunderstood or forgotten something). We also prefer to get something working quickly first as a proof of concept (principle 3).

The last point about preliminary prototyping is something that we do a lot. It's only when you've played around with alternative options that you can really tell which option is going to work best. At that point it is hard not to get on and finish off the job, whilst you have what needs doing in your

head. It is frustrating (and inefficient) to wait for somebody to review the approach that is being taken. It seems that this is the way that most experienced NEMO developers work. Some people (depending on the type of development and their character) might write some outline code designs first; some aspects of the design will be written as comments structuring the code as it is written. A succinct summary of the design is usually written once the code is working. It's rather unreasonable to ask people outside the NST to follow a methodology most of us don't follow ourselves.

Gurvan and many other NEMO developers (like Andrew, Clement, Dave, Italo, Jerome, Mirek, Pierre and Seb) work very well face-to-face in small teams (usually individuals) on specific code developments with people outside the NST (e.g. David Marshall, Wayne Gaudin). These "external experts" bring specific expertise that they share with us. In the jargon these are code sprints. One question is how other members of the NEMO System Team can similarly support people who want to contribute to NEMO.

The obvious approach is to engage with "external experts" at an early stage in the development of code for NEMO through face-to-face meetings (in person or by video) where the basic ideas are discussed and options to implement them are debated. (First meetings usually need to be face-to-face.) This is time-consuming. It is also quite demanding. If the ideas that are being offered are not very good, it can be difficult to transform them into something better or politely decline to pursue them. The System Team members perhaps need some guidance ("training") in how to handle such situations.

An alternative to personal engagement of this sort is to write a better explanation of the NEMO code design principles. This would probably be quite difficult and might be difficult to maintain. It would need a competent and willing volunteer and a lot of discussion.

Several teams have been working this way recently. Some examples of them are:

- variable re-naming for the 2LTS – Dave, Andrew, Gurvan, Mike
- 2LTS – Florian, Jerome, Gurvan, Dave, Andrew, Mike
- Halos, tiling – Italo, Francesca, Daley, Seb, Gurvan, Mike
- GEOMETRIC – David Marshall, Gurvan
- Adaptation of NEMO to GPUs – Wayne Gaudin, Mirek, Andy Porter

There are probably many more (e.g. in IMMERSE WP5, ….)

Our effectiveness in engaging in this way with a wider (but carefully chosen) scientific community will be one of the key factors determining the long term future of NEMO. Just to repeat, this is not a revolution. We're just explicitly recognising an effective way of working in order to encourage its use.

If we agree that this better describes our ways of working, we should modify our description of the NEMO development practice. This should describe typical approaches to engagement, testing, use of branches, documentation and formal review. **There is probably relatively little change required to the documentation**. But it is important that people are not confused by the process they are asked to follow and that they recognise the potential value (and costs) of team-work.

Something else we should bear in mind is that it is the outcomes not the process that really matters. It matters that the code is readable and well organised so that it is secure and flexible; that it does not undermine the NEMO naming standards or its implicit design; that it does what it is intended to do; and that people can understand it relatively quickly and see what tests of it have been done.