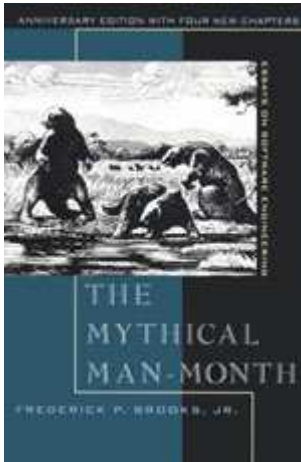


The Future of Software Development

Written by [Alex Iskold](#) / October 16, 2007 1:28 PM / [56 Comments](#)



In 1975, Frederick Brooks wrote a classic book on software project management called [The Mythical Man-Month](#). In the book, he famously argued that adding more people to a development project will hinder rather than help to get things done faster. The reason is that having more people working on the project introduces a non-linear overhead in communication.

Five years before Brooks' book, a software development methodology called the [Waterfall Model](#) was coined. This approach applied the insights from mature engineering disciplines (mechanical, civil, etc.) to software. The idea was to construct systems by first gathering requirements, then doing the design, then implementing it, then testing, and finally getting it out the door in one linear sequence.

We have come a long way since then and learned a lot about making software. The Waterfall Model is now considered a flawed method because it is so rigid and unrealistic. In the real world, software projects have ill-defined and constantly evolving requirements, making it impossible to think everything through at once. Instead, the best software today is created and evolved using agile methods. These techniques allow engineers to continuously re-align software with business and customer needs.

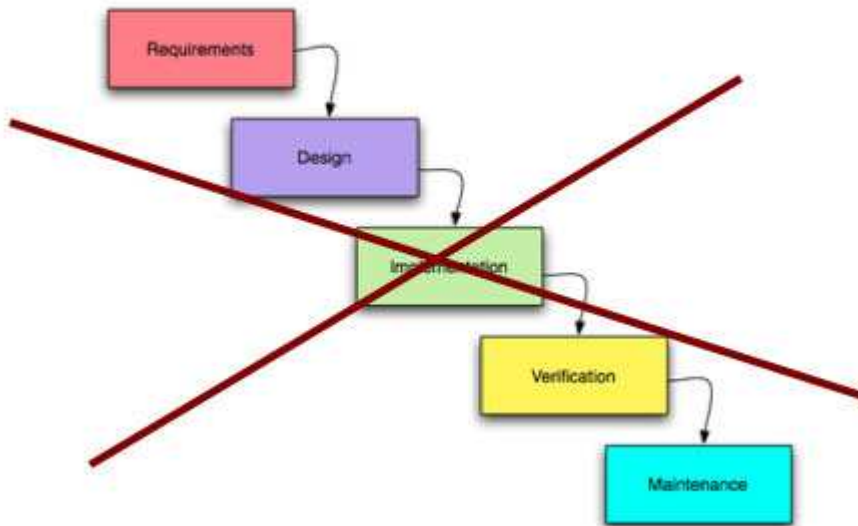
With the advent of modern programming languages (Java, PHP, Python and Ruby), rich libraries, and unprecedented infrastructure services like those from Amazon, we are arriving at yet another evolutionary step. Digg, del.icio.us, YouTube and other poster children of the new web era were developed by just a handful of programmers. To build software today all you need is a few good men (or women!). In this post we trace how we got here and where we are heading next.

Why The Waterfall Model Failed

Non-technical people tend to think that software is **soft** or easily changeable. Since there are no visible nuts and bolts and no hood to open people think that software can be tweaked and re-wired on a whim. Of course, this is not the case. Software, like any mechanical system, has a design and the structure; it is not as soft as it seems.

Yet the accelerating pace of business requires constant changes to software. Older development methods completely fail to address business needs. Using the Waterfall Model, these changes were impossible, the development cycle was too long, systems were over engineered and ended up costing a fortune, and often did not work right.

Waterfall Software Development Model



The problem was that the Waterfall Model was arrogant. The arrogance came from the fact that we believed that we could always engineer the perfect system on the first try. The second problem with it was that in nature, dynamic systems are not engineered, they evolve. It is the evolutionary idea that led to the development of agile methods.

Agile Methods - Evolving Software

In the early nineties a number of [agile software development methods](#) emerged. While they differed in details, they agreed at large that software development needed a major rethinking. First, software has to embrace change. Today's assumptions and requirements may change tomorrow, and software needs to respond to changes quickly. To meet the challenge, agile approaches advocate focusing on simplicity. Make the simplest possible system that satisfies today's requirements and when tomorrow comes, be ready to adapt.

Agile Software Development Principles

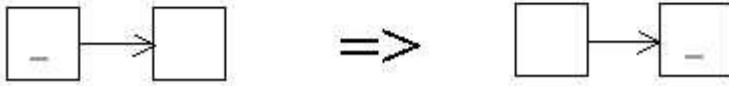
Have fun	Focus on Simplicity	Adapt the code
Embrace Change	Get Feedback	Refactor
Communicate	Release Often	Test

Two techniques pioneered by agile methods are worth particular attention - refactoring and developer testing. Refactoring, elegantly described by Martin Fowler in his [classic book](#) is the idea of improving the design of the existing code without changing how it works.

