

Development Methodologies: TDD, FDD, and BDD

Abdulmalik Elghedafi

27th November 2024

1 Introduction

Test-Driven Development (TDD), Feature-Driven Development (FDD), Behaviour-Driven Development (BDD) are among the three most popular methodologies used in Agile practices throughout the software development industry. All of them offer a distinct contribution in software projects. By being aware of what each of these methodologies can do, it can offer software teams to understand how it can leverage their projects by weighing the pros and cons, ultimately maximising the project outcomes to delivering the best software to a client. The aim of the report is to provide the reader with an overview of what each methodology is, a brief history and the benefits and drawbacks, as well as showcasing a comparison between each approach and finishing off with a section about the application to software engineering.

2 Test-Driven Development (TDD)

2.1 What it is and used for

TDD ((Figure: 1)) is a popular software development methodology that is used by developers to write automated test cases before any code is written. This allows the developer to then run a test, which should fail in the first instance by turning red and then re-write the test to make the software run the code to pass and make it green. TDD should be following a continuous cycle where the developer creates a test, which will check and confirm the expected behaviour, then develop as little code required to pass the test written initially. Then refactor the code, which is after the test passes, where the developer would improve the code, readability, structure and removing duplication, but without altering the execution [1].

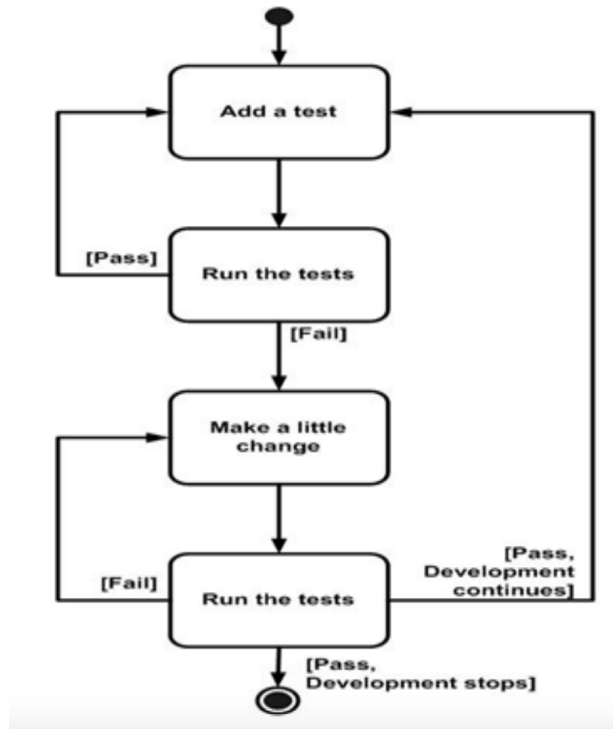


Figure 1: TDD Process Life Cycle [2]

Through studies it's been deduced that developers spent less time debugging code with TDD. It is used for writing automated tests on classes, functions or methods in code, which is known as a unit test. Testing can be done manually and performed through an automated process which can reduce manually testing each function, the tests can be written by the same developer of the code or through a software tester. When a test is created, it is involved through a process using a popular open-source framework called Junit which is for Java [1].

TDD is used for ensuring that an engineer must think about the process of writing a test on how the output should be executed before writing the functionality. This results in TDD creating cleaner, more readable, structured code, while enhancing the workflow process for the developer. It is known that TDD relies on the agile mindset which helps focus on continuous improvement [3].

2.2 Brief History

TDD was conceptualised in 2003 by Kent Black, however NASA had employed its use during the development of the Mercury project in the 1950s [4]. Extreme Programming (XP) is a key practice of TDD.

2.3 Benefits and Drawbacks

Benefits:

- Testing occurs constantly after a small bit of code is written; therefore, bugs are handled in the early stages of the development process, which reduces the need to debug code at a later point [4].
- Refactoring code to meet and pass a test, ensures that the code is being written to a high-quality standard [4].
- It does not focus on the software development principles, instead it focuses on the development progression [4].

