LTIMindtree

POV

Can **Extreme Programming** (XP) Help Encounter Attrition?

Authors:

Nilesh Dubey

Can Extreme Programming (XP) Help Encounter Attrition?

Repeatedly most of the teams have experienced attrition which impacts the working of the team negatively. Attrition can be due to varied reasons for different teams; however, its impact is the same on all teams with different magnitudes. In the software development industry, attrition is a common phenomenon, and teams must recognize this as a potential risk and be prepared to encounter it. Improved employee engagement, a positive workplace environment, competitive pay, and benefits are certainly effective means to reduce attrition, but they fail to encounter the effect of attrition.

Can we encounter and minimize the impact of attrition by any means? The answer to this question is the adoption of a way of working which can reduce the dependency on an individual team member and makes the new team member productive in lesser time.

Let us see if a particular software development methodology can tackle the effect of attrition. There has been a buzz in the software industry about agile development methodologies in recent years. One such agile software development methodology is Extreme Programming (XP) which nominates coding as the key activity throughout a software project and takes good common sense to extreme levels. Let's see how different practices of XP help in encountering the impact of attrition.

Extreme Programming (XP) practices help to encounter the impact of attrition

Extreme Programming (XP) is a lightweight agile software development methodology that organizes people to produce higher quality software more productively. XP is a discipline of software development where teams are required to do certain things to be doing XP. The practices in XP are not new, in fact, most of them are as old as programming.

XP practices involve pair programming, refactoring, unit testing, continuous integration, everyone's involvement in defining and refining the architecture, and short iterations which have a close resemblance to any other agile development methodology and can encounter the impact of attrition on the team's productivity up to a larger extent. XP makes sure that all these practices support each other to the greatest possible degree. Below given are a few practices recommended by XP which can help to tackle the effect of attrition in an effective manner.

01 Use of metaphor to build collective understanding about the project

Each XP software project is guided by a single metaphor which enables the whole team to visualize the resulting software in a certain way common to every member of the team. For example, like saying "the pension calculation is like a spreadsheet," "application like a cheque-book" builds a collective understanding within the team about the project. The use of metaphor helps everyone on the project to understand the basic elements and their relationships and enables them to have an analogy for better understanding. The use of metaphor gives a story to the team that can be easily shared by the business and technical folks.

Newly onboarded members of the team can easily get the high-level gist of the project by just knowing the metaphor used within the team for the project. Metaphor eases the process of thinking about future requirements and probable solutions. This reduces the effort and time in bringing the new member of the team up to the desired level of understanding about the project and reduced the learning curve with respect to the functionality of the application. However, the use of metaphor is one of the least followed and underestimated practices of Extreme Programming methodology.

02 Use of Pair-Programming for continuous knowledge sharing

This is the practice followed in Extreme Programming (XP) methodology where all the production code is written with two people looking at one machine, with one mouse and one keyboard. There are two roles in each pair one who thinks about the best way to implement a particular functionality and the other who thinks more strategically if the whole approach is going to work, will all the other test cases pass, and will solution cater to all the testing criteria.

This approach to development enables both people to have the same understanding of the implementation and reduces personal dependency within the team.

XP recommends that the pair should be dynamic. If the person in the team is responsible for a task unfamiliar to him can ask someone with recent experience to pair with him. This enables the smooth flow of knowledge within the team and reduces the dependency on a person or a pair. A new person in the team can pair on a task with someone who is having experience on a similar task and can be productive in lesser time.

03 Availability of on-site customers to reduce functional knowledge dependency

As per the XP practice, a real customer must sit with the team, be available to answer questions, resolve disputes, and set small-scale priorities. The customer can produce value for the project by writing functional test cases and thereby helping the team to do the User acceptance test (UAT) level of testing at the very stage of the development.

In the majority of the development or enhancement projects, resources who have worked on the application for a significant duration acquire more functional knowledge compared to a new team member, but they may not have a significant difference in technical skills. In such scenarios, an on-site customer is a readily available source of functional knowledge of the project who can resolve the functional queries of the team in lesser time and can effectively eliminate the dependency on older resources in terms of functional knowledge.

04 Collective Ownership to reduce dependency on individual heroics

Traditional models of code ownership were no ownership and individual ownership. In the no-ownership model, nobody owned a particular piece of code, and anybody could modify it to suit their purpose whether it was fit to what was already there or not and resulted in chaos. In the individual ownership model, only one person could change the code as per the requirement, this helped to keep the code stable but introduced a heavy dependency on individuals. XP focuses on the collective ownership of the design, architecture, and code.

Collective code ownership made everyone responsible for the whole of the system. With the help of pair programming and dynamic pairing, everybody in the team gets a chance to work on the majority of the parts of the system. In this model, not everyone knows equally all the parts of the system, but everyone knows something about everything. Shared responsibility for the entire system by each team member brings a sense of ownership and promotes a teamwork culture within the team. Collective ownership reduces the dependency on any individual to a larger extent and can encounter the impact of attrition within the team effectively.

Conclusion

Attrition is not good and cause adversities for every team but an inevitable phenomenon that team must manage and encounter its impact. In software development industry, attrition has deeper impact irrespective of being plethora of processes and risk mitigation techniques in place because knowledge and intellect that an individual carries and develops during the software development process cannot be transferred to someone else very easily.

However, if we transform our way of working and adhere to the development methodology which focuses on reducing the dependency on any individual and ease in making new members of the team productive in lesser time, the impact of attrition can be encountered effectively. XP embraces the practices which are based on the techniques proven over decades and makes sure that the practices support each other to the greatest possible degree. Practices recommended by XP like collective ownership, pair programming, on-site customer, and metaphor help to reduce the dependency on any individual in the project and helps in encountering the impact of attrition within the team.

About the Author

Nilesh Dubey is Project Manager for AI/ML platform Deployment & Support. He has 15+ years of experience with expertise in Java/J2EE, Service Oriented Architecture (SOA), and Microservice Architecture solutions. He is an agile practitioner with experience in methodologies such as Scrum, Kanban, and Extreme Programming (XP) using Test Driven Development (TDD), pair programming practices, and SAFe framework. He is a Certified Scrum Master