

Behaviour-Driven Development (BDD), Feature-Driven Development (FDD), and Test-Driven Development (TDD) - A Comparative Report

Table of Contents

Introduction	2
Overview of the Methodologies	2
Behaviour Driven Development (BDD)	2
Benefits of BDD:	2
Challenges of BDD	3
Feature Driven Development (FDD)	3
Benefits of FDD	3
Challenges of FDD	3
Test Driven Development (TDD).....	4
Benefits of TDD	4
Challenges of TDD	4
Comparison	4
Focus	4
Collaboration	5
Test Coverage	5
Project Suitability	5
Conclusion.....	6
Reference List	7

Introduction

Within software development, Behaviour-Driven Development (BDD), Feature-Driven Development (FDD), and Test-Driven Development (TDD) are methodologies within the agile process. Each approaches development testing and maintenance differently and has its own merits and use cases. Understanding the differences between these methodologies is critical in ensuring the best approach is taken for a project and benefits the outcome.

This report will highlight each methodology and its strengths and challenges within software development, giving a comparison of each in their focus, collaboration, test coverage and use cases.

Overview of the Methodologies

Behaviour Driven Development (BDD)

BDD (Behaviour-driven development) is a derivative of the TDD (test-driven development) methodology [1], which emphasises collaboration between developers, testers, and business stakeholders. Focusing on the system behaviour from an end-user perspective [2]. BDD scenarios are written in plain language, often following the Given-When-Then format, allowing non-technical stakeholders to understand the expected behaviour of the system and ensuring that the software is aligned with the business needs. These scenarios can also be used to implement behaviour tests to ensure the software is functioning correctly [1], [2].

Using a language such as Gherkin [3], users can define and write the steps expected of the software using the Given-When-Then format, which is stored as a feature. These features can then be used in combination with Cucumber [4], which validates whether the software is functioning as expected. This is an example of the automation testing possible with BDD, and it can then be expanded to other features, reducing overall complexity and duration of development.

Benefits of BDD:

Improved Communication: BDD excels in allowing non-technical stakeholders to define how the software should behave. And bridges the gap between the technical functionality and the business needs. [2]

Clear Specifications: Using plain language to define how the software should behave provides transparency for all stakeholders involved in the development of the final system.

Automated Testing: Tests can be seen as an advantageous byproduct of the BDD process using tools like Cucumber[4], where the plain language features explain the functionality that is expected of the system already act as the test criteria.

Challenges of BDD

Large Projects: Scaling BDD to larger projects can become a hindrance if not adapted correctly by all parties, such as the use of required tooling needed to work in a BDD [5].

Significant Collaboration: There is a requirement for continued collaboration within BDD which can result in less progression and may even be unfeasible in some organisations [5].

Feature Driven Development (FDD)

FDD (Feature Driven Development) is a methodology that focuses on development by delivering small incremental features that build on the overall functionality of the software. The process of completing this requires a high-level plan for the system being developed. This is then broken down into a list of the individual features that make up the system. Each feature is planned out, designed and built. This process of plan-design-build is then complete for all features. Within the build stage, the feature is also tested and implemented in the overall system. [6]

FDD was created to tackle development in large-scale projects and simplify many of the challenges encountered. By breaking down the solution into feature list it offers an excellent way of tracking progress of the project, while simplifying the planning of the overall system.

Benefits of FDD

Feature List: FDD utilises a feature list which when, well structures and clearly planned out allows quick delivery of the business-critical components of the system.

Highly Scalable: Due to the modular nature in which FFD approaches implementing features, it is well suited to large projects and large teams and can be scaled exceptionally well. [7]

Regular Delivery: the iterative process of how small features are developed, FDD allows for continuous delivery, this provides an excellent loop for feedback from stakeholders.

Challenges of FDD

Lack of Flexibility: Due to having a pre-defined list of features, changes to project requirements can be challenging to adapt to. This can create issues with existing features that need to be redefined or removed. [8]

Expert Dependant: FDD relies heavily on one person, the Lead programmer often end up acting in multiple roles such as the coordinator, mentor, and designer. This approach can lead to an increased risk of human error.

Maintainability: Long-term Maintainability can suffer in an FDD environment. This is especially true for newer developers adopting the system further down the line. [8]

Test Driven Development (TDD)

TDD (Test-driven development) is a development methodology where tests are written prior to any functional code. Its focus is on improving code quality by ensuring all new features have accompanied unit tests following the red-green-refactor process.[9]

In TDD, the process begins by writing a unit test with the expectation of it failing, the next step would be to write the minimum amount of functional code to result in the test passing, next would be to refactor the code if needed, then moving on the next test and repeating the cycle [9].

