Comparative Analysis of FDD and SFDD

Shabib Aftab†, Zahid Nawaz†, Madiha Anwar††, Faiza Anwer†, Muhammad Salman Bashir†, Munir Ahmad†
†Department of Computer Science, Virtual University of Pakistan
††Department of Computer Science, University of Gujrat, Lahore Campus, Pakistan

Summary
No doubt, development of high quality software depends upon the selection of software process model. Conventional software development models such as Water fall, Spiral and V-Model have been dominant in software industry till mid 1990s and then the era of agile development models started. Agile process models got the attention of software industry by proposing the solutions of problems which developers were facing with conventional models. Feature Driven Development (FDD) is one of the widely used software development models from agile family. FDD is known as client centric model as it develops the software product according to client valued features. It follows adaptive and incremental approach to implement the required functionality and focuses on designing and building aspects of software development with more emphasis on quality. However besides the benefits, FDD lacks at some areas. Having less ability to respond towards the changing requirements, reliance on experienced staff and no focus on small projects are the main problems of FDD. To overcome these issues, Simplified Feature Driven Development (SFDD) was proposed. This paper empirically compares both the models by presenting the results, which are obtained from the development of real time client oriented projects.

Key words:
Agile Modeling, Process Evaluation, Empirical Evaluation, Empirical Comparison, Feature Driven Development, Simplified Feature Driven Development, Modified FDD, SFDD.

1. Introduction

Agile models provided a lighter way of software development with the intention to overcome the limitations of traditional software development models. Drawbacks of traditional models include less user interaction, long development duration, high cost, no adaptability and most importantly no response to frequently changing user requirements [17],[18]. Agile methodologies shifted the focus from process to people and valued those factors which were neglected in traditional models [19]. Agile models include Extreme Programming (XP), Scrum, Test Driven Development (TDD), Dynamic System Development Model (DSDM), Crystal methods and Feature Driven Development (FDD) etc [17],[19]. All these models follow the values, principles and practices suggested by agile manifesto. The agile manifesto can be considered as a parent document of all agile models which contains twelve foundation principles of software development. These principles are about frequent team communication, customer satisfaction, managing frequent changing requirements and early delivery of partial working software module [17],[18],[19],[23]. The teams in agile models are self-organizing where members work in close collaboration with each other, moreover agile manifesto focuses on timely delivery of reliable and quality product with simple design. These models develop the software in multiple iterations, each iteration ends with a working module of the complete upcoming product which helps in early feedback from the customer [20],[21], [22]. FDD is a process oriented agile development model that mainly focuses on design and building aspects of software development [11],[12],[13]. The lifecycle of FDD follows a well-known pattern called ETVX. The development process is completed in five phases, which are: Develop an Overall Model, Build a Features List, Plan by Feature, Design by Feature and Build by Feature. Each phase consists of different tasks and activities [11],[13], [14]. FDD develops the Software according to client valued functionality by using the iterative and incremental approach [15]. It uses eight best practices such as: domain object modeling, development by feature, individual class ownership, feature teams, inspection, configuration management, regular builds and progress reporting [11] [12],[16]. Besides the advantages, FDD faced some limitations as well such as its heavy structure only makes it suitable for medium to large scale projects. Further limitations include explicitly dependence upon experience staff and rigid nature for handling changing requirements. To overcome these limitations a simplified version of FDD called SFDD is proposed by [16] which tried to reduce its limitations without affecting the agility. SFDD is designed for small to medium scale projects with the feature of handling changing requirements more effectively. Besides the designing and building aspects the proposed model also concentrates on early delivery of qualitative product by introducing a testing phase within the iteration. SFDD also removed the constraint of trained staff which was one of the key limitations of classical FDD. The purpose of this paper is to perform a comparative analysis of FDD and SFDD on the base of empirical results.

Further organization of this paper is as follows. Section 2 describes different attempts of FDD customizations and also precisely explains FDD & SFDD models. Section 3 presents the comparative analysis of both models. Section 4 finally concludes the paper.

Manuscript received January 5, 2018.
Manuscript revised January 20, 2018.

