Model driven development vs test driven development computer science

Model driven development is a top-down, traditional approach that has been around for a long time. Test Driven Development (TDD) is a bottom-up, new approach in the sense that it has picked up its value in the recent years. The focus of MDD as the name implies is on the models rather than on code that guides the development of software whereas focus of TDD is on the tests. There have been several changes to MDD since the beginning to deal with the changing environment and complexities raised by projects. TDD started as a part of extreme programming (XP) methodology and has gained popularity among many developers as a separate practice itself. Traditional, hard cored MDD developers are accustomed to developing software that seems more natural to them considering it to be an Engineering approach. TDD developers approach the problem in a totally different perspective concentrating on customer views of the system. There are advantages and disadvantages to both approaches. For a high quality software product that meets the goals of the organization, it is important to understand and apply the most suitable methodology while making certain trade-offs if necessary. The best approach to follow depends on several factors including the type of organization, type of project, and experience the developers have. This paper is aimed at explaining Model driven and Test Driven development along with case study and observations.

1. Model Driven Development or MDD.

Model can be precisely defined as aˆ? a description of a system from a particular perspective, omitting irrelevant detail so that the characteristics of interest are seen more clearlyaˆ?. (Source: Patterns: Model-Driven Development Using IBM Rational Software Architect)

Consider an example in which there is a pressing business need such as automation of certain processes and understanding the data collected in the business. In this situation a Business exists and there are people who are performing certain business activities in certain way. These are the people who are thinking of having a tool or a software product to help them increase their productivity and their functionality. These people who are called Domain Experts or Subject Matter experts or Business Experts know what they want. However, they currently do not have any software product that will help them to meet their needs. In this situation Domain Experts are the driving force. They do things in a certain way and expect the software product developed for them meet their needs should also do things in that manner. However, they do not know how a new software product will look like and how it will help them to meet their needs. People who develop this software product understand the needs, but they may miss the intricacies of the desired end product. Building a MODEL will help alleviate this situation. A MODEL can be a diagram, an animation or a presentation. If the Software that has to be developed is very small or involves 1 or 2 people, then the process need not be so much structured, even though it helps. However, while developing a large product which will involve tens or hundreds of people having different expertise, model will help them to accomplish this easily.

A new product development building a Model is absolutely necessary. This is analogous to a civil Engineer creating a model of Building before it is actually built.

Model Driven Development (MDD) is a methodology that focuses on designing models at an abstract level without considering the implementation details that guide the development process, focusing on one part of the system to manage complexity. Models help to analyze the problem and visualize the process. Most commonly used language for this purpose is Unified Modeling Language (UML). It establishes certain vocabulary and structure so as to understand and communicate between various developers. Artifacts are produced from the models, so the models are machine-readable, which is an important aspect of MDD.

MDD is also called as aˆ? Model Driven Engineeringaˆ?. An implementation of MDD is given by Object Management Group (OMG)'s Model Driven Architecture (MDA). Focus of MDA is on forward engineering, i. e. creating code from modeling diagrams. Apart from MDA, there are also other views of MDD such as Domain Oriented Programming, Agile Model Driven Development, etc.

MDA uses platform-independent model (PIM) to describe the system functionality as the first step. Platform Independent Model (PIM) is a generic way of representing the end view of the product without considering any technology or platforms going to be used. This model just transforms the Business requirements into a Model. This format does not contain any particular software code but shows the end state of the product in one of the Model development modes such as Diagrams, Flow Charts, and Animations etc. Based on the PIM and considering some economic conditions or availability, a proper platform is selected to develop the software. A model is then converted from PIM to platform specific model or PSM. . This contains software code at a higher level.

Platform is a loose word in this context. This can be referred to type of hardware, software, operating system, programming language or combination of any of these put together. That is why choosing a platform is related to economic viability also.

PSM can be thought of Developer's model where as PIM can be thought of Domain Experts' or End users' model.

UML representation using IBM's Rational Rose can be considered as PIM. Whereas, tools like IBM WebSphere Studio and Borland Together/J are used for developing PSM. In these, we have CODE VIEW and MODEL VIEW. These two views are synchronized together. When code is changed in Code View, it is reflected in the Model View and vice versa.