Drawbacks:

- Programmers require a variety of skills to write test cases, therefore a tester is required to write the test, which can cause confusion to the programmer [4].
- TDD can create unsatisfactory documentation, as the test-cases are created for development only. Whereas documentation can be used in the maintenance section [4].
- TDD can take up more time than expected for complex systems, if test cases happen often [4].

3 Feature-Driven Development (FDD)

3.1 What it is and used for

FDD is based on Agile methodologies that use small sprints to build fully implemented software. FDD is built for flexibility where it's committed to quality on each section of the engineering process. FDD uses features which is critical to the development process, which allows the end-user to have any desired functionality within their software i.e. features that meet the direct needs of the end-user [2], [5]. The main goal about FDD falls onto the design and building sections [2].

The five stages of FDD ((Figure: 2)) are: develop an overall model, build a features list, plan by feature, design by feature and build by feature [2].

These five stages expand into detail. The first one is developing a model which based on the plan of the project through discussions with project managers, lead programmers through meetings [5]. The second stage consists of developing of the features list, which is then gathered to make a set of features. The third stage is setting importance for every single feature, so when a set of features are created, then it would be cross-referenced to see if it matches the needs of the business and what was set out in the requirements of the project [5].

Designing by feature is to have a detailed design process using i.e, sequence/class diagrams to represent how the feature will work. After creating the design, the feature would be developed through code based on the design structure. The development team is given a set of requirements to work on and then implement a working feature. The code is then reviewed i.e through a pull request, then performing a unit and integration test, deployed and merged into the system [5].

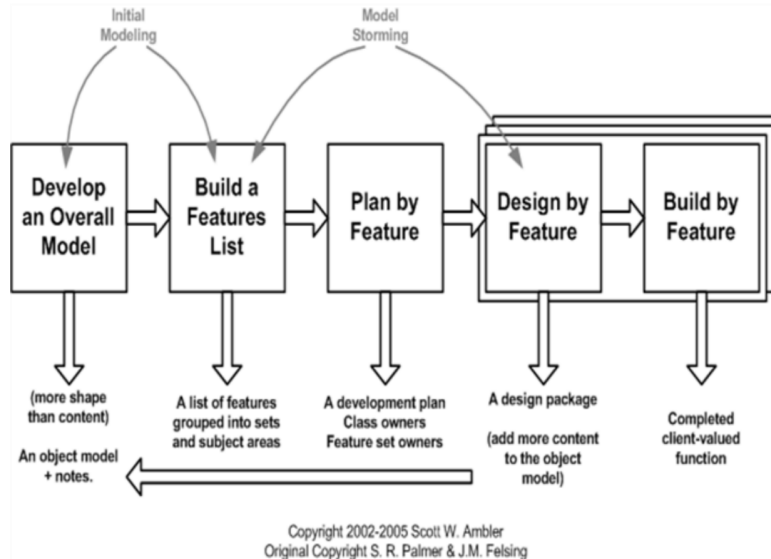


Figure 2: FDD Process Life Cycle [2], [6]

3.2 Brief History

FDD was first introduced by Jeff De Luca through the success on the Singapore project where there were multiple developments made using FDD in 1997. This is when he identified that classic models weren't great at delivering high-scale, complex projects that required large teams to work on and within deadline dates [7]. It was then in 1999, Peter Coad had partnered with Jeff De Luca to write a book about the FDD model, which talks about the concepts of features in the software industry in the Java Modelling in colour with UML book. It was then in 2002, Stephen Palmer and Mac Felsing had written a book about the practical guide to Feature Driven Development [2], [7].

3.3 Benefits and Drawbacks

Benefits:

- FDD is versatile, as it's main focus is on designing and representing viewpoints of a project [2].
- Teams concentrate on fostering growth when working on the development stage [2].
- Takes roughly one to four week sprints to get a speedy response on the feature developed [2].

Drawbacks:

- FDD is compatible for small to a moderate sized team to ramp up on projects [8].
- Focuses heavily on team-working, in person meetings and lots of client contributions [9].
- There is a lack to flexibility when it comes to changing requirements suddenly - which in turn can be a prolonging process to responding to new challenges [9].

