The Ultimate Method

Essay by Marc de Bruijn
MA Media Design
Piet Zwart Institute, 2006

Ye olde zealots

01

Computer zealotry is something which happens a lot in the various communication channels of the Internet. There are “editor wars” between proponents of “vi” and “emacs”. People have vicious discussions about the godlike abilities of a company like Apple as opposed to the general and extreme inferiority of Microsoft.

There are similar ongoing debates in the world of programming. From the 1970s several methodologies have been invented to streamline the practice of developing software and programming in particular. “Agile software development”, as well as the “Waterfall model” are examples of such programming paradigms. Each method claims to be the most efficient and productive way to realise software products, while rejecting the core philosophy of the other paradigms.

Agility

02

"Agile software development" is collection of conceptually related methods, each of them more or less propagating the idea of developing software in short iterations, each iteration being a small part of the final piece of software. The software is developed in small teams, consisting not only of programmers, but also managers, designers and technical writers. Agile methods propagate verbal communication, rather than using written text. The term "Agile development" was defined in the "Agile Manifesto" by a group of prominent programmers – like Kent Beck, Ward Cunningham, Jim Highsmith and Ron Jeffries – in 2001. The manifesto was accompanied by another publication called "Principles behind the Agile Manifesto", providing a more detailed view on the concept of "Agile development".

> *"What emerged was the Agile 'Software Development' Manifesto. Representatives from Extreme Programming, SCRUM, DSDM, Adaptive Software Development, Crystal, Feature–Driven Development, Pragmatic Programming, and others sympathetic to the need for an alternative to documentation driven, heavyweight software development processes convened."*
> **("History: The Agile Manifesto", Jim Highsmith, Agile Alliance, 2001)**

The "Agile approach" isn't that new, several other methodologies existed before "Agile development" was coined in 2001, before that the common name for these conceptual systems was "lightweight methods". Older methods include "Adaptive software development", "Feature Driven Development", DSDM (Dynamic Systems Development Method), Crystal Clear and "Extreme programming".

Extreme Programming

"Extreme programming" (XP), created by Kent Beck, Ron Jeffries and Ward Cunningham in 1996, is a widely used technique and certainly popularised the usage of the "Agile methodology". The practices of XP illustrate the mechanics of the "Agile development" methodology quite completely.
The textual outline of what XP is in a way much like a company memo explaining the goals of the corporation in enthusing sound–bites. XP focuses on four and later after the second edition of the treatise, five core values, being:

Communication
Communication in the XP sense means that developers and end–users have frequent meetings to talk about the product that they are going to produce and share relevant knowledge with each other.

Simplicity
The simplest solution is always the starting point, developing this to a better solution meeting the requirements as they are at that moment. XP deals with "the now", rather than programming a solid basis which is able to support future necessities. Development with the future in mind is regarded as inefficient and a drain on resources, because a lot of possibilities which were taken into account during the development cycle might not be needed in the future or have become obsolete because of shifts in the requirements of the final product.

Also, the start of development cycle is marked by commencing with the simplest solution possible and slowly building up to more complex systems, this is called "re–factoring". A developer can also communicate a simple solution easier with fellow programmers and end–users.

Feedback
Feedback is received from end–users, the team as a whole and by testing the product itself.

Courage
Developers have to be courageous when it comes to their coding practice. A programmer must be able to take radical decisions during a development cycle, even if it means throwing the result of several days of substantial programming.

Respect
Team members have to respect each other and their work, so that a coherent software project will be allowed to emerge.

These five values are important in the "Extreme" development process which breaks down in four stages.

Coding is a phase in the development cycle, in XP this is regarded as the most important stage of development. Coding is perceived as a way of communicating, programmers can communicate problems or ideas with each other through code. Coding is also the way to test the most suitable solution. Rather than thinking about what the best solution might be, archetypical "extreme programmers" would code all the alternatives when presented with a variety of possible choices.
"Extreme programming" uses so-called "unit tests" to determine whether the coding phase is completed. Essentially a programmer will write as many automated tests as possible which attempt to break the code he or she is currently writing. When the results are positive – the code doesn't break – the coding phase is complete.