The models may contain fixed and variable data, business requirements, presentation elements (forms, reports). If one of these elements changes, they can be incorporated into the model to see the changes that occur due to interdependencies. Based on the affects, all the other depending code can be changed accordingly to accommodate these changes.

Models also describe what is called the Gap Analysis. Models are first built with an Ideal situation, but in reality the business processes and other aspects may be different. Therefore, the models will show the difference between the Real Life situation and the End State Ideal situations. This will help the Business units to define the changes in Business Practices or the

Developers to meet the Business needs and bridge the gap. A well defined Structured System Analysis and Design (SSAD) is required to develop software based on Models. In this process, the design (Model) is taken as the basis to start with the development and as the development process is advanced, it is analyzed with the design and tested against the Model and re-visited to meet the requirements.

Hence, software development using MDD approach requires Domain (Business) experts, software developers, Team leaders and Integration experts. Someone should also be able to play a Liaison role to understand both sides of the table – Business side and Software side. This person will help each of the expert groups to understand the needs of other groups and will bridge the gap. Number of iterations in the project development process will be reduced drastically if this Liaison person has good knowledge and is an expert on both sides.

MDD is used with many development processes such as waterfall model, iterative model, spiral model, etc. In MDD approach, a model plays an important role and forms the basis or driving mechanism to develop a software product.

Test Driven Development:

Test Driven Development (TDD) started its roots in Extreme Programming (XP) approach. Later on, it became a popular method by itself. TDD was originally called aˆ? Test First Programmingaˆ?. TDD can best be described in three words as aˆ? Red-Green-Refactoraˆ? (source: Kent Beck). Simple essence of TDD is to write tests before the code is written. First, unit tests are written from the requirements. These tests will definitely fail because the code for it has not been written. In order to write the tests, it is necessary for the developer to understand the requirements well. Then, code that implements the test cases is written. The code that is written should be just enough so that the tests pass, no more, no less. This means that no prediction about the future story must be made. Test driven approach is aˆ? then & thereaˆ? approach. This means that code is written at that time from the user story requirements without making any assumptions or predictions about future. After writing the code, the tests are run and seen if they succeed. If they do, then programmer can be assured that the requirements were met. After this, a process called refactoring is done. This refers to improving the quality of code and removing any duplication in the code. If the design is changed for the better, the developer can be sure he is not breaking any functionality by running the tests again. This process is repeated for the test cases that follow. This process is shown as a flowchart below:

Source: http://en. wikipedia. org/wiki/Test-driven_development

There are different issues to consider in this process. Test cases are written taking small steps at a time, such as implementation of one method. It is important to know the size of the test case and when the test case exceeds its limit of functionality to test. A test case contains the following: condition that specifies the system’s state, an event that is to be tested, and finally system’s state after the event has occurred. Almost every language has associated tools for writing these tests. In general, they are XUnit tests available for each language. For example, java has JUnit, C++ has cppUnit, . Net has NUnit, etc.

The amount of designing that has to be done in TDD depends on developer. In Extreme Programming, no designing is done, directly jumping to test cases. However, some developers prefer to spend some time on design. Too much time should not be spent on the designs and deciding on that right amount of time to spend on it comes with experience.

As suggested by Dave Chaplin, it takes almost a year for a good developer to learn most of the techniques in TDD. He divides the learning process in three stages. First stage would take three months to master writing the tests correctly. TDD is a totally different approach to take in developing software and most developers believe that hardest part about it is getting used to it and thinking in that direction. It takes another six months to learn about Mock objects. Last would be to be able to draw UML diagrams in a TDD perspective. This takes about three months. Those that become familiar with TDD find many advantages in it. These advantages are explained later in the paper.

Pair programming is considered one of the best ways to develop a program using TDD. This is because another person can make sure you are going in right path. It is hard to make developers believe that this approach works. Also, management believes that it is a waste of money to make two people work on one feature while they can work on different features.

Through test driven development, the focus is on customer’s requirements. TDD is now part of many other methodologies, such as Scrum, Agile Unified

Process (AUP), and Rational Unified Process (AUP). TDD gives confidence to the developer and produces enthusiasm as they can see parts of the program coming together when they run the tests and see them pass.