4 Behaviour-Driven Development (BDD)

4.1 What it is and used for

BDD is part of the agile software development methodology that inspires teams of software engineers to deploy software to encourage participation between investors [10].

When using BDD to write software, usually it is started by assessing a few components that an application is vital to have. This is then transformed into examples in the template of "Given-When-Then" which helps expand to the developer what a feature should do [11], [12]. For example, Given, the user enters their bank card into the ATM and enters the pin number and has enough funds in their bank account, when the user selects withdraw, they enter the amount and press confirm, then the system should deduct the amount withdrawn from their account, eject the card from the ATM and dispense the cash.

Writing a user scenario as described, is considered standard. Approval is required from interested-parties to go ahead with the user scenarios before developing the feature [11]. Tools such as Cucumber which is common framework in software engineering, uses the gherkin language to operate the automated tests. Gherkin describes the way a system works, using English i.e. "Given-When-Then" [10], [12], [13]. If a test fails using Cucumber, then it will use a method like TDD where a tester will develop test scripts and run them until the test succeeds.

BDD is used to think about creating an application from a customer view point. By using BDD through development, it is essentially leveraging all the benefits of TDD [12], [11]. BDD is known to be built on top of TDD and is an improvement, BDD uses real-world examples to create test cases [11].

BDD reflects on the commercial and technical viewpoints by developing test scripts using business domain language (BDL) which is to create a sample similarly to English, which makes it easier to understand by anyone who is not a developer [13]. BDD prioritises on user acceptance tests, by using user acceptance criteria (UAC) and user stories, which in essence it defines how customers will interact with the system and manage requirements [13], [10], [12].

4.2 Brief History

BDD, started in 2003 by Daniel Terhorst North, where he created a substitution for writing JUnit tests, where he would focus on the behaviour of an application rather than the test cases. This was to unite non-technical and developer teams to understand the requirements, without a background in programming using the "Given-When-Then" model to capture user stories and acceptance criteria [14].

4.3 Benefits and Drawbacks

Benefits:

- Enhances teamwork with all involved on the project [13].
- Writing tests in english, ensure it's easier for non-programmers to understand the purpose and requirements of the system [13].
- Reduces misunderstandings and ambiguity, as the requirements are laid out to ensure what the user is expected to receive [13].
- Helps align user's needs, as BDD concentrates on user behaviour, which produces a bigger impact [13].
- Higher quality code, as development ensures lower costs and risk of the project [12], [13].

Drawbacks:

- BDD requires developers, business analysts and testers to work in a team, otherwise it's not unsuitable to leave it to just testers which results in an overhead expense [12].
- There is no prerequisite to learn extra knowledge to understand BDD, although many companies require users to know programming for BDD [13].
- BDD will be unsuccessful if created using an incorrect layout or without knowledge and without communicating the goals with stakeholders before implementation starts [12].

5 Comparison between approaches

Aspect	Test-Driven-Development (TDD)	Feature-Driven-Development (FDD)	Behaviour-Driven-Development (BDD)
Core Focus	Writing a test first, then develop code to pass [1]	Built to providing the highest quality on each section [4, 5]	Using business domain language to dictate the behaviour of the system [10, 11]
Primary Stakeholders	Mainly software developers as well as software testers. [1]	Software developers, project managers, lead programmers [5]	Software developers, project managers, business analysts [12]
Process Flow	Write a test case that fails in the first instance before implementation, then write the code to pass the test. [1]	Build on each stage, before moving to the next phase using small sprints. [4, 5]	Reflects on behaviour of system using user acceptance tests to manage requirements. [12, 9, 11]
Testing Approach	Writing automated / unit tests on classes, functions and methods [1]	Performed using a unit and integration test [5]	Uses Gherkin language to operate automated tests [9, 11, 12]
Flexibility	It's somewhat flexible, as it relies on ensuring the code works based on the test case. [1]	Lack of flexibility when it comes to changing requirements [8]	BDD is flexible, as you can edit the user acceptance criteria based on feedback [12, 9, 11]
Speed of Development	Can take up more time, if systems are complex [3]	Can be quick, as uses small sprints to build fully implemented software [4, 5]	Not sure, have a look online for journals
Best for	Small to medium teams [17]	Larger teams and more complex projects [18]	small teams [10]

Figure 3: Comparison between approaches

In this section ((Figure: 3)) a table delves into a comparison between the approaches of testing.

