

Reports In This Document

<u>1. A comparison between Behaviour-Driven, Feature-Driven & Test-Driven Development</u>	<u>2</u>
<u>2. High Cohesion, Loose Coupling, Data Hiding & Information Hiding ...</u>	<u>10</u>

A comparison between Behaviour-Driven, Feature-Driven & Test-Driven Development

Tudor Haba
2420645

Table of Contents

<i>Introduction.....</i>	<i>4</i>
<i>Behaviour driven development</i>	<i>4</i>
Benefits and Drawbacks	4
<i>Test driven development.....</i>	<i>5</i>
Benefits and Drawbacks	5
<i>Feature driven development</i>	<i>6</i>
Benefits and Drawbacks	6
<i>Comparison</i>	<i>7</i>
<i>References</i>	<i>8</i>

Introduction

In this paper I am going to explain what behaviour driven development (BDD), test driven development (TDD) and feature driven development (FDD) are while highlighting the main benefits and drawbacks of each. I will then give a detailed comparison between all of the approaches discussed and see where they are similar and where they differ from each other.

Behaviour driven development

Behaviour driven development (BDD) is a test-driven development inspired methodology. It was first introduced so that the client, software developers and testers would all use a common language that is easy to understand to communicate between them in order to avoid misinterpretations of the problem at hand [1]. The problem gets broken down into small user stories that are described as follows ‘as a [user] I want [feature] so that [why]’ [1]. This story is then understood by a wider range of people with different skills and for the developers they can easily see exactly what the client wants to achieve from a given scenario [2]. The developers can then take any user story and start implementing it avoiding any ambiguity.

Benefits and Drawbacks

The biggest benefit of BDD is that it unifies the language being used so that every feature that needs implementing can be understood by everyone working on a project. This in turn makes sure that the feature means the same thing throughout the project [1] and as BDD uses natural language to communicate it also improves collaboration between different parts of the organisation as multiple different departments can work together on the stories [3].

Automatic testing is another big benefit that BDD provides. Since BDD is written in a structured language and the brief is broken down into stories that follow a pattern, tools such as Cucumber are able to generate acceptance tests for each story [2]. The tests written by Cucumber are in a language called Gherkin. Gherkin, similar to how BDD follows a pattern, provides tests that follow a pattern and that can easily be understood by people with different skill level [3] as the tests are written using a natural language syntax.

However, BDD is difficult to work with for inexperienced developers. This is due to the fact that when they start using BDD they will have no experience with writing good stories and using the tools needed [4]. Also, because of this there is a possibility that duplication of stories starts to happen. It can then become confusing for the team what the stories are supposed to do and if they mean something different [4].

Test driven development

Test driven development (TDD) is a coding technique where the developer thinks about and writes the test cases before implementing any of the code for the functionality of the problem [5]. Due to writing the tests first the developer is thinking about what needs to be achieved and how to achieve and therefore has a better understanding of the problem. There is an expectation that every test is going to fail the first time you ran it as there is no code to implement the functionality. TDD is a short and iterative process that is broken down to writing a test case for a specific functionality of the problem, run the test, then write the code that should pass the test and finally re-run the test and check if it passed or failed [5][6]. Once the initial test has passed the developer starts the process again on a new test. This way the only focus of the programmer is on the single test and making sure the code that is being implement works towards satisfying the passing condition of the test.

Benefits and Drawbacks

One of the benefits test driven development provides is instant feedback. Since the developer is writing the test first and then writing the code, once the code is being run the developer can see the outcome of the code instantly [7]. This way they can see whether the code they implement needs further changes, if the test failed. Along with instant feedback this also provides a high code coverage, writing the tests first and the implementing a solution will make sure that most of the lines of the code are tested for [5].

TDD ensures that the code written is of high quality. This is due to the fact we are running the test frequently and ensure all tests pass every time in order to move on [7]. It's also been found that having a test first approach, which TDD has, improves the understanding the developer has of the problem they are trying to solve. They have to think about the problem and what needs to be tested first resulting in them having a better idea of what the problem means [7].

However, TDD has a steep learning curve [8] it is hard for a new team to learn how to effectively use TDD as it takes a lot of practice to use it. Due to the steep learning curve an inexperienced team would spend a lot of time at the beginning of a project thinking about possible test cases. This could have an impact on the implementation of the code and it could mean that the deadlines are not respected having spent too long writing the test cases [8].