2. Material and Methods

FDD has been discussed and tailored by many researchers in the last decade. To make the FDD more effective, its limitations have to be eliminated. For this purpose, some researchers have presented its customized versions whereas some proposed its integrations with other process models. Here we are going to discuss some selected studies. In [1], authors proposed SCR-FDD, which is a hybrid model that integrated Scrum and FDD. This model tried to reduce the limitations of both the agile methods by taking the schedule related aspects from Scrum and quality related aspects from FDD. The proposed solution has tried to resolve the limitations regarding schedule, quality and deployment. Authors in [2] presented Feature-Driven Methodology Development (FDMD), an extended version of Feature Driven Development. The features of object oriented approach are integrated with Situational Method Engineering (SME) in FDMD. The proposed solution represented the requirements as features, which are based on object oriented principles. This model tried to reduce the issues of maintainability and reusability. Authors of [3] proposed an extended version of FDD, Secure Feature Driven Development (SFDD). This model introduced two new phases in the development life cycle of FDD, "Build security by feature" and "Test security by feature". Moreover it also included the element of "In-phase Security" in each phase as well as the new role called security master, to ensure the secure software development. Authors of [4] proposed Feature Driven Reuse Development (FDRD) which introduced reusability feature by considering re-useable feature sets for new requirements. In [5], authors presented an ontology based feature driven development model for semantic web application. The proposed model uses the concepts of domain ontology from domain knowledge modeling. Each phase of this model consists of ontology as a basic building block. Ontology languages like RDF and OWL helped to overcome the language ambiguity and inconsistency. In [6], authors conducted a case study to check the suitability of FDD for secure web development. They have pointed out that by integrating more iterations, security practices and other helping tools can make this model suitable for secure software development. Authors in [7] presented a framework to handle the changing requirements in an efficient way. The proposed model is based on Adaptive Software Development and Cognizant Feature Driven Development (CFDD). CFDD is a customized version of FDD. In [8], authors have proposed a hybrid software architecture evaluation method (SAEM) by integrating Quality Attribute Workshop (QAW), Architecture Trade-off Analysis Method (ATAM) and Active Review for Intermediate Designs (ARID) with FDD. Authors in [9], presented a supporting tool for the implementation of FDD. This tool implements the model in a multi-user web based environment, in the form of sub processes. This tool holds the ability to track changes in requirements and also can map the modifications in design classes. In [10], authors customized the FDD for aspect oriented development. The proposed solution focused on the separation of concerns which can help in handling complexity and maintenance problems. According to authors the refinement in FDD can be helpful for the detection of inconsistencies among features and also can help in smooth transition from one phase to other.

2.1 Feature Driven Development (FDD)

FDD is a process oriented and client centric agile process model which works by focusing on the designing and building aspects of software development process [11], [12],[13]. FDD follows a well-known pattern called ETVX and consists of five phases also known as processes [12]. The phases include: 1) Develop an Overall Model, 2) Build a Features List, 3) Plan by Feature, 4) Design by Feature and 5) Build by Feature. Each phase further consists of different series of activities [11],[13],[14]. Like other agile process models, it also follows iterative and incremental approach for software development. FDD develops the software according to client valued features [15] by using eight best practices such as: domain object modeling, development by feature, individual class ownership, feature teams, inspection, configuration management, regular builds and progress reporting [11], [12]. In the first phase, key activity is the development of overall model, which is performed after the discussion regarding scope and context of the project in a walkthrough meeting [11],[12]. The responsibility of modeling authority is to select one best model for initiating further processes [11] then different domain experts develop object models. Second phase focuses on the creation and management of feature/requirement list, which would be developed in later phases. The feature list is further classified into groups called feature set [12],[13]. In third phase, priority is assigned to every feature [11] so that the higher priority feature would be considered in early iterations. After assigning the priority, every feature is checked against its business need, which verifies that the features are according to the project's requirements. This phase also deals with the identification of dependencies among features and measuring the complexities. Feature ownership is also assigned to each developer in the form of classes. Fourth phase focuses on different activities such as: designing the sequence diagrams, writing the classes and refining the overall model [11]. Moreover different design packages are also produced against each class in this phase [11],[13].