Ideally, developing a system can be done by just coding, testing and communicating the process and results with each other is enough to release a successful product. "Extreme programming" acknowledges that this isn't a realistic view on developing software. From time to time thing start to get more and more complex and the overall structure becomes unclear. Thus "design" enters the development cycle. A well–designed framework helps a development team being able to maintain focus on the overall structure of the product.

Extreme programming, and agile developing practices in general, became extremely popular during the dot–com hype during the late nineties. In a period where everything seemed possible and the demands of the market constantly rising and changing, a lot of developers adapted the new idea of extreme programming as opposed to the traditional developments.

Waterfalls

03

"The Waterfall Model" was introduced by W. Royce in 1970 – although the name "Waterfall" is said to be first used by software engineer Barry Boehm – and is still very common in contemporary programming. Royce, however, proposed this theory as being flawed and risky. He later proposed a final method generally called "the spiral model", which is an iterative approach. His original "Waterfall model" is still more popular than Royces later theories on software development.

> *"The fact of the matter is that, despite much progress, the Waterfall model isn't quite dead yet. A lot of people identify it as their development method of choice."*
> **("'The Demise of the Waterfall Model Is Imminent' and Other Urban Myths", Phillip A. Laplante and Colin J. Neill, Penn State University / acmqueue.com, 2004)**

The name "Waterfall model" comes from the visual representation of the model. A programmer proceeds through string of five stages of development, completing every stage before moving on to the next. The five stages, rather than the "values" found in "Extreme programming", are described as follows:

Requirements
Defining what the requirements of the software to be developed are.

Design
Designing a "road map" for the coding activities during the implementation process.

Implementation
This is the actual coding phase, where teams of coders work on different aspects of the software.

Integration
At the end of the implementation cycle the various pieces of code are assembled and integrated in a version of the eventual product.

Verification
Verification is the process of testing and debugging. Any errors are removed so that the product is ready for the next stage of development.

Installation
The completed product is installed.

Maintenance
The maintenance phase can be a very long cycle where additional bugs are removed and new versions of the software are introduced, adding new features to the completed application.

The "Waterfall method" builds on the idea that each stage is completed "perfectly" before moving onto the next. So, in theory, if the development team hasn't agreed with the requirements of the project they are going to develop, they can't move on to the actual designing of the software. One could say that every stage of development is equally important and has to be completed in order to achieve the final result.

Agility and Waterfalls

04

It's not hard to spot the differences between XP – or "Agile development" in general – and the "Waterfall method". The "Waterfall method" is in a way a very strict development process, where the various stages have to be completed and perfected in a sequential order. Whereas XP proposes a more asynchronous approach, with intertwined development cycles. Martin Fowler states in his article on "The New Methodology" that Agile development in general is indeed a reaction and improvement upon older, monumental methods, like the "Waterfall model".

> *"Engineering methodologies have been around for a long time. They've not been noticeable for being terribly successful. They are even less noted for being popular. The most frequent criticism of these methodologies is that they are bureaucratic. There's so much stuff to do to follow the methodology that the whole pace of development slows down.*
>
> *Agile methodologies developed as a reaction to these methodologies. For many people the appeal of these agile methodologies is their reaction to the bureaucracy of the engineering methodologies. These new methods attempt a useful compromise between no process and too much process, providing just enough process to gain a reasonable payoff."*
> **("The New Methodology", Martin Fowler, martinfowler.com, 2000)**

The XP development cycle generally isn't documented very well, because of the extensive verbal communication between team members and customers. "Waterfall method" aims for a well documented development cycle, so that no knowledge is lost if a team member leaves a project. A well documented development cycle is also vital when another team of developers, not involved with the building of the original software, wants to build newer versions upon the completed software architecture.

