MAHANTESH C SAJJAN

TECHNICAL

BDD ON LEGACY CODE IN REAL TIME PROJECT

This paper in brief explains about Behavior Driven Development and how it can be used in a project having legacy code in real time. In general, we have seen people having reluctance in using BDD in real time project. Yes, there are challenges using BDD in real time projects and I have explained here some of the tips that could be used to overcome these challenges.

This paper is mainly based on my experience in following BDD in my project having legacy code and tight schedule.

INTRODUCTION:

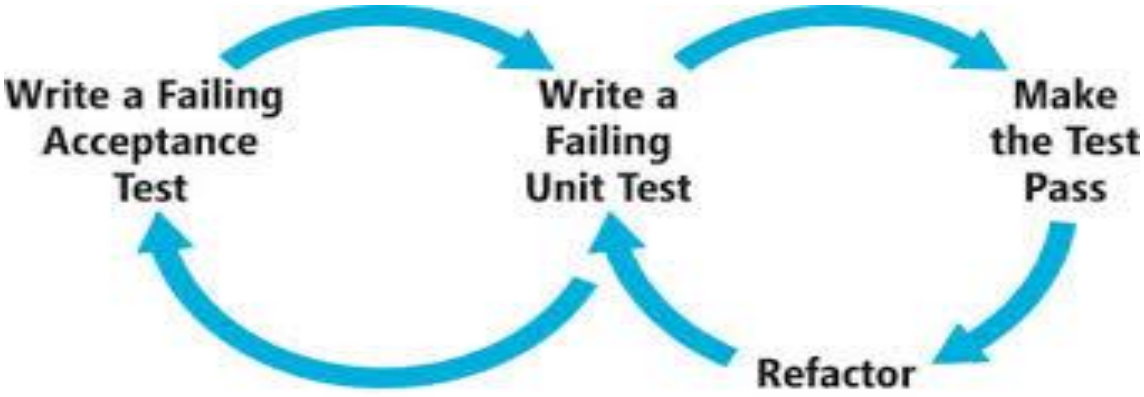
Once, I was taking a session on BDD to my colleagues and in the session, main challenge everyone had was how it could be used in the existing project. In real time, we have challenges like legacy code, tight dead line, and new development process and not sure about success or gain in the project.

In this paper I will explain about BDD and while explaining, I will mention about some the tips that could be used to solve challenges in real time projects.

DETAIL SECTION:

WHAT IS BDD?

Behavior driven development is a project coding methodology which is driven by requirements. Developers will first collect the requirement from the users in a specific format (gherkin language format) understood by test frameworks. Test frameworks converts these requirement in to test methods and developer start development with these test methods, write the test code and then the production code.



Steps:

1. Write the requirement in feature file using gherkin format
2. Convert the feature file in to Acceptance test method template using testing tools
3. Write acceptance test code which fails
4. Acceptance test will have multiple unit tests. Write unit test method
5. Write unit test code which fails
6. Write the production code to make unit test method success using TDD (Test Driven Development)
7. Refactor the production code for better design and quality
8. Re-run the unit test method for success
9. Re-run the acceptance test method which may fail
10. Change the production code to make acceptance test success
11. Re-run the acceptance test method for success
12. Continue the cycle from 5 to 11 to cover all unit tests
13. Continue the cycle from 3 to 12 to cover all scenarios present in feature file

Advantages of BDD:

Based on my experience in using this methodology in my project, I found following are the main advantages

1. Writing requirement in gherkin language format makes requirement more clear and specific. Number of hidden assumptions will reduce, hence less last moment surprises.
2. Functional and unit test time will be reduced due to automated unit tests.
3. Developer will be more confident to refactor the bad code as there are automated unit test cases and hence quality of the code gradually improves
4. Design of the project will become lean as developers think only in terms of requirements.
5. Number of production issue will reduce due to automated solid test cases. Hence technical support time and defect fixing time of the project will reduce.
6. Frequency of production deliveries could be easily increased due to automated test cases and hence required featured given to end users quickly and efficiency of end user increases.
7. Ultimately all the above advantages reduces the maintenance cost of the project and increase the efficiency of end users.

Abbreviations and Terminology:

Following are the terminology used in the paper. Detail explanations about these terminology could be found in net. Please go through below terminology before reading next sections.

TDD (Test Driven Development): TDD is same as BDD which goes at unit test level. BDD is for acceptance and TDD is for unit test.

Gherkin Language: Gherkin language is used to write the user stories (requirements) in language understood by the test frameworks.

Solid Design Principles: Basic principles of object orient programming like, The Single Responsibility Principle, The Open Close Principle, The Liskov Substitution Principle (Inheritance), The Interface Segregation Principle (Low coupling), The Dependency Inversion Principle

Epic: Epic is the term user represent high level of requirement. Epics are often used as placeholders for new ideas that have not been thought out fully or whose full elaboration has been deferred until actually needed

BA: Business Analyst

YAGNI Principle: You're NOT gonna need it. Do not implement the code until it is needed.

Brittle Tests: Tests which are failing regularly in continuous integration and becoming maintenance heck.

Lean design: Design only for requirement. YAGNI principle and lean design go hand in hand. You are not going to develop which is not needed for the current requirement. Do not generalize and make the design heavy weight in advance.

Requirement (User story):

In BDD, requirements are written in Feature file using Gherkin language. Gherkin is the language that Test frameworks understands. It is a Business Readable, Domain Specific Language that lets you describe software's behavior. It is written using "Given, When, Then, And, But" words.