Case Study:

Results of TDD and MDD are seen more effective by example. Therefore, I consider a case in which a Software Engineering class was given a choice of either doing MDD or TDD project. The projects were done for the same problem using different approaches. It was a calendar program that consisted of certain functionalities to fulfill. For TDD people, six user stories were given one after another without knowing what the next user story is. MDD people were given a problem definition and they were to submit GUI, design, code, tests at regular intervals. The functionalities that had to be implemented included finding the following: next date, previous date, zodiac sign, day of the week, next Friday the 13th, number of shopping days left until Christmas. These were each given as a user stories to TDD people. From the results of these, the following statistics were made:

Model Driven Development approach results

CriteriaMDD

User 1

User 2

User 3

User 4

User 5

User 6

Time To Code (hrs)

8

52

89

8. 67

11

17

Time to Test (hrs)

2

15

13

3

3. 3

2

NCSS (non comment source statements)

275

600

692

499

280

Number of Test Cases

109

142

51

Technology Used

VBA

Java

Java (using Eclipse)

C#. NET & VS2008

C# & VS Express

C#

Decision Complexity

79

59

Referential Complexity

26

52

Cyclomatic Complexity

105

111

83

GUI

yes

yes

yes

yes

yes

yes

Test Driven Development approach results

CriteriaTDD

User 7

User 8

User 9

User 10

User 11

User 12

User 13

User 14

User 15

User 16

User 17

User 18

Time to code

16. 5

22

17

33

28

13. 5

33

19. 5

33

15

28

12. 75

NCSS(non comment source Statements)

349

397

276

654

240

233

1095

279

196

298

328

277

Test cases

150

84

124

70

107

247

112

88

262

56

889

128

Technology

C#. NET & VS2008

Java

Java

Java

VB. Net Express

C

VB . Net

VBA

Java

Java

VB. NET

Java

Decisional Complexity

106

66

76

76

97

115

62

57

77

145

81

Referential Complexity

12

43

40

24

24

34

19

102

9

160

29

Cyclomatic Complexity

118

109

117

100

65

121

149

81

159

86

115

110

GUI

no

no

no

no

yes

no

no

no

no

no

No

no

Number of Times Refactored

none

1

2

none

5

none

5

4

6

1

1

3

Observations:

Even though there were almost twice as many people who did TDD as MDD, certain trends can be seen from the statistics. Since majority of the projects were developed using object oriented technology such as Java and C#, most of the observations are made based on these languages only. First and major difference that can be seen is creation of GUI. Only 1 out of 12 TDD people developed GUI while all of the six people who did through MDD developed it. One of the reasons for this is that in MDD, designing of GUI was first part of the task. In TDD, although there are tools that test a user interface, it is hard to obtain them and so user has to manually test them. Since it wasn’t part of the requirement also, many had chosen to omit it.

I noticed that neatest and well designed code came from most of MDD people. Even though there is refactoring in TDD, many had chosen not to do it. This can be seen by the statistics that 6 out of 12 people have either not done refactoring at all or did it only once. As research suggests, TDD is supposed to lead to high quality code. However, most of TDD project’s code was of less quality. This comes to the point that in order to successfully carry out TDD, experience is needed. The lack of experience that most people had in doing a TDD might have been a factor for such quality. Many of TDD people had no experience in TDD methodology including me. As suggested by Dave Chaplin, in order to carry out TDD effectively, minimum one year of learning the techniques involved in the approach is needed for a good developer.

Even though the quality and design is neater in MDD, number of lines of code was much less in TDD than MDD projects. In object oriented languages such as Java and C#, MDD projects had lines of code as 518 on an average while TDD projects had 350. This is partly due to refactoring eliminating unnecessary conditions and mostly due to the reason that since code was developed based on test cases, just about enough code that was necessary was written.

Since the whole point of TDD is to start out with test cases, TDD projects have more test cases compared to MDD projects. For object oriented languages, TDD tests were 124 on average while MDD was 100.