"Agile development" opposes extensive documentation, the explanation of Rachel Davies in an article on "requirements documentation" from 2005 explains to a certain extent why:

> *"Documents are a one–way communication medium; information flows from author to reader. There is no opportunity for the reader to ask questions, offer ideas and insights. The author may try to anticipate questions but cannot realistically be expected to address everything and in an attempt to cover everything, a concise abstraction may be lost in a forest of pages."*
> **("Agile requirements", Rachel Davies, Methods & Tools, 2005)**

Furthermore Davies argues that badly written documents could also lead to defective software. Writing and responding to documents also takes a lot of extra time, which is too valuable to waste and slows down the development process significantly, according to Davies. Additionally she opposes the way requirements are "set in stone" in the documentation. In the traditional "Waterfall method" all the

requirements for the final piece of software are accumulated and noted in the documentation. This list then becomes the working document, and thus the requirements are definitive and have become the absolute goals for the developers. Agile methods call for a more "natural" way of handling this. Requirements change often during the development process and cannot be written down as being definitive in a projects documentation. Davies claims that the best way to deal with changing requirements is not to freeze them on a certain level, but to adapt them during the development process.

The way requirements should be documented or shouldn't be documented at all is a point of an ongoing dispute between agile methods and the "Waterfall method". It's characterised by a concept called "Big Design Up Front", which is a key element of the "Waterfall method". Projects developed using the XP method are designed mostly during the process of coding and testing, while a software project developed according to the principles of the "Waterfall method" is designed almost entirely up front. The "Waterfall method" tries to define a complete project in advance, whereas the XP defines it during the development of the product itself. This is generally called "Big Design Up Front".
The "Big Design Up Front" idea takes the first stage or two stages of the "Waterfall method" as the ultimate starting point. The design of a program should be completed and perfected thoroughly before actually writing any code at all. This contradicts of course with the agile notion of code being the most important asset of a development cycle. Proponents of the agile methodologies criticize the idea of having a "Big Design Up Front", because they state that it is impossible to predict a complete development run and the problems related to it. Agile developers state that one can't foresee any problems without doing exhaustive prototyping and attempting to actually assembling and implementing portions of code.

Doing it open source

05

The "Waterfall method" and "Agile" practices are mostly popular in the world of commercial software development. Both methodologies claim to be the most efficient and solid way of producing software. In the open source realm it is harder to distinguish these methodologies on a macro level. This is of course due to the decentralized nature open source development. While parts of an open source project can for example be produced along the lines of "Extreme programming", the over arching paradigm concerning the actual implementation of all those individual isn't necessarily the same.

Erik S. Raymond describes the differences between commercial and open source software development models in his famous essay "The Cathedral & the Bazaar" (1997 – 2001). Although Raymond focuses on the differences between open source and free software development by documenting the production of "fetchmail" (a UNIX system–like mail client), the terms "Cathedral" and "Bazaar" are now commonly used in relation to propriety and open source software, respectively – even by Raymond himself.

> *"The real battle isn't NT vs. Linux, or Microsoft vs. Red Hat/Caldera/S.u.S.E. – it's closed–source development versus open–source. The cathedral versus the bazaar."*
> **("Halloween Document I (Version 1.16)", Eric S. Raymond, catb.org/~esr, 1998)**

One of the primary features of the "Bazaar" model as conceived by Raymond is the subsequent and frequent release of code to the public, a concept which is of course non–existent within commercial project development.

> *"Linus's open development policy was the very opposite of cathedral–building. Linux's Internet archives were burgeoning, multiple distributions were being floated. And all of this was driven by an unheard–of frequency of core system releases."*
> **("The Cathedral & the Bazaar", Eric S. Raymond, catb.org/~esr, 1997 – 2001)**

Frequent public releases ensure a large (beta–testing and co–developing) user base which is able to discern bugs and other problems quickly, ultimately leading to an increasingly solid software product.

