A Reaction to XP: Xtreme Programming

The methodology of Extreme Programming is a long standing controversy in the software engineering world. The articles on the Internet I had previously read regarding the methodology were, for the majority, clearly critical of its failings rather than enthusiastic about the prospect of a new, improved way to code. These articles’ attitude led me to develop something of a critical attitude towards the system myself, though it was more one of curiosity since I did notice that for all the opponents’ words, they offered no alternative to the problems that XP treats.

In Dr. Roberts’ lecture, he gave a first-hand example of those problem areas that XP handles well, while minimizing his coverage of the drawbacks of the new system. This gave me a more balanced view of the XP methodology, one in which I can now see how the strengths of the system yielded direct and obvious improvements over previous ones. Dr. Robert’s explanation of his work with the asset liability management software team also helped me understand the practical application of XP in a very real-world way rather than how I thought of it before, as just a list of concepts. Whether or not XP is a good fit for every programming job is impossible to say, but the improvements gleaned in migrating away from the legacy software engineering methodology at the finance firm were clear.

XP advocates some very unorthodox practices (at least, unorthodox for the software engineering world) in order to yield positive result. XP’s rewrite of software engineering culture is a very interesting concept to me. I think inter-team, face-to-face communication in software engineering is incredibly important to the overall robustness of the software product, and thus XP’s focus on such a topic is well warranted – even though many of us would likely prefer to be
sequestered away, writing code to solve issues exactly how we want. I also thought that the concept of purposely creating situations where “residual communication” (basically overhearing your colleague’s conversations) is inevitable, for the purpose of boosting productivity, was very novel. A huge change for programmers would be the focus on pair programming. Egos would have to be shoved under the rug or conflicts would erupt as the person watching his or her partner write code would get annoyed at the apparent lack of skill of said partner, even though the real issue is just that any two people will likely approach the same problem in different manners. I like the idea of pair programming, though I think it would take a good amount of getting used to; I would be fairly nervous being the one in the “driver’s seat” until I got comfortable with being reviewed by my colleagues. The proposition that pair programming is just as efficient or more so as the style of one-person-one-computer is fascinating to me, as it illuminates what really takes time when writing code: not the writing at all, but the algorithms, plans for the structure, and debugging. Increased communication and pair programming are two interesting concepts that seemed to alleviate the problems that Dr. Roberts’ firm was having before his arrival, seemingly caused by the failing old software engineering methodology.

I feel that the important point to take from Dr. Roberts’ pseudo-case study is that not only did the productivity and reputation of the development team improve when using XP, but also that the programmers themselves swore by the new ideas after giving them a chance. This support from the actual employees seems to me to be a sign that XP is a superior methodology, or at least it is such in situations similar to the one that Dr. Roberts helped rectify. I am sure my opinion on the methodology will change as I hear more cases, both in which it helped and also sometimes hurt, but at least after this lecture I have a more solid basis of knowledge on the implementation of Extreme Programming.