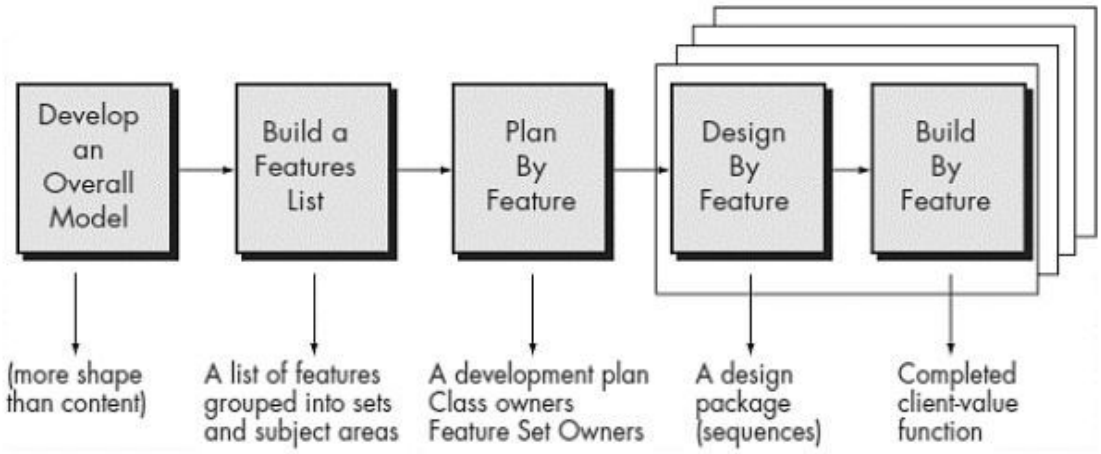

Fig. 1 FDD Process Model [18]

The fifth and last phase deals with the implementation and development of design packages. This phase includes many activities such as coding, code inspection, unit testing and integration testing [11],[12]. These activities are actually performed in iterations. FDD defines six key roles such as project manager, chief architect, development manager, chief programmer, class owner and domain experts. Besides the key roles, this model has five supporting roles such as: release manager, language guru, build engineer, tool smith and system administrator. It also includes three additional roles: tester, deployer and technical writer [11],[12].

2.2 Simplified Feature Driven Development (SFDD)

Authors in [16] presented Simplified Feature Driven Development (SFDD) model to eliminate the limitations of classical FDD. SFDD focuses on small to medium scale projects where requirements are more likely to change. The proposed solution concentrates on story cards for requirement elicitations and also intends to improve the software quality by introducing a testing phase within the iteration. SFDD also removed the constraint of trained staff which was one of the key limitations of classical FDD. The first phase of SFDD is 'Develop an Overall Model', in which project scope is finalized and requirements are gathered. Project scope is finalized by the chief programmer and domain expert, which are the two active participant of this phase. Chief Programmer is the focal person from development team and Domain Expert represents the client. The requirements are presented by the Domain expert along with the priorities in the form of story cards. These story cards effectively explain the required functionality without involving any technical detail. After the completion of requirement gathering task, the chief programmer develops the use case diagrams and class diagrams with the help of other team members. At the end of this phase four documents are generated: 1) Project Scope, 2) Functional & Non-Functional Requirements, 3) Use-case Diagrams and 4) class diagram. The second phase of SFDD is 'Build Feature List'. In this phase the chief programmer extracts and classifies the features for each domain of the system to be developed by using the documents produced in previous phase. Features under a specific domain are called a feature sets. The related requirements are collected as a single feature set. Complete list of features is documented and approved by the domain expert. At the end of this phase one document is generated named feature list. Third phase of SFDD is 'Plan by Feature'. This phase deals with the project planning activities and starts with a meeting between Domain Expert and Chief Programmer regarding the financial budget and time frame. This activity is followed by the development of project plan in which chief programmer decides about the number of iterations, selection of the features to be developed in each iteration and other required resources such as hardware, softwares, time and effort (resource persons). After all these crucial estimations, chief programmer assigns classes to class owners. At the end of this phase one document is generated named Project Plan. Fourth phase of SFDD is 'Design by Feature'. In this phase, the class diagrams which were developed in first phase are refined. After that an object model is developed of the system by Chief Programmer and Class owners. The pseudo code is written by the class owners for their concerned classes. Complete design of the software is documented and inspected by the QA a manager. Fifth phase of SFDD is 'Build by Feature'. This phase enters in the iteration. The purpose of the iteration is to develop and deliver the project in small workable modules. The iteration of SFDD consists of two phases: Build by feature phase and the Test by feature phase.

