Implementing Test Driven Development in the Big Data Domain: A Movie Recommendation System as an Exemplary Case

Daniel Staegemann[a], Matthias Volk[b], Priyanka Byahatti, Nikhilkumar Italiya, Suhas Shantharam, Apoorva Byaladakere Chandrashekar and Klaus Turowski
Magdeburg Research and Competence Cluster VLBA, Otto-von-Guericke University Magdeburg, Magdeburg, Germany
{daniel.staegemann, matthias.volk, priyanka.byahatti, nikhil.italiya, suhas.shantharam, apoorva.bc,

Keywords: Big Data, Test Driven Development, TDD, Microservice, Software Engineering, Quality Assurance.

Abstract: As a consequence of the ongoing digitalization in today's society, the amount of data that is being produced is rapidly increasing. Moreover, not only the volume of the data is growing, but there are also more complex types of data and, depending on the use case, it is also necessary to integrate heterogenous data into one analysis. Since traditional ways of dealing with data are oftentimes overstrained by those new challenges, novel approaches and technologies have been developed. In its entirety, this phenomenon is summarized under the term big data. However, quality assurance in the big data realm is still not mature and this even more applies to the actual testing. Therefore, it is necessary to explore new approaches. One rather recent proposition was the application of the test driven development methodology to the big data domain. To further evaluate its feasibility and go beyond a purely theoretical point of view, the publication at hand discusses the test driven implementation of a movie recommendation system as an exemplary case. In doing so, it facilitates the general understanding of the topic, helps in judging the approach's feasibility and provides some practical insights concerning its actual application.

1 INTRODUCTION

As a consequence of the ongoing digitalization in today's society (Musik and Bogner, 2019), the amount of data that is being produced is rapidly increasing (Herschel and Miori, 2017). Moreover, those data are not only produced, but oftentimes also captured, stored and/or analyzed. However, not only the volume of the data is increasing, but there are also more complex types of data (e.g. image, audio or video) and, depending on the use case, it is also necessary to integrate heterogenous data into one analysis (Volk et al., 2020b). Since traditional ways of dealing with data are oftentimes overstrained by those new challenges (Zhu et al., 2019), novel approaches and technologies have been developed, which are subsumed under the terms big data (BD), respectively big data analytics (BDA).

When implemented and applied correctly, BDA promises noticeable benefits (Müller et al., 2018). Yet, its utilization is a highly complex endeavour that is based on several dimensions (Staegemann et al., 2019).

On the one hand, the data utilized as input have to be of high quality to allow for good results (Hazen et al., 2014), while on the other hand, those who operate and steer the systems have to be qualified (Lee, 2017). Further, if BDA is supposed to support human decision making, those that are in charge also need to be willing to incorporate the findings instead of ignoring them or only using them when it is to support their own pre-determined opinion (Günther et al., 2017). In addition, even if the aforementioned factors are sufficiently covered, the actual implementation of a BDA application is also a highly challenging task (Volk et al., 2019; Volk et al., 2020a). Subsequently, a very important part of that process is the testing of the developed solution. However, quality assurance in the big data realm is still not mature and this even more applies to the actual testing (Davoudian and Liu, 2020; Ji et al., 2020). Therefore, it is necessary to explore new approaches (Staegemann et al., 2021b).

[a] https://orcid.org/0000-0001-9957-1003
[b] https://orcid.org/0000-0002-4835-919X

Staegemann, D., Volk, M., Byahatti, P., Italiya, N., Shantharam, S., Chandrashekar, A. and Turowski, K.
Implementing Test Driven Development in the Big Data Domain: A Movie Recommendation System as an Exemplary Case.
DOI: 10.5220/0011085600003194
In *Proceedings of the 7th International Conference on Internet of Things, Big Data and Security (IoTBDS 2022)*, pages 239-248
ISBN: 978-989-758-564-7; ISSN: 2184-4976

One rather recent proposition was the application of the test driven development methodology to the big data domain (Staegemann et al. 2020b). To further evaluate its feasibility and go beyond a purely theoretical point of view, the publication at hand is committed to answering the following research questions:

RQ1: Is the test driven development methodology a feasible approach for implementing big data applications and how can it be applied?

RQ2: What are the implications and findings of applying the test driven development methodology in a big data context?

While only one single exemplary case is regarded and, therefore, a general statement cannot be deducted solely based on this contribution, it will facilitate the general understanding of the topic, help in judging the approach's feasibility and provide some practical insights concerning its actual implementation.

For this purpose, the remainder of this work is structured as follows. After this introduction, necessary background information concerning the concepts of big data and test driven development are given. Afterwards, the exemplary task is introduced and some of its challenges highlighted. Subsequently, in the fourth section, the actual implementation is described. This is followed by a discussion of the corresponding findings. Finally, a conclusion is provided and potential directions for future research are highlighted.

2 BACKGROUND

To provide a foundation that the ensuing parts of the publication at hand can build upon, in the following, the concepts of big data and test driven development are briefly discussed.

2.1 Big Data

With the amount of data being produced, captured and analysed rapidly increasing as well as its complexity and the demands for its processing growing, traditional applications that were previously used for its harnessing are oftentimes no longer sufficient (Chang and Grady 2019). Subsequently, new tools and techniques had to be developed, which are able to satisfy the challenges posed by this new trend that is referred to as big data.