6 Application to software engineering practices

6.1 Best practices for BDD, TDD & FDD

6.2 BDD

A case study in healthcare had used BDD to implement the ATRI EDC System. BDD was helpful as it improved interaction and working in a team. It helped by setting a standard which many teams use to set a contract, to establish test requirements that are system-required. All parts of the process, are tracked and visible to all parties of the organisation. Automation simplifies the process of writing documentation and testing, allowing developers to spend time elsewhere [15].

Before a product release, project stakeholders evaluate test scenarios, ensuring they cover the original requirements of the project. However, given the advantages, there are disadvantages to BDD, when writing gherkin syntax, it failed to capture nuance and intention. Although, using gherkin syntax does help collaboration between developers and non-technical stakeholders. If a web browser i.e Google Chrome does not have up-to-date web drivers, then the test scenario can fail, which helps run the user behaviours [15].

In this case study, this highlights that BDD saved money and time to implement the ATRI EDC System. As part of the health case study, Cucumber was used to communicate how the system should behave, using DSL which in turn uses the Given-When-Then syntax [15].

While BDD offers many benefits (Figure: 4), it also comes with a few challenges. 51% had found the use of a new tool in the software industry to be challenging which can alter the team's workflow, therefore making it unconventional. While 28% found it to be difficult to learn, which can emphasise that team's require time to become familiar with the gherkin syntax, and 20% have mentioned it reduced the team's productivity which reinforces the idea that, software developers and business stakeholders, having to spend time creating user acceptance stories, which can slow down development of application features [16].

6.3 TDD

In this section a case study has been conducted between two approaches, one being TDD and another being Non-TDD. Case A outlines that TDD had reached far lower defects and lead to better client experience. Thus leading that TDD is an excellent agile methodology to use in the healthcare industry, as healthcare companies rely on data accuracy. According to the research TDD promotes the use of classes, which is less influenced than in Non-TDD methodologies. Which makes TDD easier to use, as it simplifies the software, into smaller chunks of code

BENEFITS AND CHALLENGES OF USING BDD

Benefits of BDD	Rate (%)
Software specifications are expressed in domain-specific terms, and thus can be easily understood by end users	67
Improves communication between various project stakeholders	61
Specifications can be executed to confirm correctness or reveal problematic software behaviour(s)	52
Code intention can be easily understood by maintenance developers	50
Attention is paid to validation and proper handling of data	24
Could produce better APIs since it emphasizes writing testable code	28
Other	7
Challenges of BDD	
Its use changes the team's traditional approach to software development, and that can be challenging	51
Its benefits are hard to quantify	35
It involves a steep learning curve	28
Other	21
It can lower team productivity	20

Figure 4: Benefits and Challenges of BDD: [16]

into independent modules known as modularity. This allows each component of the system to be independently coded, tested and maintained. This results in the code being readable and adjustable in the future [17]. TDD is beneficial for small to medium teams to work on software projects [18].

The study in case B outlines that the results in the Non-TDD approach were much higher in terms of bugs than TDD, prior to the product being published. This resulted in clients being frustrated as it had resulted in higher frequencies of issues using the Non-TDD approach. Whereas clients that used TDD were significantly happier, which concluded that far less bugs were encountered. As well as making TDD suitable for large scale projects [17].

6.4 FDD

It was observed that in this case study, which is based on an academic piece of work through the implementation of the empirical analysis, that FDD resulted in unsatisfactory performance in a short-term project, which indicates it is appropriate for bigger and advanced projects that are well-defined to handle the scope [8].