Benefits of TDD

Reliable code: TDD allows for very precise and robust code as only the code required to pass the associated test is written, once the test passes the code is refactored to ensure the minimal amount of code required remains [10].

Cost savings: while adding additional functionality to the project, you are ensuring each addition works as you intended, resolving any issues before moving on, this reduces the risk of encountering issues further into the development process [10].

Improved maintainability: due to refactoring the code at each stage, this keeps a high level of code readability, in addition each tests acts as documentation of how the code should be working. Because of these factors maintaining the code base becomes simpler and can be picked up by newer members.

Challenges of TDD

Time consuming: Due to requiring test being written upfront this can be time-consuming and burdensome at the beginning of the project. This can also lead to issues where there at deadlines required.

Limited testing: TDD does not always excel when it comes to testing some aspects of the code, as it is primarily focused on testing the expected outcome, this can create issues when unexpected issues occur.

Different mindset: TDD requires a disciplined mindset for it to be effective. This does not always work well with some projects and teams. Some developers may prefer to approach development in alternate ways, which can become an issue if the whole team does not agree.

Comparison

Focus

- BDD focuses on the expected system behaviour from the perspective of its end users. It relies on and promotes broad collaboration with all stakeholders, technical and non-technical ensuring the business needs are aligned with the technical functionality.

- FDD focuses on well defines features that is required by the user and these features drive the development. Being feature driven it emphasises delivering functionality in small iterations.
- TDD focuses on code correctness and reliability; by developing the required tests before the system functionality it allows for a modular codebase with high maintainability.

Collaboration

- BDD is highly collaborative and relies on all involved stakeholders. The use of plain language promotes understandability of the systems in a way technical and non-technical stakeholders can grasp and reach agreement on how the system should function.
- FDD follows a top-down approach where developers work with the feature owners. There is collaboration with testers and non-technical stakeholders, however, it is considerably less than the BDD method.
- TDD is the least collaborative and often only involves the developers and testers. This is due to the importance of code correctness above the business needs.

Test Coverage

- BDD is an extension of TDD with a requirement for acceptance criteria and focus on expected behaviour for the end user. These acceptance criteria allow for automation of the tests for each feature.
- FDD approaches testing in a different format where each feature is tested to ensure it is working correctly, however can often ignore lower level testing such as unit tests, and correctness of the code at an individual level.
- TDD has very high-test coverage due to all functionality having associated unit tests, however this does not necessarily result in the expected behaviour as an end user.

Project Suitability

- BDD is the ideal choice where there is frequent collaboration between development teams and business stakeholders. Especially when there is a significant requirement on acceptance criteria and user stories.
- FDD is best suited in large-scale projects where clearly defined features are required within the final solution. this works best in a top down approach but can result in issues with flexibility.
- TDD is best for largely technical teams where the project requires high quality in its code and technical correctness.

Conclusion

BDD, FDD, and TDD all have their place within an Agile working environment, each committed to the delivery of high-quality development. Each approach reaches the outcome significantly different, BDD encourages a high level of collaboration between all stakeholders resulting in a well understood end product by all parties, technical and non-technical alike. FDD encourages producing the software through incremental features and follows a detailed plan, which can result in difficulties in flexibility. TDD emphasises correctness and code quality, this minimises unexpected issues due to developing around the tests, however this can hinder non-technical stakeholders. A combination of these practices would be beneficial depending on what the software solution is trying to achieve and by the stakeholders involved in its development.