Example:
Scenario: Buy last coffee
Given there are 1 coffees left in the machine
And I have deposited 1 dollar
When I press the coffee button
Then I should be served a coffee

Spend a day to understand about gherkin language either by training, reading books or internet. Lot of materials available in the net. Note, this paper is more about BDD tips for legacy code and not about BDD in detail.

My Experience, Tips and Suggestions:

1. Do not use feature file to write Epics or High level requirements. It will complicate the process. Use it only for writing smaller user stories.
2. Feature file should not be technical and should be understood by the end users. It could be used as requirement document.
3. Use scenario outline (tabular format), pre-hooks and post-hooks for writing scenarios as it is well maintainable and well readable.
4. Writing feature file is an Art where we need to convert complex scenarios to simple "Given When Then" syntax. It will be gradually improved by writing more scenarios.
5. Beginners: Developer should always start writing first scenario and do not write all scenario in the initial step. Complete the development for first scenario in BDD way. Take next scenario and complete the development. Refactor the feature file for better maintainability using the tabular format, pre-hooks and post-hooks, if needed.
6. Initial scenarios of user story should be written by developers and when scenario refactoring is completed, BA and developers could co-ordinate to writes remaining scenarios.

Coding:

Coding has two sections, one is test code and other is production code. Development always starts with test code and moves to production code.

TEST CODE:

Note that, Test code should not be deployed to production. Using third party tools, feature file is converted into test code. Tools like Cucumber, Specflow are available in the market to convert feature file into test code. Cucumber is used for Ruby, Java languages and Specflow is used for .NET language. Generated test code is a template having only test method names associated with scenario.

Developers need to fill the content of test method using BDD way. That is by

1. Write a failing acceptance test code in the test method.
2. Create a unit test class and write a failure unit test method
3. Write the production code using TDD for unit test case to success
4. Make the unit test method success
5. Refactor the developed code for better design and quality
6. Execute the unit test code
7. Execute the acceptance test code
8. Continue the cycle for all scenarios.

My Experience, Tips and Suggestions:

1. Test code is important component of TDD and BDD. Test project code is equally important as production code.
2. Test code should be well maintainable and re-usable for multiple acceptance test cases and unit test cases. Hence always good to have common test framework specific to the project.
3. To save time, do not over design test framework in advance. Follow lean design and YAGNI principle. That is, is should be developed only for the current requirements and refactored for well maintainable and removing the duplicate code.
4. Test code should not be Brittle and should not become maintenance heck.
5. Using third party tools to save time and to make our life easy. Testdirven.Net or Re-Sharper could be used in .NET for better integration of Feature file, Test code and Production code.
6. If time is the constraint, in each release, at least have 2 to 3 scenarios written using gherkin language and developed using BDD. By this time, code design for the requirement would have matured. Writing requirement in gherkin language will change our thinking perspective also.
7. As the BDD process gets matured in the project, we will have more common re-usable test code and development life cycle time improves. Hence have philosophy to have some scenarios developed in BDD way for each release.

PRODUCTION CODE:

Production code is the only code which is deployed to production and executed. It is the code which make all test cases success and satisfies the user requirements. In legacy applications, there is already

existing production code. Whenever a new enhancement comes or a defect need to be fixed, then it only needs to be changed.

Biggest challenge we have in real time project is the Time. BDD takes time and especially in the beginning it takes a lot. We can handle this challenge in a better way following some tips.

My Experience, Tips and Suggestions:

1. In beginning, do not plan for drastic change in the development process. This will be more risky as we are new to the process.
2. Technically, do not try to implement BDD as given in books on legacy code, as it is challenging and stressful. Process needs some technical fine tuning as per the project
3. On legacy code, while doing TDD do not be purist. Most of the legacy code is tightly coupled with external resources like Database. It is tough to mock it. During such scenarios, use test DBs or NULL DBs
4. If team size is big, start with 2 developers on BDD Process to reduce the Risk. But we need to start with developers who are really passionate about it, go to extra mile for it. Later they can share the knowledge to whole team.
5. To start BDD implementation, target very simple requirement or even some defects fixing. It may look silly, as we feel we have more test code for just one line production code fix. Who knew that, first house size calculator for simple addition and subtraction will be useful for future computers. Do not worry, it will be useful.
6. Have a philosophy that, before each production release, I will have some section of requirement developed using BDD process. Definitely something is better that nothing. As we go on gaining the experience, we will have more requirement done using BDD process.
7. Always start development from Out to In. This is one of the main reason BDD started. That is from Future File → Acceptance Test → Unit test → Production code. Never break this cycle, else BDD process will complicate.
8. In the beginning, we are not sure how much useful will be BDD process in the project. Yes that is fact. BDD is the process which is well proven on many complex applications and running in production. Hence we should start with positive note on it. Definitely we will come to know, it success or advantage once we complete some major requirement in BDD way. We need to have patience until that time and I found that in my project it was very useful.
9. Learning curve: Learning any new technical process will take time and energy. Even, when we move to a new project, it will take time and energy to understand the new business domain and technical design. This is rule of Nature. Most important thing is passion and interest. If we have both of them, we can overcome any challenges. For a week or two spend extra time and energy to understand the BDD Process and Gherkin language. This will be useful anytime until you are in the software industry.

REFERENCES:

Gherkin Language: http://docs.behat.org/guides/1.gherkin.html

The RSpec Book: Behavior-Driven development with RSpec, Cucumber, and Friends