According to the findings, It has been identified that the Islamabad software organisation had developed their software using FDD, as a result of the group of skilled specialists with a background in Computer Science. This implies that FDD is powerful when teams navigate sophisticated tasks, while working together [8].

When it comes to managing significant quantities of requirements, this research shows that FDD is good at handling the amount of user stories (Figure: 5), which is well-suited for projects with a wide range of requirements [8].

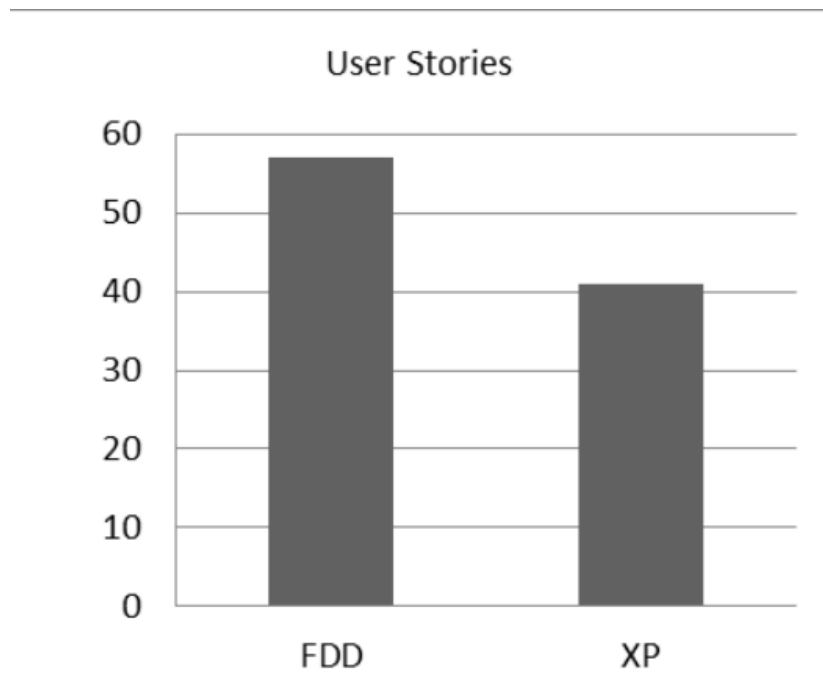


Figure 5: No of user Stories [8]

7 Conclusion

This report focuses on the three types of testing methodologies in Agile. TDD focuses on writing tests before implementation, which leads to catching bugs early on and is focused on the technical aspects, ensuring developers meet the requirements of the project. While FDD encourages developers to work in small sprints, ensuring that each section is of high quality while providing flexibility and is ideal for larger projects i.e. banks. Whereas BDD improves collaboration between developers and non-technical stakeholders. Depending on the aim of the project, there is not one methodology that beats them all, but more based on the use case of the aim of the project.

8 Detailing the importance to software development - Introduction

High cohesion, loose coupling, data hiding and information hiding are among four concepts used in the object oriented programming industry by software developers. This report explores these different concepts in more detail, where it covers the definition of what each concept is and what it's used for, as well as providing a brief history / background about how this concept was originally introduced and by who. By understanding these concepts in further detail, it can provide software engineers with crucial information on how to develop highly efficient applications and clean code.

9 High cohesion and loose coupling

9.1 What it is and used for

In the context of object oriented programming (OOP), high cohesion and loose coupling is a pivotal factor in the software engineering industry. Both of these features assess the level of power required to connect with components of the software code. For example, in object oriented application code, these two features would be strongly used in methods and objects. The primary goal when creating software application code is to design software applications to use high cohesion and loose coupling [19], [20], [21].

Coupling is the scope of which programming code from one component library depends on another component library. The difficulty of the application will rely on the relationship and dependance within software libraries [19], [21].

By enhancing the flexibility of an application, using coupling between diverse classes should be kept as low as possible. When a class is linked with other classes that are liable to be changed, it turns to being too difficult to handle, leading to flawed code and becomes difficult to perform tests. If the

class becomes unclear, then it's tough to perceive what it does from a developer standpoint. Hence ensuring why the loose coupling is preferred [19], [20], [21].

