According to Wikipedia, “Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes a failing automated test case that defines a desired improvement or new function, then produces code to pass that test and finally refactors the new code to acceptable standards.”

Understanding TDD
TDD has become an integral part of Agile development methodology. A typical test driven development model cycle consists of:

1. Writing a Test: A unit test (manual or automated, preferably automated) is first written to exercise the functionality that is targeted for development. Before writing the test, the developer is responsible for understanding the requirements well. A unit test shall also contain assertions to confirm the pass/fail criteria of the unit test.

2. Run to fail / make it compile: Since the feature is yet to be implemented, the unit test that was written in Step 1 is bound to fail. This step is essentially a validation step for the unit test written, as the test shouldn’t pass even if there is no code written for it. Often unit tests are automated and there are chances that the tests fail because of syntax or compilation errors. Sanitization of the tests by removing these errors is also an essential part of this step.

3. Implementing the (complete /partial) functionality: This step involves developing the part of the functionality for which the unit test is written and will be validated.

4. Making tests to PASS: Once the unit tests for the developed code have passed, the developer derives confidence that the code fulfills the requirements.

5. Code refactoring: The unit tests might have passed, but code refactoring may still be required for reasons including handling errors elegantly, reporting the results in the required format, or carving a subroutine out of the written code for re-usability.

6. Repeating the cycle (improving the scope): The unit test/set of unit tests is/are refactored to cater to new functionality or push towards completion of the functionality (if only part of the functionality is developed in first cycle). The process of continuous integration would ensure that developers can revert to older checkpoints in case the newly developed code doesn’t PASS the new unit test.

One picture speaks 1000 words! Here’s a block diagram representation of a TDD cycle.
Challenges with acceptance of TDD in Agile

TDD is a great process to comply with, but has challenges in real world development environments.

Element of change in Agile:

In Agile, there is a possibility of features getting removed during customer interactions at the end of every development cycle. For example, a developer might have spent time developing automated unit tests for a feature of a web application, and in the next cycle, the feature may not exist or could be drastically changed.

Time constraints:

With a lot more to achieve in less time, writing unit tests can become an overhead when compared to the traditional development model where unit tests development is usually a one-time activity.

Challenges to traditional project planning

Traditional project planning might not consider efforts estimation for iterative unit tests development. For instance, if developing a feature is a metric against which developers are measured, the purpose of writing unit tests in TDD might get lost.

Change in perspective for developing unit tests:

Writing unit tests in TDD requires know-how and background in the development and testing domains. A professional has to be creative in thinking of new tests (motivated with run-to-fail), and should be proficient in developing unit test code, too.

Maintenance of unit tests:

At times, it is cumbersome for developers to develop and maintain unit tests. It requires time and effort, especially if the unit test needs to be re-used for testing an ever-evolving code base.

Environment:

Setting up the correct environment for testing becomes imperative in TDD, as unit tests are validated against the developed code. Consider a case of setting up a web application that requires installing a Database or Web Server. Bringing up such an environment demands intensive effort.

Role of automation in TDD

Interestingly, automation can play a big role in the wider adoption of TDD in Agile teams. We would consider automation’s role from two different perspectives:

Effort of automation engineers

Automation engineers essentially perform the role of ‘software development engineer in test’. Not only are they aware of development practices, but they also possess a test-to-break attitude. Developing automated unit test can be shared between development and automation engineers. This would reduce the load on the development team, and also give added value for automation teams, as they get a first-hand understanding of the feature set.

Features of automation frameworks

Rich feature sets provided by the automation framework would essentially reduce the effort put in by the development teams in the TDD workflow. Here are some of the aspects of automation frameworks that can pay rich dividends:

- Ease of tracking: Unit tests would be stored in a central repository (part of the automation framework) with all the development team members submitting their unit tests to it. Tests would be stored in hierarchical folder structure based on the product features and its components. With this, viewing and tracking of unit tests within and across teams would be smoother.
- Traceability of unit tests: Automation frameworks can ensure that each product requirement has a unit test associated with it. This ensures that all requirements are developed as part of the TDD process, thus avoiding development slippages.
- Improving the development and review process: Automation infrastructure can facilitate tracking of all requirements by associating them with a developer and reviewer(s). This would ensure that development and review processes are organized.
- Unit test execution: A good automation framework ensures quick running of automated unit tests. The tests could be executed selectively for a component, set of features or the product itself.
- Reporting of test execution results: Results of the automated unit test for a component/ feature would be sent to the respective developer; this ensures quick reporting and cuts short the response time in refactoring unit tests from the developer.
- Automation infrastructure components: Automation frameworks could facilitate:
  - Cross-platform testing
  - Compatibility testing
**Conclusion**

In this article, the authors introduced the concept of Test Driven Development (TDD) and the steps involved in its implementation in the development workflow. The article also discussed the challenges the development teams face while working with TDD. The authors emphasize the role of automation engineers & automation frameworks in easing out the process of employing TDD in the development process and share tips on building automation frameworks with the help of a pictorial representation.

**Suggested Automation Framework**

As discussed in the previous section, automation frameworks can play a crucial role in simplifying the development workflow. An automated framework for such a purpose can be developed along the following lines:

![Diagram of Suggested Automation Framework]

- **Unit Test Repository**
- **Common/Application Libraries**
- **Test Runner**
- **Reporting and monitoring infrastructure**
- **Automation Infrastructure for:**
  - Cross-Platform testing
  - Compatibility testing

**References**

- Wikipedia - the free encyclopedia, Article on ‘Test Driven Development’.
- Parasoft, Article on ‘ALM Best Practices’.

---

**About the author**

**Chetan Giridhar**

(Chetan Giridhar has more than 5 years of experience working as a software engineer in research and product organizations. Chetan is a technology enthusiast and runs a website TechnoBeans (http://technobeans.com/) that he updates with tools developed, books, articles and publications written by him. You can reach him at cjgiridhar@gmail.com.)

**Vishal Kanaujia**

is a Member Technical Staff at NetApp, Bengaluru, India. He has written articles for “Linux-For-You” magazine on topics including virtual machines, Android OS, and Python.

He has interests in compiler development, performance tuning, and algorithms. You can reach him at vishalkanaujia@gmail.com.