Although theory says that TDD results in highly cohesive and loosely coupled, but in practice, it is seldom like that. This point can be seen in this study. Most of the MDD projects were more cohesive and loosely coupled resulting in more reusable code compared to TDD. In TDD projects, it was seen that most of them had 2 or 3 classes at a maximum reducing cohesive nature and reusability. I observed that this is due to unpredictability of future user stories. MDD people can carefully plan the code such that it results in certain functions that can be used by other functions. In object oriented languages, correct amount of responsibilities and collaboration among objects in my opinion can be done more effectively if more designing is done. This is because it is much easier to see visually through the models. The reusability of the code in TDD also depends on the order of the user stories given. Most people reused day of the week code in shopping days till Christmas user story or Friday the 13th user story. Suppose that the user stories are given in a different order, then different approaches might be taken that might not result in the efficient piece of code. Refactoring tells you to remove duplications and keep the code clean, but the logic of the code will not change. In our case, if suppose number of shopping days left before Christmas was given before day of the week user story, then different approach might have been taken and the reusability of the day of the week code wouldn’t have existed. It is easier to see collaborations and responsibilities of objects when the whole picture of the problem is present. Even the small amount of reusability that existed in TDD was due to the logical order in which the user stories were given.

The time spent on the whole project for TDD is much less than MDD. Considering object oriented languages, TDD average time spent was 21 hours while MDD was 42 hours (almost double!!). TDD is proven to be faster and easier technique than MDD. In TDD, concentration is on current user story and it is faster to write test cases and code directly from user requirements. The time spent on test cases is paid off since no debugging has to be done at the end.

Time and effort can be distributed and planned in advance in MDD. However, in TDD, since the amount of time and effort for the next user stories is hard to predict, it is difficult to plan. Some user stories take less amount of time while others consume lot of time. This can be seen in the case study. Most people said that finding zodiac sign took very less time and effort compared to finding the number of shopping days till Christmas .

Pros and Cons of Using MDD approach

From research and experience, these are some of the observations that were made on the MDD approach in general.

A model in projects has following advantages –

It helps to break down the project into smaller code development pieces or modules which can be assigned to different teams. It helps each team to understand their role and how their part of the development is integrated into the whole product developmentThe project managers and team leaders will understand on how to integrate all the modules and do testing. The end user will be able to visualize the integrated end product. If enough of time is spent in building a model properly, even though it takes time and effort, it will greatly reduce the time to build the product. A properly built model will also help in reducing the iterations of testing process, thus creating the integrations much quicker. There is also an economic advantage of building the models. A model will help to present a product to the investors easily and attract investments to fund the projects.

Some disadvantages of using MDD approach –

The disadvantages of using MDD approach are not related to the approach itself, but rather it is related to the application of this approach. If MDD is not used properly, there will be extended delays in the product developments.

Some of these possible setbacks are –

Building a Model will take lot of time and resources. MDD is not always aˆ? fit for all approachaˆ?. The project and product has to be analyzed before this approach is taken. Some people visualize a Model as an abstraction layer hiding all the complications of the product development process. Too much of abstraction may be good to certain audience in the project, but it over all defeats the purpose of building a model in the first place. So, care should be taken as not to ABSTRACT too much. If proper resources, such as, proper Domain experts are not involved while building a model, then the model as well as end product will be disastrous failure. MDD should not always be thought of the end point. Always underlying approach for each module should be associated with proper testing. Failure to consider the real life situations also causes the MDD unusable. The key to the whole process of

Product development using MDD approach is having a proper liaison between Business group and IT group. Pros and Cons of Using TDD approach

Some of the advantages of using TDD are:

Since test cases are developed first, developer understands the requirements thoroughly in order to write them. The focus is on the functionality perspective of the client. TDD involves taking small steps at a time and focusing on one task at a time. Even though it consumes lot of time to write many unit test cases covering all possibilities, this makes it an advantage as it avoids debugging at later stages in the project. Finding bugs as early as possible is always beneficiary in terms of time and cost. Fixing bugs at a later stage is difficult because it is hard to determine what caused the bug. Since just enough code to pass the test is written, TDD will result in thorough testing covering all possible paths. If tests are written well, then it will produce stable code. Code is developed faster and there is working piece of code at every level, which inspires confidence and encourages the developer. Developer is aware of the progress made and can set goals to achieve a particular goal. TDD results in good design because of many factors. With experience, TDD can be a very effective method as it lets the developer think in small units of code leading to modularity and good design. This is also the result of looking at the problem from customer’s perspective to understand how it will be implemented. Refactoring also leads to producing good designs. Mocking and faking is beneficial in the sense that it will define the boundaries of the classes. This is because through mocking, you will find out what your classes should and should not know about the other class. This is the basic essence of encapsulation in any object oriented language. The test cases provide support for faster regression testing. On next iteration or anytime in future, when you add more tests, you can run all the tests to make sure that it works and if it doesn’t, it is easy to see what part of the code broke.