Feature driven development

Feature driven development (FDD) is an agile methodology that focuses on breaking down the problem into features that are agreed by the client [9] and completing them in short iterations. These features should be quick to implement, and they should aim to achieve a single feature of the project. An example of this would be selecting the drop off location when booking a car [10]. The process of breaking down the problem is made up of five sub processes. First one is discussing and understanding the problem between the client and everyone on the team to make sure everyone understands the context of it. Second step being to break down the problem discussed into a list of features that will be implemented by the team. The next step is when the team looks at all the features identified, and priorities are assigned to each feature. After assigning the priorities it is then checked with the client. The higher priority features are done in the first iterations of the projects. The next step is where the team decided the complexity of each feature, and everyone gets assigned a feature they take ownership and work on. The last process is creating design packages for each class which include code inspection, unit testing and integration testing [9][11]. This process is an iterative process. According to [9] these phases are based on well known pattern called ETVX.

Benefits and Drawbacks

One of the benefits FDD provides is that it aims for high quality code that is implemented quickly. FDD is a highly adaptive solution for many use cases [9]. Due to the fact that FDD is iterative and works on short periods of time and because it produces high quality code throughout the process due to the testing that is carried out it is liked by developers and clients [12]. FDD produces visible results that are easily seen by the client. Due to the results coming out at the end of one short iteration the client can then see how the problem has been interpreted and it is easier to make any changes early in the development of the project [12][13].

FDD however relies on a big team and a lot of people with prior expertise in their subject to implement [13] which would therefore make it harder for a smaller team of developers to implement. Due to the experts needed FDD works better for larger scale projects. Due to the structure that FDD follows it also makes it hard for any change in the requirement while an iteration is in progress because of the various steps that would need to take place before applying the change [14]. Some would argue that because of this FDD breaks some of the agile principles [14].

Comparison

When looking at behaviour driven development and feature driven development, we can see that they are very similar between them. Both of them work in iterative short sprints. Both break the problem down into smaller more achievable tasks. While BDD breaks the problem down into stories, FDD breaks it down further into each individual feature. Another difference between BDD and FDD is that BDD focuses on how the system should behave based on the user stories whereas FDD focuses on delivering the features as quickly as possible. Both approaches are collaborative and involve different parts of the organisation with different skill set. They both also make sure the problem is thoroughly understood and checks against with the client throughout the process. Test driven development on the other hand does not involve the client and it focuses more on how the code is implemented.

We've also seen that all of the approaches discussed work best on different scenarios. Test driven development works well on smaller scale projects due to its complexity to begin with [8] and is harder to scale up for large scale projects. On the other hand FDD doesn't work so well on smaller scale projects because of the experts needed to implement [13] and is better suited for larger scale projects where multiple teams are working on [14].

When talking about implementing the code for the problem TDD will take a long time to get started compared to both FDD and BDD. This is due to the fact that while FDD and BDD involve planning at the beginning to understand and break down the problem into manageable tasks, TDD on the other hand just gets straight into implementing. This makes it so that at the beginning the developers are trying to understand the problem they want to write tests for.

Finally, both FDD and TDD mention testing in their approaches, one focusing entirely on testing from the beginning. TDD will have a test first approach while FDD on the other hand will have a test last approach as testing is done in the last step of the process. Similarly, BDD also has a test last approach as we generate acceptance tests based on the stories, and we only test the stories after we've implemented them. In BDD however, test cases can be automated because of the natural language syntax it is written in so we don't need a team of testers like we do in FDD [9].

