

2 Pages on Agile (Software) Methodologies



Introduction

This document reveals the extremely obvious secret why most agile methodologies (such as Scrum) actually work, and why most heavy-weight processes (such as RUP) don't. It bothers me quite a bit that so many people fail to see what's actually going on when it comes to software development. I'll try to make a small difference in that regard. Will I be able to explain the evident in just two¹ pages? Perhaps.

The Fallacy of Perceived Control

Almost every (software development) manager in the world wants to know what those damned programmers are actually doing. The only way, it seems, is to give them detailed orders of what they are supposed to do. Those orders come in a variety of disguises; design constraints (for example in a SDD), requirements (for example in a SRS), and so forth. Problem: You cannot easily specify all details for something inherently complex. Since your specifications implicitly (by omission) also determine the things that do NOT need to be done, creative people are going to make a big mess of your documentation-intense, and ridiculously futile, attempts at gaining control. It's simply not possible to handcuff artists by restricting their palettes and brushes. They'll just paint something ugly for you (and it will always be ready in exactly a week from now), delivered right from the intestines of their claustrophobic and lonely cubicles. But there is another way of getting the results you want, and it's achieved through a little something called group dynamics.

Empower Your People

Smart managers make sure that other people do most of the useful work, since that's the only way for them to achieve true scalability (and to avoid having to be the ones doing the work). To avoid letting programmers roam free and have software engineers play their crazy über-generic "lonely wolf" hack games – which will lead to

¹ Originally, the title of this document was "1 Page on Agile (Software) Methodologies".

nowhere and create diminutive customer value – you have them build **teams** and give them **power** to make **decisions**. For some strange social anthropological reason, this actually works, because we (programmers *and* humans) are social beings that operate successfully in lots of different social contexts. The groups will slowly shape the raw material of sadly lost individuals into solid high-performing steel. There’s virtually no limit to what functioning groups can do. (The manager’s job is simply to keep the groups healthy. Easy as pie.)

The Big Secret is Spelled Communication

Now that you have read between the lines about the software industry’s glittering waterfall that has brought us tumbling down to the current state of chaos², I still have some room left to tell you what the fuss is really about when you look under the covers of all the manifests, acronyms, and mighty fine speeches from great people in the forefront of agile methodologies. It’s all about **communication**. With the documentation-centric methodologies, people communicate through crappy software specifications, and ask their day-to-day questions using email. It’s slow and inefficient, and worse, it’s ignoring the fact that the complexity of software far exceeds the shallow treatment even a long-winded document can give it. Non-linear complex systems always defy attempts at simplifying them in a single or a few lousy dimensions. I know that some people won’t believe me; at least not until they’ve experienced it.

Enough of that! Put people together in a room and give them challenging tasks with a simple framework (and goals!) that defines what they’re supposed to do. Magic happens. Guess what? People are actually quite excellent at collaborating and handling massively complex tasks when given direct **access to knowledge** and granted rights to make important decisions. When building software systems, the knowledge exists in the minds of the people designing and implementing them. They cannot reasonably put all that knowledge on paper. They cannot possibly make a list of all questions that will need answers. It’s ludicrous to think that their manager (or worse, a product manager) can give them a piece of paper that tells them exactly what to build and how. But they are all capable of answering the questions LIVE. They can also formulate the questions directly, and clarify when something is not clear. They can, believe it or not, manage two-directional communication with ease. In fact, that’s what most people are good at; much better than at writing speculative documents that will never be read or understood.

Another piece of the puzzle falls into place when the group dynamics really kick in, and the individuals in a team start taking responsibility. Suddenly, things that were previously lost between unwritten documents are instantly brought forth into the light and decided upon. Managers are forced to do their real jobs, which is to get rid of impediments and make it possible for their people to do what they are hopefully best at; creating world-class software.

Of course, if managers of software houses try to do their jobs from the safety of their spacious offices, they’re in for a surprise of the same magnitude as the people who still believe that complexity can be tamed by increasing the documented detail resolution for the constituent parts of a system. But that’s another story.

Thank you for reading,

Bjorn Karlsson

² There’s **always** going to be chaos until I have mathematically explained to you my dawning theory in *“Managing complexity in parallelized systems by restricting the available reusable operators”*. Or something to that effect. That’s going to be a 3-page document, at least.