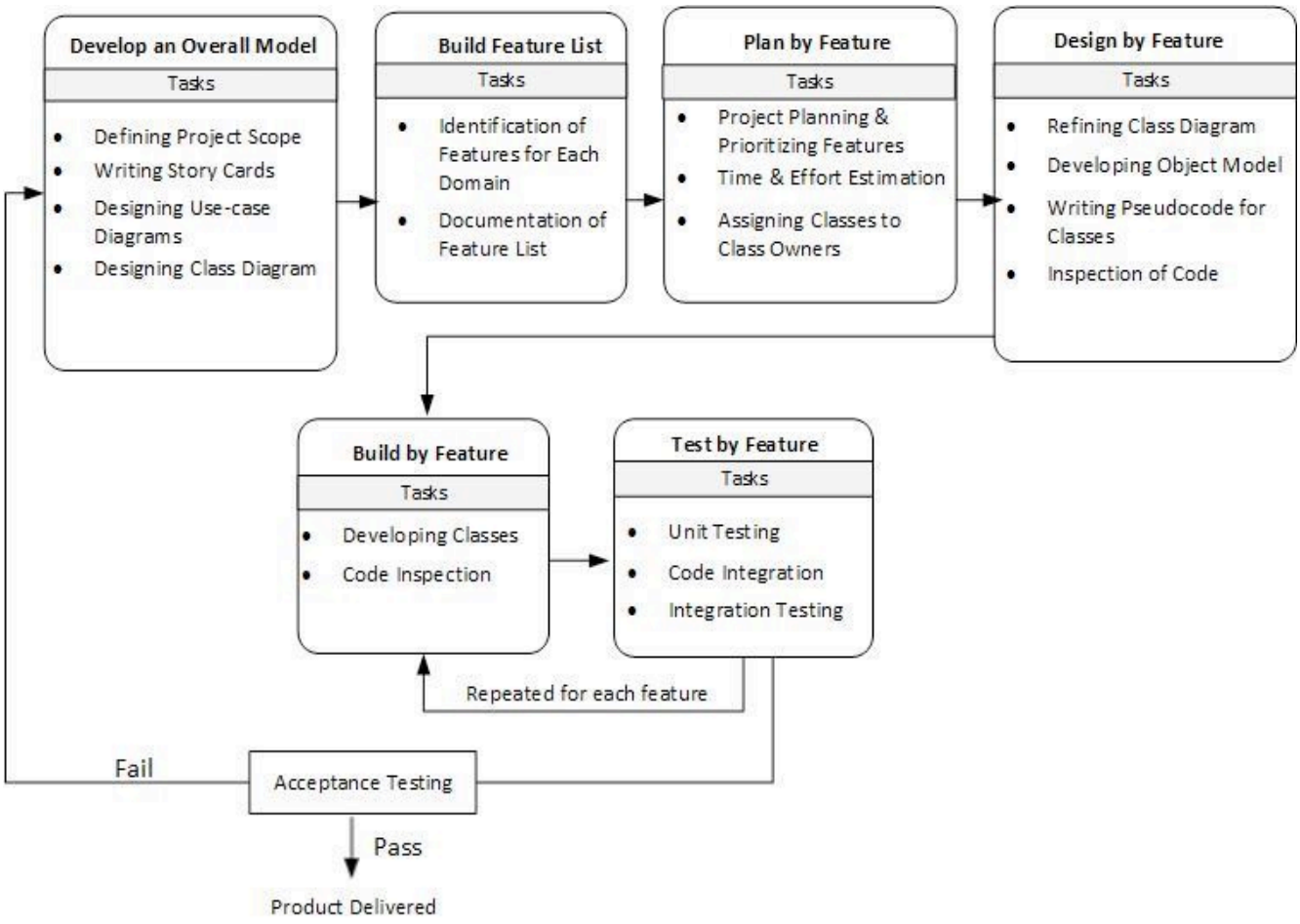

Fig. 2 SFDD Process Model [16]

This phase deals with the actual implementation of features and classes. The class owners write code and a formal code inspection session is conducted in the supervision of QA manager to assure that code is written according to the pseudo code and is working properly. At the end of this phase small workable module will be ready to go in the next phase of the iteration. A document named Inspected Module is generated which will consist of the detail regarding developed module. Sixth phase of SFDD is 'Test by Feature'. This is the second phase of iteration and the last phase of SFDD. This phase makes sure that the software is bug free and working according to the required features. It starts with the unit testing in which QA manager assured that the developed module is working properly as per required functions. In case of successful unit testing the module is integrated with the previously developed module followed by integration testing. Domain expert finally performs the acceptance testing. At the end of this phase two documents are generated, Testing document and User's manual.