References

- [1] M. Irshad, R. Britto, and K. Petersen, 'Adapting Behavior Driven Development (BDD) for large-scale software systems', *J. Syst. Softw.*, vol. 177, p. 110944, Jul. 2021, doi: 10.1016/j.jss.2021.110944.
- [2] M. S. Farooq, U. Omer, A. Ramzan, M. A. Rasheed, and Z. Atal, 'Behavior Driven Development: A Systematic Literature Review', *IEEE Access*, vol. 11, pp. 88008–88024, 2023, doi: 10.1109/ACCESS.2023.3302356.
- [3] L. Pereira, H. Sharp, C. De Souza, G. Oliveira, S. Marczak, and R. Bastos, 'Behavior-driven development benefits and challenges: reports from an industrial study', in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, Porto Portugal: ACM, May 2018, pp. 1–4. doi: 10.1145/3234152.3234167.
- [4] V. M. Arredondo-Reyes, S. Domínguez-Isidro, Á. J. Sánchez-García, and J. O. Ocharán-Hernández, 'Benefits and Challenges of the Behavior-Driven Development: A Systematic Literature Review', in *2023 11th International Conference in Software Engineering Research and Innovation (CONISOFT)*, Nov. 2023, pp. 45–54. doi: 10.1109/CONISOFT58849.2023.00016.
- [5] M. Pančur and M. Ciglarič, 'Impact of test-driven development on productivity, code and tests: A controlled experiment', *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 557–573, Jun. 2011, doi: 10.1016/j.infsof.2011.02.002.
- [6] 'Course: (DVLA) L7 - Agile Software Engineering Principles (ACCA7047) | Home'. Accessed: Dec. 23, 2024. [Online]. Available: <https://moodle.uwtsd.ac.uk/course/view.php?id=13412>
- [7] H. Erdogmus, M. Morisio, and M. Torchiano, 'On the effectiveness of the test-first approach to programming', *IEEE Trans. Softw. Eng.*, vol. 31, no. 3, pp. 226–237, Mar. 2005, doi: 10.1109/TSE.2005.37.
- [8] H. A. Ramzan, S. Ramzan, and T. Kalsum, 'Test-Driven Development (TDD) in Small Software Development Teams: Advantages and Challenges', in *2024 5th International Conference on Advancements in Computational Sciences (ICACS)*, Feb. 2024, pp. 1–5. doi: 10.1109/ICACS60934.2024.10473291.
- [9] Department of Computer Science, Virtual University of Pakistan, Z. Nawaz, S. Aftab, and F. Anwer, 'Simplified FDD Process Model', *Int. J. Mod. Educ. Comput. Sci.*, vol. 9, no. 9, pp. 53–59, Sep. 2017, doi: 10.5815/ijmecs.2017.09.06.
- [10] R. Mahdavi-Hezave and R. Ramsin, 'FDMD: Feature-Driven Methodology Development', in *2015 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, Apr. 2015, pp. 229–237. Accessed: Jan. 02, 2025. [Online]. Available: <https://ieeexplore.ieee.org/document/7320358/?arnumber=7320358>
- [11] Department of Computer Science, Punjab College, Dunyapur, Pakistan and Z. Nawaz, 'Proposal of Enhanced FDD Process Model', *Int. J. Educ. Manag. Eng.*, vol. 11, no. 4, pp. 43–50, Aug. 2021, doi: 10.5815/ijeme.2021.04.05.
- [12] 'International Journal of advanced studies in Computer Science and Engineering', vol. 4, no. 11, 2015.
- [13] S. Aftab et al., 'Using FDD for Small Project: An Empirical Case Study', *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 3, 2019, doi: 10.14569/IJACSA.2019.0100319.

- [14] S. Aftab, Z. Nawaz, F. Anwer, M. S. Bashir, M. Ahmad, and M. Anwar, 'Empirical Evaluation of Modified Agile Models', *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 6, 2018, doi: 10.14569/IJACSA.2018.090641.

High Cohesion, Loose Coupling, Data Hiding & Information Hiding

Tudor Haba
2420645

Table of Contents

<i>High cohesion</i>	12
<i>Loose Coupling</i>	12
<i>Data Hiding</i>	13
<i>Information Hiding</i>	13
<i>Conclusion</i>	14
<i>References</i>	15

High cohesion

Cohesion is a term used in object oriented (OO) programming which defines how closely related the elements from within the class are to that class. Cohesion can either be low or high [1]. High cohesion means that the class only has a single purpose, for example a player class that only contains player actions. It is easy to understand when going through it and because it only achieves a single purpose it makes it easy to test as well [1][2]. A easily testable class makes for clean code that is bug free and it behaves how the developer expects it. Also, because it is easy to understand developers aim for high cohesion because it makes it faster and clearer what the class does when multiple people are working on the same code base.

A benefit of aiming for a high cohesion program is that it makes it very easy to maintain the program. This means that further down the line as the project expands and more classes are added along with more functionality the developers are able to effortlessly meet the new requirements [1]. It makes it easier to maintain the program because the higher the cohesion the less likely you are as a developer to have and go into other classes to make changes to the code as the code that needs updating is likely encapsulated with the same class [1]. Reusability is another benefit that comes from a high cohesion program. A single purposed class can be reused more than a more complex class that implements multiple functionalities [1].

Loose Coupling

Coupling is a term used in object oriented programming to describe how closely related are classes to each other. The more related a class is to another, high coupling, would reduce how often you could reuse the class [3]. Also, the time allocated for maintaining the project would increase because now classes are more interconnected. Changing something in one class might affect something in another class thus finding and fixing everything will take a long time [2].

