Table of Contents

XP (eXtreme Programming) extolls the virtues of “simple design” but I always felt that the concept, while something that we should strive for, was not exactly clearly defined.

The agile principles call for “simplicity, the art of maximizing the amount of work that is not done”.

Both talk about simplicity at one level, but they also have a tendency to be applied differently depending whether you are looking through the lens of “requirements” or “development”.

Requirements View

The idea from a requirements viewpoint is to stop producing work that nobody needs or uses. Traditionally the source of these features has been our development process. If we tell people that they need to ask for everything upfront, and that it will be expensive for people to change their minds, then it is logical that people will ask for everything, whether it is useful or not. This pretty much described our traditional development process. In addition, because we are trying to make sure we are getting providing useful capabilities to our customers, developers will often add in additional capabilities that the customer might need (“gold plating”).

With agile we have an iterative and incremental approach to development. Since the cost of change is cheap, we can avoid both the need for upfront specification of requirements, and for gold plating because we have a regular conversation with the customers and can steer our development accordingly.

Development View

From "The Secret Assumption of Agile" by Fred George we learn that “simple design” from a developers perspective is:

1. The software works - not a bad start.
2. The software communicates its intent - XP practitioners say “the software should scream intent”. If the softer is solving an accounting problem, then it is clear it is solving an accounting problem. This contrasts with the normal way we “design” a system in that we specify things like the framework, the language, etc and describe the application as “a web based application driven by micro-services”. This could be true, but it is not the intent of the system. But it also is applied at many levels - if you feel a need to comment the code, then perhaps the code does not communicate intent?
3. No duplicate code - since, lets face it, duplication is bad.
4. Least classes and methods - and this means any extra code that we don't need. Just kill it. IF there is a piece of code that is commented out, kill it. Don't read it, because then you'll be trying to figure out if it is useful or not. And if someone else comes along, they'll be wondering the same things. What a waste.

These ideas are applied to both application code and the tests.

Team, XP, Simple, FAQ