Software engineers are more favoured to use cohesion, as when it comes to designing software applications, high cohesion is linked with preferred features i.e reusability, ensuring the code is reliable, readable and durable. The use of low cohesion is not favourable to work with, as it becomes being difficult to support and software test, as well as challenging to reuse and comprehend [20], [21]. Below is a figure ((Figure: 6)) which shows the best to lowest in terms of cohesion [19].

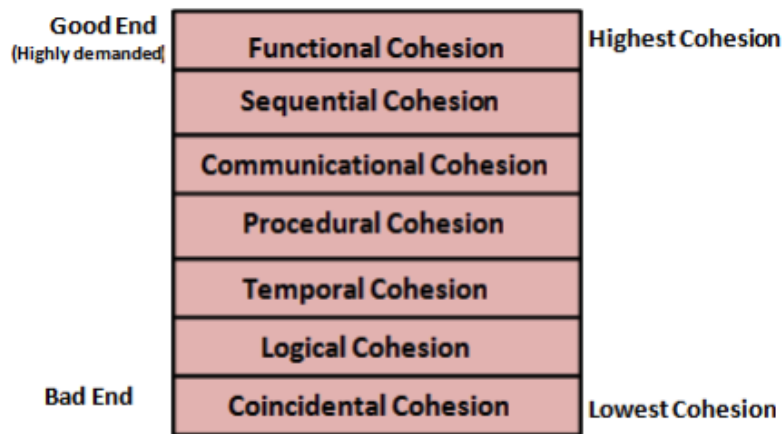


Figure 6: Degree of cohesion from good to bad [19]

9.2 Brief History

Stevens et al, was one of the initial people to present the idea of coupling in the software engineering field. The term coupling means the intensity of the connection between two software libraries [22].

10 Data hiding

10.1 What it is and used for

In OOP (Object-oriented programming), there is a software engineering approach that is commonly used to restrain access to data members, which was discussed by Cory (2014) [23].

By protecting against unexpected or expected updates, a process used is through hiding data by only allowing limited access to data to members of a class, thus protecting object coherence in OOP. Encapsulation and information hiding is alternatively another term for data hiding in OOP. To enhance the systems architecture in OOP, data hiding is used, which helps to provide reliability across application components [23].

10.2 Why is it important

Data hiding is important in software, for many reasons:

1. When restricting other software engineers from depending upon the use of the inner working details of a class for example the object properties and method functions which is the connection that is used to interact with the code needed, is the sole component that must remain stable. Which in turn means that the programmer can edit the internal implementation of a class, without having any effect on the current code that uses the class [23].
2. Data hiding helps to safe guard classes in OOP from unplanned mistakes when programming. This is because each class will hold a few or more fields that are reliant on each other which are held in a stable condition. If a developer was to modify a field directly, they may alter one field without modifying the other field that relies on it, which causes the class being left in an unreliable state [23].
3. As a result, hiding particular methods in OOP that are used by a class for private use, will stop developers from invoking a class [23].

10.3 Brief History

David Parnas initially defined the principle of data hiding back in 1972. Input and output sub-systems and application modules were frequently employed by many commercial applications through the use of modular code implementation. A book was then written to address the structured design in the book named "Designing Systems Programs", authored by Stephen Pont and Richard Gauthier [23].

In 1992, Andrew Tirkel and Charles Osborne had invented the concept of "digital watermark", which is a stamp that identifies the trademark for the

brand. This was initially introduced back in Italy during the time period of 1201 to 1300 and was created in the paper manufacturing process. Additionally watermarking had made it's way to Europe, where it was used as a way to identify the papermaker or trade guild to who produced the paper, giving a unique way of providing credit and authoring the brand behind the product. To this day, this process is still used in factories all over the world and to restrict the creations of counterfeits [23].

The concept of digital watermarking is important as it relates back to data hiding, where only the inner workings of a class are not displayed to external users of the class [23].