Raymond summarized these concepts in nineteen standards, ranging from communication to the actual practice of code–writing. "The Cathedral & the Bazaar" still serves as an influential factor in relation to open source software development.

Niels Jørgensen has published an essay – "Putting it All in the Trunk" (2001) – on the development of FreeBSD, where he investigates the benefits and disadvantages a method of releasing the code of an entire operating system – being FreeBSD – in development, similar to the one described by Raymond. Jørgensen does this by documenting the various stages layed out in the several development guidelines of the FreeBSD project. Every time a FreeBSD developer changes an aspect of the

system in development, he/she is required to follow a certain cycle, which Jørgensen calls a "life cycle for changes". The "life cycle" consists of six incremental stages, being: Coding, reviewing, pre–committing, testing, development release, parallel debugging, production release. **("Putting it All in the Trunk: Incremental Software Development in the FreeBSD Open Source Project", Niels Jørgensen, Roskilde University, 2001)**

Jørgensen ends his essay with the conclusion that incremental development is a good approach for producing software, even if the software in question is quite substantial, which is the case with FreeBSD. However, for organizing very complex features – Jørgensen mentions "Symmetric Multiprocessing abilities" as an example – incremental development isn't the best solution. Furthermore, mixing the classical development approach (designing and planning everything up front) with the incremental one also produced a lot of problems regarding the integration and debugging of the components, whether finished or not, developed simultaneously but along different development paradigms.

Erik S. Raymond stated in his essay "Hacking and Refactoring" (2003 – 2005) that open source development and the "Agile" methodology are moving towards each other increasingly:

> *"I've seen agile concepts and terminology being adopted rapidly and enthusiastically by my colleagues in open–source–land — especially ideas like refactoring, unit testing, and design from stories and personas. From the other side, key agile–movement figures like Kent Beck and Martin Fowler have expressed strong interest in open source both in published works and to me personally."*
> **("Hacking and Refactoring", Erik S. Raymond, catb.org/~esr, 2003 – 2005)**

But Raymond continues radically by claiming that the "Agile principles" are indeed a reformulation of Unix development philosophies and hacker beliefs. The only difference between the "Agile method" and the traditional hacker mindset is that most people who advocate the "Agile" way have a commercial background and hackers are largely independent or working in the academic/specialist field, according to Raymond.

> *"The most important difference I see between the hackers and the agile–movement crowd is this: the hackers are the people who never surrendered to big dumb management –– they either bailed out of the system or forted up in academia or R&D or technical–specialty areas where pointy–haired bosses weren't permitted to do as much damage. The agile crowd, on the other hand, seems to be composed largely of people who were swallowed into the belly of the beast (waterfall–model projects, Windows, the entire conventional corporate–development hell) and have been smart enough not just to claw their way out but to formulate an ideology to justify not getting sucked back in."*
> **("Hacking and Refactoring", Erik S. Raymond, catb.org/~esr, 2003 – 2005)**

The answer to the ultimate question...

06

So after distinguishing three of the main popular software development methodologies, is it possible to say that one of the approaches is the best way to develop a software product? Of course the advocates of each of the paradigms claim that their way is the ultimate way to deliver, while the other method only leads to money wasting, crippled software or indeed chaos, but the truth is somewhat in the middle of either extreme.

The problem the "Waterfall method" is the fact that following a process of development as strictly defined as in both methodologies is almost impossible. David L. Parnas and Paul C. Clements wrote an essay on why this is not possible, but also provide a method to at least present a process as being developed along a rational methodology.