Some disadvantages with TDD approach are:

TDD requires commitment and supporting management. Developers should be committed and write proper tests. If the tests are deleted or changed accidentally or purposefully, it will give false impression that the code is bugs-free. Support from management is important and they have to believe that this methodology works. If either of them does not exist, then TDD approach fails. Functional tests need to be done for programs that use a database or for creating user interfaces. For such things, using TDD approach is difficult. Tests and code are written by the same person most of the time and if the developer overlooks certain things, then it will affect the code as well and may not result in what is supposed to happen. If he interprets the requirements wrong, then the tests he writes and the code that implements them will turn out to be wrong and will lead to code that will not be as efficient as it should be. Another example would be if the developer doesn’t check for certain specific inputs, then the code that needs to implement that will never be implemented. Having large number of working unit tests may build up over confidence that will lead to less concentrations on additional activities such as quality assurance evaluations. Unit tests only tell if the piece of code you wrote works. Other tests such as domain testing, integration testing, etc have to be done. Amount of coverage and details of testing that is done in TDD development cannot be reproduced at a later stage. So, these tests become an important aspect and it is necessary that they are well-written. There is no way to predict the type of user stories and it is not possible to gain a complete understanding of the system from the user stories. This leads to extensive code rewriting and refactoring. WHEN to use WHAT?

This leads to the question of when is it appropriate to use TDD and when is it appropriate to use MDD. Software development is a process of developing a product to meet a certain business need. In a well defined environment and in large houses Software Development is done using the techniques described in Software Engineering and experience gained over the years. Many factors have to be taken into consideration when deciding on the methodology to use. These include things such as type of organization, management, type of project, experience of the developers, and availability of effective tools.

Type of Project:

Software development is done under two major circumstances –

1. To develop altogether a new product –

a. A product that never existed before, but there is a need in the minds of people.

A great example of this in recent years is aˆ? Youtube’. There was no such product before, however, people had thought about having some kind of sharing videos. So, there is direct business need here except that a physical need is transformed into a product.

b. There is a pressing business need such as automation of certain processes and understanding the data collected in the business.

In this situation a Business exists and there are people who are performing certain business activities in certain way. These are the people who are thinking of having a tool or a software product to help them to increase their productivity and their functionality. These people who are called Domain Experts or Subject Matter experts or Business Experts know what they want. But they currently do not have any software product that will help them to meet their needs. In this situation Domain Experts are the driving force.

2. To augment or to modify the existing software either to improve the functionality or incorporate changed business practices or fix the bugs developed.

In this situation a software product already exists and is being used by people. However, there could be changes in situation or changes in technology which requires enhancing the product or developing altogether a new Code to meet the new technology needs and/or business needs.

In the situation 1(a) or 1(b) above, the end product does not exist and it has to be visualized first before any attempt to develop a software product is done. In situation 2 above, the product already exists and everyone knows what it looks like and what it does, but they would like to see a new functionality in it.

Building a new product can be completely based on an idea or based on a business need as described in (1a) or (1b) above. In either of the situations a

Model will facilitate everyone to understand the end state of the product. This is more so in case of a requirement coming from a Business Need.

TDD approach of development of software is best suited for the situation described as in (2) above. In this approach software code is written to meet criteria and tested thoroughly until the required criteria are met. Hence this will be a good approach to fix the issues in the existing Software products or to enhance the existing Software. These enhancements and fixes are part of usage of the Software for the business purpose and are already being used in the real world. The changes, enhancements and fixes may be required due to Change in the business practices, business rules or change in the platform or technology and to incorporate more automation or integration.

TDD approach also works well when there is a new requirement that comes just before the product is to be released. Due to time constraints, one option is to write the code and deliver it hoping that there are no defects or do TDD and see the effects of code that is changed almost immediately.

Experience counts!!!