4. Results and Discussions

FDD and SFDD both are implemented via real time client oriented projects in a software house, situated in Islamabad, capital of Pakistan. The software house consists of experienced staff with dominating knowledge of software development along with higher degrees in computer science. They were already using agile models for most of the development. Both case studies were carried out in same working environment but with different teams. FDD case study was implemented by an experienced team whereas SFDD case study was assigned to the team having less or no experience with agile development. However training session of 10 days was organized for those team members.

Two web based applications were developed with FDD and SFDD respectively. Both applications were related to Human Resource Management Systems. Most of the characteristics of applications were same such as size of the project, no of iterations and no of team members etc, details are given in Table 1 and Table 2.

Table 1: Case Study of FDD

Characteristics	Description
Product Type	Human Resource Management
Size	Small
Number of Modules	6
No of User Stories	57
Number of User Interfaces	12
Iterations	4
Programming Approach	Object Oriented
Language	C#, ASP.NET
Documentation	MS Office
Testing	Browser Stack
Web Server	IIS
Project Type	Average
Team Size	5 Member
Feedback	Weekly
Development Environment	Visual Studio 2012
Other Tools	MS Visio
Reports	Crystal Report

Table 2: Case Study of SFDD

Characteristics	Description
Product Type	Human Resource Management
Size	Small
Number of Modules	4
No of User Stories	65
Number of User Interfaces	10
Iterations	4
Programming Approach	Object Oriented
Language	C#, ASP.NET
Documentation	MS Office
Testing	Browser Stack
Web Server	IIS
Project Type	Average
Team Size	5 Member
Feedback	Weekly
Development Environment	Visual Studio 2012
Other Tools	MS Visio
Reports	Crystal Report

Comparative analysis is performed on the basis of following performance related parameters:

- Completion Time
- Total Line of Code
- Budgeted Work Effort
- Actual Work Effort
- Post Release Defects
- Team Productivity
- Time to Manage Pre-release Change Requests

Table 3 describes the overall performance of both models with the selected parameters. It can be seen that SFDD improved the overall development process as it was simplified to handle small to medium scale projects in an efficient manner. The Results empirically demonstrates the effectiveness of proposed SFDD.

Table 3: Comparison of FDD and SFDD

Parameter	FDD	SFDD
Completion Time (in weeks)	4	3.2
Total Tine of Code (LOC)	12810	13110
Budgeted Work Effort (in hours)	800	640
Actual Work Effort (in hours)	700	592
Post Release Defects	12	5
Team Productivity	18.3	22.14
Pre-release Change Request	10	12
Time to Manage Change (in hours)	14	11

Total project completion time is less in SFDD because of its simplified nature. FDD contains heavy architectural design and a complicated development life cycle along with large number of roles which makes it only suitable for large scale projects. This is the reason that its completion time with small project is larger than the completion time of SFDD (Table 3, Fig. 3).

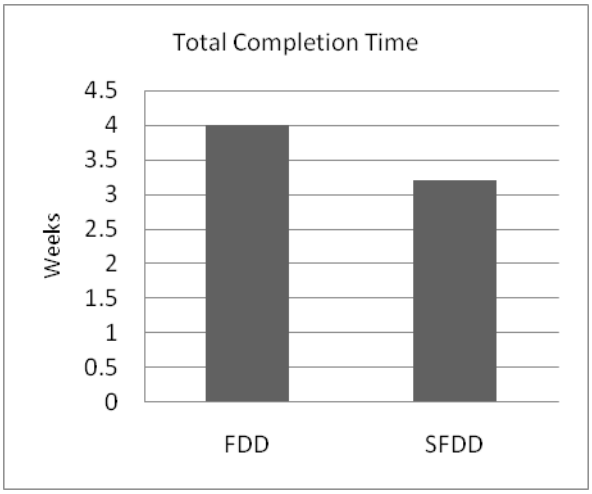

Fig. 3 Total Completion Time

One notable thing is that the line of code was larger with the case of SFDD but with effective customization the completion time is significantly lower than FDD. The budgeted work effort is also lower with the case of SFDD. Total budgeted work effort is calculated using the following formula;