> *"We will never find a process that allows us to design software in a perfectly rational way. The good news is that we can fake it. We can present our system to others as if we had been rational designers and it pays to pretend do so during development and maintenance."*
> **("A rational design process: How and why to fake it", David L. Parnas and Paul C. Clements, Computer Science and Systems Branch, Naval Research Laboratory, 1986)**

Of course the essay by Parnas and Clements is from the eighties and "Agile developing" didn't exist until 2001, one of the main goals of "Agile development" is actually to be more adaptive to outside influences and other development related to a project in production. The main problem with the "Waterfall model" is the urge to design everything in advance and trying to predict all the possible risks and conditions of project. While the development requirements for future software can be defined globally, one should always take into account the dynamics of developing software. Requirements fluctuate or even change radically during a development process. If one, by following a certain methodology, can't adapt to the new conditions and change the design made up front, the outcome of the project will surely have weaknesses in several areas – for example failing compatibility with newer hardware which was introduced after the completion of the design phase.

Followers of the "Agile methodology" claim that "Agile methods" are indeed more adaptable to changing requirements and that the development process as a whole is fare more flexible than the predictive "Waterfall method". The "Agile manifesto" is a reaction to the "Waterfall methodology" and in many ways an improvement in general for developing software, according to its authors and proponents. The "Agile" way of adapting to changing requirements for a project by doing most of the design while actually coding parts of the project and choosing the simplest solution and then re–factoring it into gradually more complex solutions seems like a very logical way of developing. However, there is the risk that by doing little or no design work beforehand one cannot be prepared to anticipate and overcome major problems during the development process, a lot of redesign effort then has to be completed which could have been avoided otherwise.

Also "designing with code" and re–factoring written code requires quite a lot of skill and can be hard for junior programmers to adapt. Additionally the documentation of projects in "Agile development" is limited at best, so stepping into a project in development or quickly adapting changes will probably only be possible for experienced developers.

In open source development there is no such thing as a central management board which makes overall decisions and judges a projects outcome. Having no central management or body which is able to plan and make design decisions can transform an open source endeavour into a rudderless venture. However, Niels Jørgensen argues that the management factor in open source projects has replaced by something as strong as traditional management, that being massive peer review:

> *"The absence of a traditional management authority is highly valued by the project's participants, and respondent's comments such as "embarrassment is a powerful thing" indicate that is has been replaced by an effective, but also somewhat frightening peer regime."*
> **("Putting it All in the Trunk: Incremental Software Development in the FreeBSD Open Source Project", Niels Jørgensen, Roskilde University, 2001)**

This extensive peer review system of the open source community is indeed a strong instrument for producing and improving software. But there's also the danger of something which is best described as clashes between developing philosophies within the open source community. The recent outburst of Linux Torvalds on the GNOME usability mailing–list in late 2005 is a nice example of such a clash. Torvalds didn't like the approach of the GNOME community of keeping things as simple as possible for the user and thus, in his words, marginalizing the functionalities of a GUI environment, like GNOME. On the GNOME mailing–list Torvalds blatantly advised users to use the other major desktop environment, KDE. In reply to GNOME developer Jeff Waugh, one of the many respondents to his exclamation, Torvalds wrote:

> *"No. I've talked to people, and often your "fixes" are actually removing capabilities that you had, because they were "too confusing to the user".*
>
> *That's _not_ like any other open source project I know about. Gnome seems to be developed by interface nazis, where consistently the excuse for not doign something is not "it's too complicated to do", but "it would confuse users"."*
> **("[Desktop_architects] Printing dialog and GNOME", Linus Torvalds, GNOME Usability Mailing–list, 2005)**

Often these discussion happen on all levels and stages of development, sometimes culminating in a battle of two opposite "extremes" like KDE versus GNOME. Because they are discussions about extremes and often fundamental parts of a software product, the outcome doesn't present a clear solution or any workable solution at all. That might also be a reason why there is such an overwhelming choice in Linux OS flavours and several instances of any given type of application for the Linux platform, be it a search tool or a desktop environment.

In that light could one give an answer to the question "Is there an ultimate model for software development?". A mix of all the methodologies mentioned previously might be the best model. While using the XP method when prototyping an experimental product might be the key to success,

utilizing XP exclusively when developing very large projects could result in failure, because of the lack of up front design and predefined requirements. Using the "Waterfall method" solely for any project could result in overplanning and in the end failing to meet with fluctuating requirements and contemporary standards in what the base features of a software product should be.
Implementing the open source model in commercial projects is off–course nearly impossible because of its decentralized development and strong philosophy. But "Agile development" has successfully adapted several of the methods commonly accepted in the open source realm – like re–factoring code, frequent version releasing, etc.