Experience of the developers is one of the factors that affect the approach that is used. If you put a team together that consists of fresh graduates with no real-time experience, it is likely that if they use MDD, they will succeed than using TDD. This is because in order for TDD approach to work well, the developer needs experience. This experience doesn’t just mean experience using TDD methodology, but also experience in software development. This is because of the fact that many of the activities involved in TDD, you become good at them by experience rather than reading. Also, getting habituated to TDD is the most difficult and challenging part about it. This opinion is shared among many developers that are new to TDD approach. With experience and learning the techniques involved in TDD should work out good. Experience is a plus in MDD approach, but not necessarily a requirement because different levels exist in a group and they collaborate and work together more than in TDD. However, after getting used to TDD thinking, many developers feel that it is a very effective approach.

Management and Organization

Organization and management play an important role when it comes to deciding the appropriate methodology to use. If the management believes in TDD approach and believes that the amount of time spent on testing is not waste, then go with TDD approach. It would be even better to use TDD when the management believes in pair programming. In an organization, if there are lot of people with every level of experience and expertise, then it is better to go with MDD approach. On the other hand, if there are only few people working on a system or it is a small project, you might consider going for TDD approach.

Other Resources

It is also necessary to take into consideration the availability of all the resources to decide on the appropriate methodology to use. These include things such as cost, time, tools, and people available. When you are running on tight deadlines, TDD is a better approach as it is faster compared to MDD. If quality is taken into consideration, in TDD, it is important that quality of test code should be as high as the production code. When you are unsure that you will achieve such a quality, it is better to go with MDD. Even things such as details in code play an important part on the decision. If there is lot of legacy code or multithreading involved, then MDD approach is better than TDD. Important thing is that it is necessary to make certain trade offs if necessary and use the best approach possible.

Conclusion

In summary, MDD and TDD are two different approaches used in developing software. Both methodologies TDD and MDD have their advantages and disadvantages. Every developer shares a different opinion on the approaches. Both methodologies can even be combined. For example, use MDD approach to divide the solution into certain modules and choose a TDD approach on each of them. When it comes to choosing the methodology that is most useful, choose the one that most suits your need.

As our Professor (Jorgensen) said, MDD is eagle’s view of having the whole picture of the project and TDD is mice’s view on the details and both are important views. This right here summarizes the whole study of TDD and MDD.

ReferencesBooks:

Applying UML and patterns — Craig Larman

Test Driven Development by Example — Kent Beck

Websites: http://msdn. microsoft. com/en-us/library/aa964145. aspxhttp://www. theenterprisearchitect. eu/archive/2009/06/25/8-reasons-why-model-driven-development-is-dangeroushttp://www. iturls. com/English/SoftwareEngineering/SE_mod1. asphttp://www. redbooks. ibm. com/abstracts/sg247105. htmlhttp://www. dsmforum. org/events/MDD-TIF07/http://whitepapers. zdnet. com/abstract. aspx? docid= 350788http://www. novulo. com/ModelDriven. aspxhttp://www. docstoc. com/docs/2660263/Benefit-of-Model-Driven-Development-(MDD)/http://www. infoq. com/articles/mdd-misperceptions-challenges

http://en. wikipedia. org/wiki/Test-driven_developmenthttp://geekswithblogs. net/leesblog/archive/2008/04/30/the-benefits-of-test-driven-development. aspxhttp://www. developer. com/design/article. php/3622546/Test-Driven-Development-a-Portable-Methodology. htmhttp://www. byte-vision. com/TestDrivenDevelopmentArticle. aspxhttp://blog. jtimothyking. com/2006/07/11/twelve-benefits-of-writing-unit-tests-first? t= xphttp://www. itemis. com/itemis-ag/portfolio/model-driven-software-development/language= en/2630/advantageshttp://eclipse. dzone. com/articles/strategic-objectives-and-advanhttp://www. bpmnforum. net/blog27/category/model-driven/model-driven-development/

http://social. msdn. microsoft. com/forums/en-US/architecturegeneral/thread/0893f9d2-fb5e-4172-8fd4-13d1b59b667f/http://www. smartagile. com/2007/09/test-driven-development. htmlhttp://www. agile-itea. org/public/deliverables/ITEA-AGILE-D5. 2. 10_v1. 0. pdfhttp://www. agilealliance. org/system/article/file/1423/file. pdfhttp://stackoverflow. com/questions/64333/disadvantages-of-test-driven-development