11 Information hiding

11.1 What it is and used for

Information hiding is a process where the user determines what area's of the code in a software application have to be disclosed to the external part of a software application as well as sections that have to be obscured externally and to have a degree of changing - which preserves the remainder of the application from being altered, if the design process is switched [24], [25].

A software library has a set of rules it uses, known as defining the modules interface. This is used, so it can provide a software engineer to define what is private and what is public to the software module as well as refine a method / utilise various ways of using a data structure, ensuring the external class method is unaffected [24], [25]. Due to this, any developer who wants to engage with the module, will require to understand what the module's goal is to do, rather than requiring the processes behind how it works [24], [23].

As long as the exact public interface is used and follows the same rules, then a software engineer can upgrade the software library with a new one. When information hiding is used in OOP to hide a software libraries internal working details, and all the procedures it goes through are handled using a intuitive interface, then it essentially means that it would ensure that external entities from causing unintended or deliberate interference by external systems, whether it was by accident or design in the software library. Thus, ensures that a secure and robust system is used [24].

11.2 Brief History

During the early seventies, information hiding idea was proposed for separating systems into software libraries. During that time, the aim was to boost the systems flexibility and understandability to reduce the development time. Despite

during that time, criterion was thought to be uncommon and strange. In an article written by David Parnas, had proposed an idea of splitting systems into components, however that was dismissed by the assessors. One analyst had asserted that the method was not utilised by anyone, whereas the other refused it for the opposite reason and stated that it was in operation and used universally [26].

Because of David Parnas unique outlook in the field, he had experienced significant challenges in his professional work life [26].

It was then announced that David Parnas idea of information hiding had resulted in a misfortune by Fred Brooks. As a result of Fred Brooks announcing fake news, it had caused twenty years for Fred Brooks to be responsible for his mistake and admit he was wrong and that David Parnas's idea's and concepts were valid and accurate within the industry. This caused misunderstanding through the literature of the idea and concept of hiding information after five years, as shown in the presentation in the early 21st century by Berard [26].

12 Conclusion

This report focuses on the detailing the four concepts which are important in software development. High cohesion and loose coupling are concepts which are in close association and work together by ensuring elements achieve a single purpose, which ensures that the software library has good readability and maintainability. Despite this, having low cohesion makes the code difficult to read and manage. Data hiding is a crucial concept which also goes with information hiding, while they are not the exact same. The idea is that they are concepts which are used in OOP to hide the internal workings of an object, making the class private which ensures that the class cannot be accessed externally, which maintains the data and system.

References

- [1] D. Janzen and H. Saiedian, "Test-driven development concepts, taxonomy, and future direction," *Computer*, vol. 38, no. 9, pp. 43–50, Sep. 2005.
- [2] F. Anwer, S. Aftab, U. Waheed, and S. S. Muhammad, "Agile software development models tdd, fdd, dsdm, and crystal methods: A survey," *International journal of multidisciplinary sciences and engineering*, vol. 8, no. 2, pp. 1–10, 2017.
- [3] E. H. Vihovde and Q. Meng, "Test-driven development: Ensuring code quality in integration with ci/cd," in *2024 8th International Conference*

on Management Engineering, Software Engineering and Service Sciences (ICMSS). IEEE, Jan. 2024, pp. 8–11.