Reference List

- [1] C. Solís and X. Wang, 'A study of the characteristics of behaviour driven development', in *Proceedings - 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011*, 2011, pp. 383–387. doi: 10.1109/SEAA.2011.76.
- [2] L. P. Binamungu and S. Maro, 'Behaviour Driven Development: A Systematic Mapping Study', May 2023, [Online]. Available: <http://arxiv.org/abs/2305.05567>
- [3] 'Gherkin | Cucumber', <https://cucumber.io/docs/gherkin/>.
- [4] 'Cucumber | Cucumber', <https://cucumber.io/docs/cucumber/>.
- [5] M. Irshad, R. Britto, and K. Petersen, 'Adapting Behavior Driven Development (BDD) for large-scale software systems', *Journal of Systems and Software*, vol. 177, Jul. 2021, doi: 10.1016/j.jss.2021.110944.
- [6] C. M. Budoya, M. M. Kissake, and J. S. Mtebe, 'Instructional design enabled Agile Method using ADDIE Model and Feature Driven Development method', *Int J Educ Dev Using Inf Commun Technol*, vol. 15, no. 1, pp. 35–54, 2019, [Online]. Available: <https://www.proquest.com/scholarly-journals/instructional-design-enabled-agile-method-using/docview/2227797330/se-2?accountid=130472>
- [7] S. Ahmed, A. Razzaq, S. Zeeshan, S. Ahmed, and R. Ullah, 'Evaluation for Feature Driven Development Paradigm in Context of Architecture Design Augmentation and Perspective Implications', *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 3, 2018, doi: 10.14569/IJACSA.2018.090334.
- [8] Z. Nawaz, 'Proposal of Enhanced FDD Process Model', *International Journal of Education and Management Engineering*, vol. 11, no. 4, pp. 43–50, Aug. 2021, doi: 10.5815/ijeme.2021.04.05.
- [9] J. Cui, 'A Comparative Study on the Impact of Test-Driven Development (TDD) and Behavior-Driven Development (BDD) on Enterprise Software Delivery Effectiveness', Nov. 2024, [Online]. Available: <http://arxiv.org/abs/2411.04141>
- [10] F. Shull, G. Melnik, B. Turhan, L. Layman, M. Diep, and H. Erdogmus, 'What Do We Know about Test-Driven Development?', *IEEE Softw*, vol. 27, no. 6, pp. 16–19, Nov. 2010, doi: 10.1109/MS.2010.152.

The Importance of High Cohesion, Loose Coupling, Data Hiding, and Information Hiding in Software Development

Table of Contents

Introduction	2
High Cohesion.....	2
Definition and Explanation	2
Importance in Software Development.....	2
Loose Coupling	3
Definition and Explanation	3
Importance in Software Development.....	3
Data Hiding	3
Definition and Explanation	3
Importance in Software Development.....	3
Information Hiding	4
Definition and Explanation	4
Importance in Software Development.....	4
Conclusion	4
Reference List	6

Introduction

There are four essential methods within software development that greatly improve the quality and structure of software applications. These objectives being high cohesion, loose coupling, data hiding and information hiding. They greatly improve the understandability of the system leading to lower complexity in improving, modifying and error handling of the systems being developed.

This report aims to highlight the key benefits of each method and its importance in creating high quality software. High cohesion promotes clearly defined functionality with a high level of maintainability, loose coupling ensures modules are independent of one another allowing improvements to be made with little impact to the overall system. Data hiding increases protection for the systems data integrity, managing access to data limits the risk of unauthorised changes or errors occurring. Finally, information hiding, similarly, increases protection of the interactions within the system and obscures its functionality.

High Cohesion

Definition and Explanation

Cohesion describes how elements of a module within a software system relate and work together to perform a task. High cohesion refers to components being closely related in their functionality, and in OOP (object Orientated programming), means that the class focuses on delivering a single function or few related functions. [1]

Importance in Software Development

Clarity: Understanding a module or class becomes clearer with a high cohesion as it has a defined purpose. This leads to improved maintenance, troubleshooting and improvements or extension.

Modularity: A class with high cohesion can be easily reused within other systems, with well defined functionality of the module, it can be easily reused within different systems with minimal modification to be accessed by the wider system. [2]

Maintainability: Through clearly defining the role of a module's functionality, this allows for an improved ability for it to be maintain due to the separation of desired functionality. Furthermore, there is a minimal impact on other aspects of the software if changes are implemented. [3]

Testing: having a clearly defined purpose, highly cohesive modules are easier to test, implementing unit testing is straightforward as the behaviour of the module is well knows and understood.

Loose Coupling

Definition and Explanation

Coupling is a term used to describe the independence of various modules and how dependent they are on other aspects of the system [4]. Loose coupling is when modules have little dependency on one another. Each module should have as close to complete independence of each other, and no direct dependencies but rather interacting through various interfaces.

Importance in Software Development

Flexibility: loose coupling allows developers to improve, replace and change aspects of the system with little impact to the wider application. This increases the flexibility of the project when there are changes to requirements, or future improvements.

Maintainability: the maintainability of loosely coupled systems is increased due to the separation from various modules. This separation implies modules have well defined functionality, in turn resulting in issues being simpler to locate and correct. [5]

Testing: The lack of dependence on other modules allows for improved testability of the components, as there are fewer dependencies on other components the behaviour should be understood and well known, allowing for well-structured unit tests to be written for each component.

Concurrent development: lack of dependency from other components allows for simpler concurrent development, with each team working on separated components there are fewer risks to overlapping code, and less risk when merging completed work.