The ultimate answer to what the ideal methodology for developing software is, isn't answered by simply saying the "Waterfall method" or "Agile development", because any methodology has its own significant advantages and disadvantages. Ideally the development of a software product is a fast, efficient process, which additionally produces a lot of meaningful documentation for future development and in the end the product is of top notch quality both quality–wise and technically. Following either methodology strictly means that one has to cut on specific features in order to comply to the ruleset of the method. Also, one has to take into account that particular project requires a particular approach. Maybe an approach even so particular that developing along the lines of any predefined paradigm is to general and constraining.

As with every extreme question related to the world of computing – "What is the best operating system?", "Which programming language is the best?" – there is no ultimate answer, although a lot of people will admit otherwise.

Marc de Bruijn
MA Media Design, Piet Zwart Institute
Rotterdam, May 2006

References

07

• *"History: The Agile Manifesto"*
Jim Highsmith
Agile Alliance, 2001
http://agilemanifesto.org/history.html

• *"The New Methodology"*
Martin Fowler
martinfowler.com, 2000
http://www.martinfowler.com/articles/newMethodology.html

• *"Agile requirements"*
Rachel Davies
Methods & Tools, 2005
http://www.methodsandtools.com/archive/archive.php?id=27

• *"Halloween Document I (Version 1.16)"*
Eric S. Raymond
catb.org/~esr, 1998
http://www.catb.org/~esr/halloween/halloween1.html

• *"The Cathedral & the Bazaar"*
Eric S. Raymond
catb.org/~esr, 1997 – 2001
http://www.catb.org/~esr/writings/cathedral–bazaar/

• *"Putting it All in the Trunk: Incremental Software Development in the FreeBSD Open Source Project"*
Niels Jørgensen
Roskilde University, 2001
http://webhotel.ruc.dk/nielsj//research/papers/freebsd.pdf (offline)

• *"Hacking and Refactoring"*
Erik S. Raymond
catb.org/~esr, 2003 – 2005
http://www.catb.org/~esr/writings/hacking–and–refactoring.html

• *"Torvalds Says 'Use KDE'"*
CmdrTaco
Slashdot, 2005
http://linux.slashdot.org/article.pl?sid=05/12/13/1340215

• *"[Desktop_architects] Printing dialog and GNOME"*
Linus Torvalds
GNOME Usability Mailing–list, 2005
http://mail.gnome.org/archives/usability/2005–December/msg00021.html

• *"Manifesto for Agile Software Development"*
Kent Beck, Ward Cunningham, Martin Fowler, Jim Highsmith, etc.,
agilemanifesto.org, 2001
http://agilemanifesto.org/

• *"Principles behind the Agile Manifesto"*
Kent Beck, Ward Cunningham, Martin Fowler, Jim Highsmith, etc.,
agilemanifesto.org, 2001
http://agilemanifesto.org/principles.html

• *"The Linux managing model"*
Federico Iannacci
First Monday, 2003
http://www.firstmonday.dk/ISSUES/issue8_12/iannacci/index.html

• *"'The Demise of the Waterfall Model Is Imminent' and Other Urban Myths"*
Phillip A. Laplante and Colin J. Neill
Penn State University / acmqueue.com, 2004
http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=110

• *"A rational design process: How and why to fake it"*
David L. Parnas and Paul C. Clements
Computer Science and Systems Branch, Naval Research Laboratory, 1986
http://www.ece.utexas.edu/~perry/education/360F/fakeit.pdf

• *"Project Lifecycles: Waterfall, Rapid Application Development, and All That"*
The Lux Group, Inc
http://luxworldwide.com/about/whitepapers/waterfall.asp