Extreme Programming

One of the foremost Agile methodologies is called Extreme Programming (XP), which involves a high degree of participation between two parties in the software exchange: customers and developers. The former inspires further development by emphasizing the most useful features of a given software product through testimonials. The developers, in turn, base each successive set of software upgrades on this feedback while continuing to test new innovations every few weeks.

XP has its share of pros and cons. On the upside, this Agile methodology involves a high level of collaboration and a minimum of up-front documentation. It's an efficient and persistent delivery model. However, the methodology also requires a great level of discipline, as well as plenty of involvement from people beyond the world of information technology. Furthermore, in order for the best results, advanced XP proficiency is vital on the part of every team member.

Extreme programming (**XP**) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development, it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.

Other elements of extreme programming include: programming in pairs or doing extensive code review, unit testing of all code, avoiding programming of features until they are actually needed, a flat management structure, code simplicity and clarity, expecting changes in the customer's requirements as time passes and the problem is better understood, and frequent communication with the customer and among programmers. The methodology takes its name from the idea that the beneficial elements of traditional software engineering practices are taken to "extreme" levels. As an example, code reviews are considered a beneficial practice; taken to the extreme, code can be reviewed *continuously*, i.e. the practice of pair programming.

Activities

XP describes four basic activities that are performed within the software development process:coding, testing, listening, and designing. Each of those activities is described below.

Coding

The advocates of XP argue that the only truly important product of the system development process is code – software instructions that a computer can interpret. Without code, there is no working product. Coding can also be used to figure out the most suitable solution. Coding can also help to communicate thoughts about programming problems. A programmer dealing with a complex programming problem, or finding it hard to explain the solution to fellow programmers, might code it in a simplified manner and use the code to demonstrate what he or she means. Code, say the proponents of this position, is always clear and concise and cannot be interpreted in more than one way. Other programmers can give feedback on this code by also coding their thoughts.

Testing

Extreme programming's approach is that if a little testing can eliminate a few flaws, a lot of testingcan eliminate many more flaws.

- Unit tests determine whether a given feature works as intended. Programmers write as many automated tests as they can think of that might "break" the code; if all tests run successfully, then the coding is complete. Every piece of code that is written is tested before moving on to the next feature.
- Acceptance tests verify that the requirements as understood by the programmers satisfy the customer's actual requirements.

System-wide integration testing was encouraged, initially, as a daily end-of-day activity, for early detection of incompatible interfaces, to reconnect before the separate sections diverged widely from coherent functionality. However, system-wide integration testing has been reduced, to weekly, or less often, depending on the stability of the overall interfaces in the system.

Listening

Programmers must listen to what the customers need the system to do, what "business logic" is needed. They must understand these needs well enough to give the customer feedback about the technical aspects of how the problem might be solved, or cannot be solved. Communication between the customer and programmer is further addressed in the planning game.

Designing

From the point of view of simplicity, of course one could say that system development doesn't need more than coding, testing and listening. If those activities are performed well, the result should always be a system that works. In practice, this will not work. One can come a long way without designing but at a given time one will get stuck. The system becomes too complex and the dependencies within the system cease to be clear. One can avoid this by creating a design structure that organizes the logic in the system. Good design will avoid lots of dependencies within a system;this means that changing one part of the system will not affect other parts of the system.

Advantages

- Robustness
- Resilience
- Cost savings
- Lesser risks

Disadvantages

- It assumes constant involvement of customers
- Centered approach rather than design-centered approach
- Lack of proper documentation