Data Hiding

Definition and Explanation

Data hiding is a term used to describe the process of limiting visibility to the internal data of an object. This is done through the use of private fields only accessible within the object, and data is accessed with the use of getter and setter methods giving controlled access to what is contained within the object. [6]

Importance in Software Development

Maintainability: The maintainability of a system utilizing data hiding becomes simpler, this is due to any changes within the object not requiring changes externally to the code required for interacting with the object. [5]

Error prevention: Data hiding helps prevent inadvertent modifications to the data, which can introduce errors within the system. By hiding the data, developers can only modify the data directly within the object which adds a level of protection against developmental errors.

Flexibility: By ensuring the access method remains in place, any modification within the object can be made with little risk to the overall system. All internal modifications of an object can be implemented without effecting the interaction to the wider system.[6]

Protected: Data hiding ensures controlled access to data contained within objects preventing unintended modification.

Information Hiding

Definition and Explanation

Information hiding covers a wider scope than data hiding along, information hiding refers to hiding both the data and implementation of the system. Focusing on what a system does rather than how it achieves this [7] promotes better quality systems, due to less focus on the interactions between modules over the output of the modules.

Importance in Software Development

Maintainability: Information hiding aids in reducing the complexity of the system by highlighting the high level functionality of the system over the low level implementation details. The reduction in complexity aids current and future developers in continued improvement and maintenance. [8]

Flexibility: By hiding the unnecessary information, objects can be worked on independently from the overall system. Reducing the risk of creating unintended errors to other components of the system while implementing improvements or modifications.

Encapsulation: When the information contained within an object is hidden, its implementation can be improved or modified with reduced impact to the overall system. [7]

Focus: Information hiding allows developers to focus on the individual module and the functionality it provides without concerns on how it achieves this. This can benefit on the time taken on the development, leading to high outputs.

Conclusion

High cohesion, loose coupling, data hiding and information hiding are vital elements of software design and provide the foundation for highly maintainable, concise and scalable software. High cohesion improves the quality of the individual modules that make up the overall system, ensuring they are easy to understand. Loose coupling promotes independence of modules reducing the risk of errors occurring within the system. Data hiding and Information hiding protects the system through controlled access and interactions.

By utilising these methods within development, developers can produce higher quality products that are easier to maintain and improve. Not only do these methods promote higher quality software, but they improve productivity of the developers themselves. When implemented correctly, it leads to reduced errors, improved performance and lowers the overall complexity of development. Additionally, there is the benefit of reduced production cost. These methods will continue to be essential in development of software.

Reference List

- [1] J. Al Dallal and L. C. Briand, 'An object-oriented high-level design-based class cohesion metric', *Inf Softw Technol*, vol. 52, no. 12, pp. 1346–1361, Dec. 2010, doi: 10.1016/j.infsof.2010.08.006.
- [2] I. Candela, G. Bavota, B. Russo, and R. Oliveto, 'Using cohesion and coupling for software remodularization: Is it enough?', *ACM Transactions on Software Engineering and Methodology*, vol. 25, no. 3, May 2016, doi: 10.1145/2928268.
- [3] S. Apel and D. Beyer, 'Feature cohesion in software product lines', in *Proceedings of the 33rd International Conference on Software Engineering*, New York, NY, USA: ACM, May 2011, pp. 421–430. doi: 10.1145/1985793.1985851.
- [4] Z. A. Alzamil, 'Software Coupling and Cohesion Model for Measuring the Quality of Software Components', *Computers, Materials and Continua*, vol. 77, no. 3, pp. 3139–3161, 2023, doi: 10.32604/cmc.2023.042711.
- [5] P. J. Kaur and S. Kaushal, 'A Fuzzy Approach for Estimating Quality of Aspect Oriented Systems', *Int J Parallel Program*, vol. 48, no. 5, pp. 850–869, Oct. 2020, doi: 10.1007/s10766-018-0618-2.
- [6] K. R. M. Leino and G. Nelson, 'Data abstraction and information hiding', *ACM Transactions on Programming Languages and Systems*, vol. 24, no. 5, pp. 491–553, Sep. 2002, doi: 10.1145/570886.570888.
- [7] D. L. Parnas, 'On the criteria to be used in decomposing systems into modules', *Commun ACM*, vol. 15, no. 12, pp. 1053–1058, Dec. 1972, doi: 10.1145/361598.361623.
- [8] G. T. Leavens and P. Muller, 'Information Hiding and Visibility in Interface Specifications', in *29th International Conference on Software Engineering (ICSE'07)*, IEEE, May 2007, pp. 385–395. doi: 10.1109/ICSE.2007.44.