Move Feature

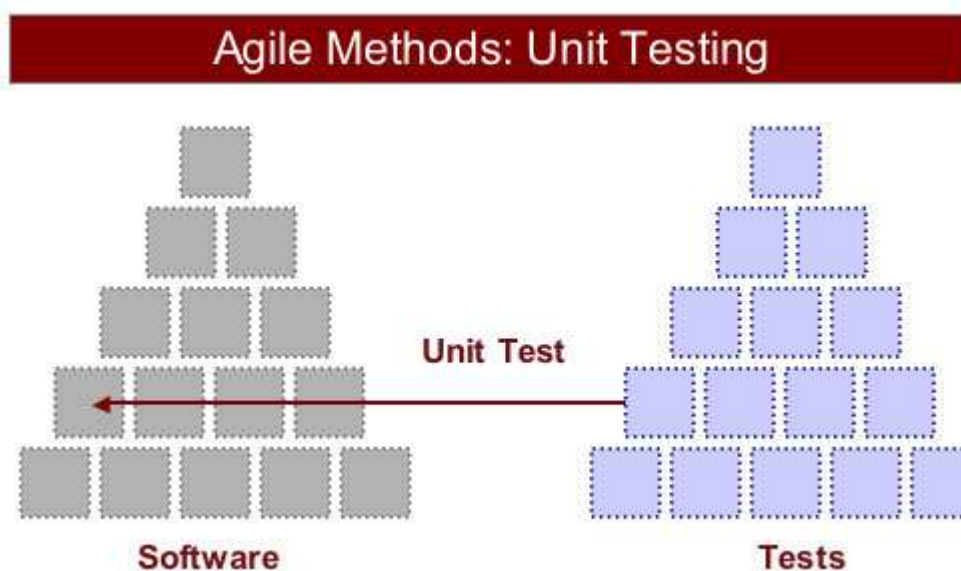
*a class implements
a foreign feature directly*

*move the feature
to its natural home*



Refactoring is what allows agile systems to embrace change, while remaining elegant and free from rot. Like an interior decorator continuously changes and improves the layout of your furniture, agile developers move code around to improve the product as a whole. Code is constantly changed to make sure we end up with the simplest, and best possible system that reflects our current needs.

To make sure that changes do not break the code, agile methods introduced unit tests. As each agile project unfolds, it grows the base of unit tests. Each test is focused on a single component of the system and acts as an insurance that the component works as expected. Typically, the tests are run continuously against the code and require immediate fixes in case of a failure.



The software systems created using agile methods are much more successful because they are evolved and adapted to the problem. Like living organisms, these systems are continuously reshaped to fit the dynamic landscape of changing requirements. Without a doubt, agile methods made a major impact on how we think about building software today - dynamically and continuously.

It's The Libraries, Stupid!

While we discovered better way of making software, we also discovered better programming languages. C was replaced with C++, then came Java. Perl was great, but PHP and Python took its lessons further. More recently came Ruby, which has become very popular because of its natural way of expressing code. Because of this evolution, today we have a number of excellent, and virtually equivalent programming languages.

While the choice of programming language is typically a sensitive subject, the truth is that it is not the language, but the libraries that come with it that make a difference. C++ never had the standard libraries

that Java has. Yes, Java is the simpler language, but people used C++ for a decade just fine. What gives Java the edge is its rich set of reusable libraries. The story is similar with PHP. It has been the language of choice for web developers precisely because it comes with such rich support for web and database processing.



In addition to the libraries that come with modern languages, the open source movement has also contributed a wealth of code towards global software infrastructure. Notably, just the Apache foundation on its own has created a huge amount of high quality reusable code. We have now arrived at an age where we have a strong foundation for building complex software systems. We know the methods and we have the tools, so what does that mean?

The Future of Software Development: Just a Few Good Men

Since early days of software development people struggled to build good systems. More and more people were thrown at the problem, making matters worse. But with the recent explosion of social web we've witnessed a new and interesting phenomenon: a handful of developers are now able to build systems that are used by millions of people. How can this be?

The secret is that as with any good endeavor it only takes a few good men (and/or women!). With a bit of discipline and a ton of passion, high quality engineers are able to put together systems of great complexity on their own.

Equipped with a modern programming language, great libraries, and agile methods, a couple of smart guys in the garage can get things done much better and faster than an army of mediocre developers.

Software Engineering: The Next Generation



Modern
Programming
Languages

Libraries /
Infrastructure

Agile
Methods

Alex Iskold for Read/WriteWeb

We are likely to see a few changes over the coming years:

- High-quality, passionate software engineers will be in very high demand and will make substantially more money.
- The developers who do not have great programming skills are going to have to look for jobs elsewhere.
- The changes that we are witnessing today in the social software market are going to reach the enterprise level.
- Software off shoring will make less and less economical sense.
- Computer science is going to remain a highly competitive and prestigious field.

Conclusion

Ironically, we are coming full circle with the mythical man-month. What was true twenty years ago is true of course today, but for a whole new set of reasons. An awesome array of programming languages and infrastructure libraries combined with agile methods has allowed us to break free of old software development dogmas. Just a handful of great engineers can now successfully build systems of great complexity. Craftsmanship has finally come to software engineering!