- [4] S. Alsaqqa, S. Sawalha, and H. Abdel-Nabi, “Agile software development: Methodologies and trends,” *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 14, no. 11, p. 246, Jul. 2020.
- [5] S. Aftab, Z. Nawaz, M. Anwar, F. Anwer, M. S. Bashir, and M. Ahmad, “Comparative analysis of fdd and sfdd,” *International Journal of Computer Science and Network Security*, vol. 18, no. 1, pp. 63–70, 2018.
- [6] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, “Agile software development methods: Review and analysis,” *CoRR*, vol. abs/1709.08439, 2017. [Online]. Available: <http://arxiv.org/abs/1709.08439>
- [7] A. F. Arbain, I. Ghani, and S. R. Jeong, “A systematic literature review on secure software development using feature driven development (fdd) agile model,” *Journal of Internet Computing and services*, vol. 15, no. 1, pp. 13–27, 2014.
- [8] S. Aftab, Z. Nawaz, F. Anwer, M. Ahmad, A. Iqbal, A. Ahmad, and M. Salman, “Using fdd for small project: An empirical case study,” *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 3, 2019.
- [9] Z. Nawaz, “Proposal of enhanced fdd process model,” *Int J Educ Manage Eng (IJEME)*, vol. 11, no. 4, pp. 43–50, 2021.
- [10] R. Broer Bahaweres, E. Oktaviani, L. Kesuma Wardhani, I. Hermadi, A. Suroso, I. Permana Solihin, and Y. Arkeman, “Behavior-driven development (bdd) cucumber katalon for automation gui testing case cura and swag labs,” in *2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*. IEEE, Nov. 2020, pp. 87–92.
- [11] H. M. Abushama, H. A. Alassam, and F. A. Elhaj, “The effect of test-driven development and behavior-driven development on project success factors: A systematic literature review based study,” in *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*. IEEE, Feb. 2021, pp. 1–9.
- [12] 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. 88 008–88 024, 2023.
- [13] R. K. Lenka, S. Kumar, and S. Mamgain, “Behavior driven development: Tools and challenges,” in *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. IEEE, Oct. 2018, pp. 1032–1037.

- [14] S. Software, “History of bdd,” Online via Cucumber Documentation, 2019. [Online]. Available: <https://cucumber.io/docs/bdd/history/>
- [15] S. Bruschi, L. Xiao, M. Kavatkar *et al.*, “Behavior driven development (bdd): a case study in healthtech,” in *Pacific NW Software Quality Conference*, 2019.
- [16] L. P. Binamungu, S. M. Embury, and N. Konstantinou, “Maintaining behaviour driven development specifications: Challenges and opportunities,” in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 175–184.
- [17] V. Bakhtiary, T. J. Gandomani, and A. Salajegheh, “The effectiveness of test-driven development approach on software projects: A multi-case study,” *Bulletin of Electrical Engineering and Informatics*, vol. 9, no. 5, pp. 2030–2037, 2020.
- [18] S. Parsa, M. Zakeri-Nasrabadi, and B. Turhan, “Testability-driven development: An improvement to the tdd efficiency,” *Computer Standards & Interfaces*, vol. 91, p. 103877, 2025.
- [19] S. Sharma and S. Srinivasan, “A review of coupling and cohesion metrics in object oriented environment,” in *A review of Coupling and Cohesion metrics in Object Oriented Environment paper*, 2013. [Online]. Available: <https://api.semanticscholar.org/CorpusID:1038098>
- [20] A. Rathee and J. K. Chhabra, “Improving cohesion of a software system by performing usage pattern based clustering,” *Procedia Computer Science*, vol. 125, pp. 740–746, 2018.
- [21] G. Y. d. O. Moreira and J. A. M. Santos, “Applying coupling and cohesion concepts in object-oriented software: a controlled experiment,” in *Proceedings of the XIX Brazilian Symposium on Software Quality*, ser. SBQS ’20. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: <https://doi.org/10.1145/3439961.3439969>
- [22] 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*, ser. ISEC ’18. ACM, Feb. 2018, pp. 1–11.
- [23] M. A. AWODUN and M. O. TENIBIAJE, “Data hiding technique in information technology: A review,” *IJO-International Journal Of Computer Science and Engineering (ISSN: 2814-1881)*, vol. 7, no. 09, pp. 01–09, 2024.
- [24] W. Al Abed and J. Kienzle, “Information hiding and aspect-oriented modeling,” in *14th Aspect-Oriented Modeling Workshop, Denver, CO, USA*. Citeseer, 2009, pp. 1–6.

- [25] P. Brada, “A look at current component models from the black-box perspective,” in *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2009, pp. 388–395.
- [26] P. Flores, N. Medinilla, and S. Pamplona, “What do software design students understand about information hiding?: a qualitative case study,” in *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling ’14. ACM, Nov. 2014, pp. 61–70.