Total Budgeted Work Effort (h) =No of hours in a day (8) * No of days in a week (5) *No of weeks* Total team size (5).
The parameters other than the completion time are same for both the case studies. As the completion time is less in SFDD than FDD so the budgeted work effort would also be lower in SFDD (Table 3, Fig. 4).

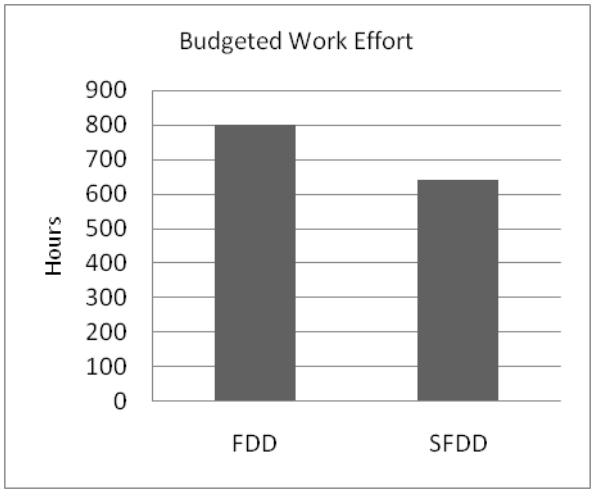

Fig. 4 Budgeted Work Effort

Actual spending hours are always less from budgeted as some time is consumed on other related activities. The actual work effort depends upon the actual time spent in a day (h) for the development (Table 3, Fig. 5). In case of SFDD, on averege 7.4 hours were given daily for project development whereas in FDD the number of hours are 7.

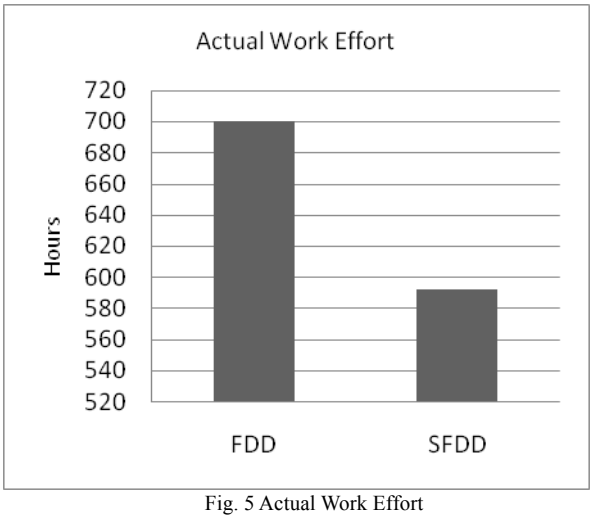

Fig. 5 Actual Work Effort

The formula for actualwork effort is given below:
Actual Work Effort (h) =No of actual working hours (h) * No of days in a week (5) *No of weeks* Total team size (5).

The number of Post release defects is one of the important parameters related to the quality of developed software and also reflects the customer's satisfaction. In case of FDD 12810 LOC were written and total 12 post release defects were reported however in SFDD case study 13110 LOC were written and total of 5 defects were reported. So the complicated nature of FDD could not handle the small project effectively. SFDD on the other hand performed quite well due to its effective testing series in iterative manner (Table 3, Fig. 6).