This is why we are aiming for loose coupling which would make it so that classes are independent from each other and they don't rely on each other to function, thus increasing the reusability as well as making it easier to find faults in case the program doesn't behave as expected [4] . Using loose coupling makes the software more stable and more maintainable[2] and it gets rid of the problem where changing something in one class will have an impact on other classes because now the code is isolated.

Data Hiding

Data hiding refers to the restriction of access provided to data. Making the data private so that it can only be accessed from within the class we are in. This concept of restricting the access of data only to that class is known as encapsulation [5].

Data hiding makes sure that no outside influence can modify the data being hidden directly or by accident. This will be eliminating any potential bugs that could occur in a program by having direct access to the data. In order for someone to modify the data we are going to use additional methods that have direct access to the data [5] [6]. The methods used to access the data are setters and getters, to set and to return the data being hidden. By having the data hidden and using methods to access the data it makes it so that we can change the functionality of the class while not having an impact on any of the other classes as long as the methods for accessing the data remain the same [7].

However, breaking down the class into methods that will access the data results in more work for the developers [8]. Some of the work they will have to do include basic code for the getters and setters and depending on the number of variables that are being hidden it could be taking them a long time.

Information Hiding

Information hiding as opposed to data hiding is more than just hiding data related to that class. It hides some of the implementation completely from the rest of the program [8][9]. It can hide some of the complex implementation by creating private methods within the function that will then be called in public methods of that class thus hiding any sensible code from the outside world [8][9]. This also makes it easy to maintain the program as if you need to make any changes to the way a method works then you can easily change the way the hidden method works without having an effect on the way the method that is shown to the rest of the program works [9]. This way the places that need changes are kept to a minimum and it avoids potential bugs creeping up.

This idea of information hiding can go further than hiding the implementation of classes in a program, you can also hide the type of an object. For the same reason as with hiding method implementation, hiding the data type an object is defined will make it easier to change down the line without affecting the project [9].

Some of the information that we might want to hide because it is likely to change in the future include complex methods, as discussed, data types, global variables, which if used correctly with data hiding should not be present and business rules. The latter being very sensitive information that should always stay hidden [9].

Conclusion

In this report I talked about what cohesion is, with a focus on high cohesion and the benefits that a high cohesion project has over a low cohesion one. I explained the concept of coupling and went into more detail on loose coupling and listed some benefits of why we aim for a loosely coupled program. I discussed data hiding and information hiding and explained how the two of them are techniques of encapsulation.

References

- [1] H. Izadkhah and M. Hooshyar, 'Class Cohesion Metrics for Software Engineering: A Critical Review'.
- [2] S. Tiwari and S. S. Rathore, 'Coupling and Cohesion Metrics for Object-Oriented Software: A Systematic Mapping Study', in *Proceedings of the 11th Innovations in Software Engineering Conference*, Hyderabad India: ACM, Feb. 2018, pp. 1–11. doi: 10.1145/3172871.3172878.
- [3] S. Husein and A. Oxley, 'A Coupling and Cohesion Metrics Suite for Object-Oriented Software', in *2009 International Conference on Computer Technology and Development*, Nov. 2009, pp. 421–425. doi: 10.1109/ICCTD.2009.209.
- [4] A. Hock-koon and M. Oussalah, 'Defining Metrics for Loose Coupling Evaluation in Service Composition', in *2010 IEEE International Conference on Services Computing*, Jul. 2010, pp. 362–369. doi: 10.1109/SCC.2010.17.
- [5] M. Fowler, 'Data access routines', *IEEE Softw.*, vol. 20, no. 6, pp. 96–98, Dec. 2003, doi: 10.1109/MS.2003.1241375.
- [6] N. M. Hassan, 'CONSTRUCTORS AND DESTRUCTUR IN OBJECT ORIENTED PROGRAMMING', *J. Eng. Sustain. Dev.*, vol. 21, no. 4, pp. 1–13, Jul. 2017.
- [7] A. Snyder, 'Encapsulation and inheritance in object-oriented programming languages', in *Conference proceedings on Object-oriented programming systems, languages and applications - OOPLSA '86*, Portland, Oregon, United States: ACM Press, 1986, pp. 38–45. doi: 10.1145/28697.28702.
- [8] B. Meyer, 'Object-Oriented Software Construction'.
- [9] S. McConnell, 'Missing in action: Information hiding', *IEEE Softw.*, vol. 13, no. 2, pp. 128, 127, Mar. 1996, doi: 10.1109/52.506472.