CHALLENGES IN TDD

One major issue with TDD is that it takes time to make it understandable to everyone on the team, thus many engineers only begin to reap its benefits much later. This means that if you want to introduce TDD into a company recently, it will take a lot of time to get everyone on board, especially certain developers who weren't familiar with its usability.

Before doing a TDD becomes second nature, some teams require continuous reminders of the advantages of their work. This is because they don't fully understand the advantages of adopting the TDD methodology. Jim Buchan et al. also state that it is challenging to learn TDD, particularly if you are a developer without prior TDD expertise.[1]

The fact that the project leader bases his explanation of TDD's advantages solely on his personal experience without taking into account the opinions of other team members is another obstacle to the adoption of TDD procedures. Once more, it took the project team close to a year of TDD use for them to integrate and internalize it as well as become aware of the gains in productivity and code quality.

It takes new team members several months to shift their perspective on how they were operating and how TDD should be used. Some developers require a long time to understand how TDD works because they have no prior experience implementing and developing systems utilizing alternative development approaches, such as agile. The project team may occasionally experience disruption from a new team member who has no prior experience with TDD. His presence might also lead to disagreement and reduce morale.

The project team should be proficient in refactoring and creating high-quality tests in order to use TDD. They must continue to develop these two abilities in order to adhere to the TDD method, which necessitates more formal training and can increase productivity.

Once more, upper management occasionally misinterprets the TDD process when they claim that developers spend more time creating tests than working on functional code. And this is accurate because the team spends more time today providing functionality than they did in the past when they were using non-TDD approaches.

Writing tests for an existing system, refactoring application code, refactoring tests, determining areas of the application that can be extended in order to choose a test, deciding on areas of the application that can be extended in order to choose a test, etc. are all examples of writing tests for an existing system. Learning TDD methods and implementing them, however, can get boring for some developers.[2]

References

[1] J. Buchan, L. Li, and S. G. MacDonell, “Causal factors, benefits and challenges of test-driven development: Practitioner perceptions,” *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, pp. 405–413, 2011, doi: 10.1109/APSEC.2011.44.

[2] R. Mugridge, “Challenges in Teaching Test Driven Development.”