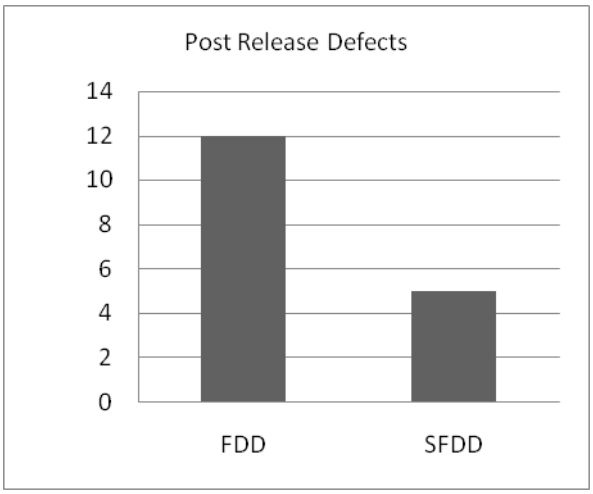

Fig. 6 Post Release Defects

Team productivity is also an important parameter to analyze the development performance with any particular SDLC. However only this parameter is not enough to judge the performance as post release defects can affect this reflection.

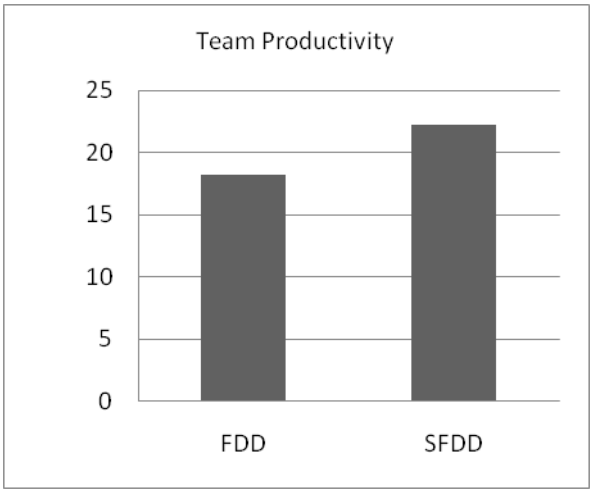

Fig. 7 Team Productivity

Team Productivity is calculated using the following formula;
Team Productivity = line of code / actual time spent in hours.
As in SFDD case study, more lines of code were written in less time that's why team productivity is higher than FDD (Table 3, Fig. 7).
In FDD case study 10 pre-release change request were implemented in 14 hours however with SFDD, 12 requests were implemented in 11 hours (Table 3, Fig. 8). This quality parameter shows the advantage of SFDD over FDD with respect to change implementation. SFDD performed better due to its simple design and effective customization. These features make the SFDD to track and implement changes more efficiently.

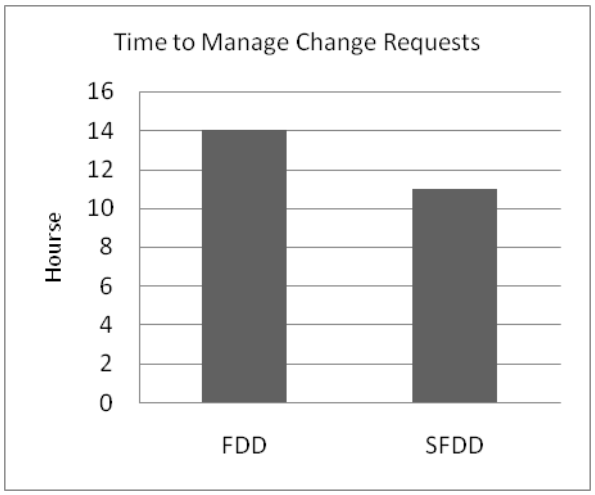

Fig. 8 Time to Manage Change Requests

4. Conclusion

This paper performed a comparative analysis of classical FDD and SFDD. FDD is an agile development model which follows process oriented and iterative approach with more focus on software quality. Its complex architecture, large number of roles and dependency upon experienced staff make it only suitable for large scale projects where there is less tendency of change in requirements. On the other hand SFDD is proposed to overcome these limitations without affecting the agility. It's simple architecture makes it suitable for small to medium scale projects and it can also handle changing requirements in an effective and efficient manner. For comparative analysis, two software applications having same nature were developed by FDD and SFDD respectively. SFDD reported better results in terms of completion time, budgeted work effort, actual work effort, no of post release defects, team productivity and time to manage pre-release change requests. From empirical comparison, It can be said that SFDD has proved to be effective than classical FDD in terms of quality, efficiency and effectiveness. However SFDD should be tested further for medium scale projects.

