The Only Good XP is Windows XP
Introduction

Extreme Programming ("XP") is supposed to be the latest and greatest programming methodology. With XP, there is no code ownership, programmers work in pairs, little or no documentation is kept, and the project is always in a state of flux, constantly being redefined by the programmers and the customers. These principles of XP are drawing attention both from executives looking to cut costs and new programmers looking for the easy way to program. However these two groups, among others, fail to see XP correctly. While Extreme Programming supposedly is a better way to program, the implementation costs and dedication required for success doom any XP project to certain failure.

How XP Works

The first step to Extreme Programming is that the client develops “stories”, or the basic ideas of what they want. The programming team then estimates how long each story would take and allow the client to decide which stories will be developed in the next build. Each selected story is then broken down into tasks. The programming team then develops tests for each task. At this point, the programming team breaks out into pairs to begin programming. Each pair shares a computer and alternate who types. The programmer that is not typing watches and checks the code that is being written. Once the code is finished, it is integrated into the current build. It is then tested with the test developed before they started coding and regression tested. If the tests fail, it is rewritten (refactored) until it does work [1].

Extreme programming also has several other requirements that are unique to XP. First, the programming team shares one large room without individual cubicles. Second, all the
computers can be used by any programming pair. Third, there is always a client representative on site to answer questions and provide feedback. Fourth, all programmers are equal in that everybody can work all the code[1]. These requirements, along with the basic workflow for XP contribute to its demise.

“The Case Against Extreme Programming” by Matt Stephens

In this article, written in 2001, Matt Stephens present his case against Extreme Programming. While his overall case against XP is fairly strong, there are several arguments that exceptionally strong while only a couple are exceptionally weak. The following is a critique of his critique.

Stephen’s first criticism of Extreme Programming is its “self-referential safety net” [2]. Stephen’s says “XP is like a ring of poisonous snakes, daisy-chained together. All it takes is for one of them to wriggle loose, and you’ve got a very angry, poisonous snake heading your way” [2]. This example shows what can happen when XP is not fully implemented or when not executed correctly. If any snake, or XP rule, is broken it causes the entire process to unravel. This is because no single XP rule is strong enough to ensure quality and each rule is needed to make up for the shortfall of another rule. This example works well because it puts all rules on the same level – they are all snakes. To the casual observer, some XP rules seem less important than others. But by comparing all rules equally to snakes, it reinforces that everything must be implemented completely and fully. This is important for all parties concerned to remember. If they try to break the chain of snakes and only partially implement XP, they will get bitten by the snake and fail.
Another criticism Stephen has of Extreme Programming is the need for “Optional Scope Contracts”[2]. As he describes, with XP a customer “signs up for a project that runs for a fixed amount of time for a fixed price, without any commitment as to what is actually delivered” [2]. I take fault with Stephen’s argument because I don’t think Stephen makes this point clear enough and does not give it the importance it deserves. For example, how do you explain to a potential customer that after x amount of days and y amount of dollars that no solution exists? I think this shows the craziness of XP; both in the policy and in trying to explain it to non-technical people. Therefore I find fault with Stephens for not going over this detail in greater detail.

Finally, it all comes down to change. Extreme Programming is supposedly wonderful at accepting change. However XP is not suitable for change. As Stephens says “there are processes that are much better suited to handling change halfway through a project” [2]. For example, suppose a team is working on a project and programming pair A make a change to a common module. And then suppose that the next day, programming pair B undoes that change. Because there is no documentation or planning, it is impossible to foresee or deal with this issue until it happens. This cycle has the possibility of continuing for a while until all the programmers are on the same page that a change needs to occur. XP also fails to handle changes from the customer well. For example, if a customer requests a change, there is no mechanism in XP to quickly determine the cost to develop it. It is also impossible to tell what that change may cause throughout the rest of the system. XP also provides no way off documenting when these changes occur. Overall, the laziness of documenting progress discredits one of the main reasons for XP.
In conclusion, XP is not all bad. However, as Stephens points out, it is not perfect. It is no magic bullet to programming. While XP may work in some situations, those situations require a programming team and customer that all get along well. In my opinion, XP may be best suited to long term consulting contracts only. The system is too loose and too open for a new working relationship to be established and work out.
References