References

[1] S. S. Tirumala, S. Ali, and A. Babu G, "A Hybrid Agile model using SCRUM and Feature Driven Development," International Journal of Computer Applications, vol. 156, no. 5, pp. 1–5, 2016.
[2] R. Mahdavi-Hezave and R. Ramsin, "FDMD: Feature-Driven Methodology Development," Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software Engineering, pp. 229–237, 2015.
[3] A. Firdaus, I. Ghani, and S. R. Jeong, "Secure Feature Driven Development (SFDD) Model for Secure Software Development," Procedia - Social and Behavioral Sciences, vol. 129, pp. 546–553, 2014.
[4] S. Thakur and H. Singh, "FDRD: Feature driven reuse development process model," in Proceedings of 2014 IEEE International Conference on Advanced Communication, Control and Computing Technologies, ICACCCT 2014, 2015, pp. 1593–1598.
[5] F. Siddiqui and M. A. Alam, "Ontology based application model for feature driven development," Proceedings of the 5th Indian International Conference on Artificial Intelligence, IICAI 2011, pp. 1125–1137, 2011.
[6] A. Firdaus, I. Ghani, and N. I. M. Yasin, "Developing Secure Websites Using Feature Driven Development (FDD): A Case Study," Journal of Clean Energy Technologies, vol. 1, no. 4, pp. 322–326, 2013.
[7] K. Kumar, P. K. Gupta, and D. Upadhyay, "Change-oriented adaptive software engineering by using agile methodology: CFDD," in ICECT 2011 - 2011 3rd International Conference on Electronics Computer Technology, 2011, vol. 5, pp. 11–14.
[8] F. Kanwal, K. Junaid, and M. A. Fahiem, "A hybrid software architecture evaluation method for FDD - An agile process model," in 2010 International Conference on Computational Intelligence and Software Engineering, CiSE 2010, 2010.
[9] M. Rychlý and P. Tichá, "A tool for supporting feature-driven development," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2008, vol. 5082 LNCS, pp. 196–207.
[10] J. Pang and L. Blair, "Refining Feature Driven Development - A Methodology for Early Aspects," Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, pp. 85–90, 2004.
[11] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods: Review and analysis," VTT Publications, no. 478. pp. 3–107, 2002.
[12] S. R. Palmer and M. Felsing, A Practical Guide to Feature Driven Development. 2002.
[13] S. Goyal, "Major Seminar On Feature Driven Development," p. 22, 2007.
[14] B. Boehm, "A Survey of Agile Development Methodologies," Laurie Williams, pp. 209–227, 2007.
[15] P. Coad, J. D. Luca, and E. Lefebvre, "Java Modeling In Color With UML," in Java Modeling In Color With UML: Enterprise Components and Process, no. c, 1999, pp. 1–12.

[16] Z. Nawaz, S. Aftab, and F. Anwer, “Simplified FDD Process Model,” International Journal of Modern Education and Computer Science, vol. 9, no. 9, pp. 53–59, 2017.
[17] G. Rasool, S. Aftab, S. Hussain, and D. Streitferdt, “eXRUP: A Hybrid Software Development Model for Small to Medium Scale Projects,” Journal of Software Engineering and Applications, vol. 6, no. 9, pp. 446–457, 2013.
[18] 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.
[19] F. Anwer, S. Aftab, S. S. M. Shah, and U. Waheed, “Comparative Analysis of Two Popular Agile Process Models: Extreme Programming and Scrum,” International Journal of Computer Science and Telecommunications Journal, vol. 8, no. 2, pp. 1–7, 2017.
[20] F. Anwer and S. Aftab, “SXP: Simplified Extreme Programing Process Model,” International Journal of Modern Education and Computer Science, vol. 9, no. 6, pp. 25–31, 2017.
[21] F. Anwer and S. Aftab, “Latest Customizations of XP : A Systematic Literature Review,” International Journal of Modern Education and Computer Science, vol. 9, no. 12, pp. 26–37, 2017.
[22] S. Ashraf and S. Aftab, “Latest Transformations in Scrum: A State of the Art Review,” International Journal of Modern Education and Computer Science, vol. 9, no. 7, pp. 12–22, 2017.
[23] M. R. J. Qureshi, “Agile software development methodology for medium and large projects,” IET Software, vol. 6, no. 4, p. 358, 2012.