While there is no unified definition for the term (Al-Mekhlal and Ali Khwaja 2019; Volk et al. 2020c), the understanding in the majority of the pertinent literature is quite similar. The arguably most popular description (Chang and Grady 2019) is based on the 4 Vs of big data, namely volume (number of data entries and size of data sets), velocity (speed of incoming data and speed requirements for the processing), variety (diversity of data in structure and content) and variability (changes in data over time).

Since improved decision making can benefit organizations across various fields of activity, BDA is being applied to a plethora of domains, such as agriculture (Bronson and Knezevic 2016), education (Häusler et al. 2020), healthcare (Bahri et al. 2019), manufacturing (Nagorny et al. 2017) and sports (Goes et al. 2020) to name just a few.

2.2 Microservices

The microservice concept generally bases on decomposing an envisioned application into a number of smaller services that interact with each other (Nadareishvili et al. 2016). Usually, those are based on business functionality, which allows for a high degree of specialization. They all run in their own processes and the communication between those services is realized only over lightweight mechanisms. Furthermore, they can be heterogeneous regarding the programming languages and technology stacks used for their implementation (Freymann et al. 2020). Those properties allow for them to be deployed independently of each other by utilizing continuous deployment tools and pipelines.

While componentization is generally considered a good software engineering practice, it is often seen as challenging to achieve a high degree of modularity (Faitelson et al. 2018). However, with microservices, this is achieved by design. This also translates to a reduced effort for maintenance and modifications, because for changes it is often sufficient to only redeploy the affected service. Consequently, an evolutionary design is promoted, which is driven by frequent and controlled changes (Krylovskiy et al. 2015).

2.3 Test Driven Development

In the literature (Staegemann et al. 2021a), test driven development (TDD) is highlighted as a promising approach to improve an implementation's quality. This is mainly achieved by influencing two aspects. Following this strategy, the test coverage is increased, which helps to find errors and, further, the system's

design is changed, since emphasis is given to breaking it down into the smallest sensible pieces. This helps to avoid issues and mistakes that are caused by high complexity and increases maintainability (Crispin, 2006; Shull et al., 2010). Besides software development, applications of TDD can also be found in other domains such as ontology development (Davies et al., 2019; Keet and Ławrynowicz, 2016) and process modelling (Slaats et al., 2018). Yet, in the context of the publication at hand, those are not as relevant.

Usually, in software development, after a desired feature has been determined, it is implemented and then tested. When applying TDD, instead, the order of those activities is changed. Therefore, after it is decided which change is to be realized, it is broken down into the smallest reasonable parts (Fucci et al., 2017). Subsequently, one or more tests are written for those, to assure that they are working as intended. Then, those tests are run with the expectation of them failing, since the new functionality still needs to be implemented (Beck, 2015). Consequently, if the test succeeds nevertheless, this means that it is not sufficiently designed and needs to be reworked. After the test failed, the actual productive code is created to implement the desired functionality. However, there is no need for it to already be perfectly and elegantly designed, since the goal is to provide the simplest solution that passes the previously written tests (Crispin, 2006). Only after this is achieved, the code's refactoring ensues to improve factors like the readability or its compliance with best practices and standards (Beck, 2015). At the same time, the tests are constantly executed to assure the functionality is not negatively affected by the refactoring.

As stated previously, due to the emphasis on small tasks and incremental modifications (Williams et al., 2003), instead of comprehensive implementations, following TDD has not only implications on the test coverage, but also the software's design. Furthermore, the short test cycles (Janzen and Saiedian, 2005) resulting from the frequent succession of testing and productive coding gives the developer more timely feedback. Unit tests make up the majority of tests in TDD, however, also other types of tests, such as acceptance, integration or system tests can be utilized (Sangwan and Laplante, 2006).

To facilitate the intended frequent execution of tests without requiring too much of the developer's valuable time and attention, as it would be the case with manual performance, TDD is often used together with test automation in a continuous integration (CI) pipeline (Karlesky et al., 2007; Shahin et al., 2017). Whenever a new commit happens, a CI server runs all applicable tests, therefore assuring that the change did not induce new errors into the already existing code.

2.4 Test Driven Development in Big Data

As indicated in the introduction, the application of TDD to the BD domain is a promising approach to assure the quality when developing BD applications, with the use of microservices being proposed as the technical foundation (Staegemann et al., 2020b). This appears sensible, since TDD is, inter alia, based on breaking down the desired application into the smallest reasonable parts. Therefore, a rather monolithic approach would be against the philosophy. Microservices, however, facilitate such a modular design (Shakir et al., 2021). Harnessing microservices allows to create a separate service for each business functionality, which, in turn, now only allows for independent scaling, but also enables the developers to distribute the implementation across teams and always use the most effective technology stack for each situation, instead of using a homogeneous toolset.

Especially in highly demanding settings, such as in the BD domain, this can be a substantial advantage. Further, by applying TDD, it is rather easy to make changes to the application, e.g., by swapping, modifying, or adding components. Since there are pre-existing tests for all the functions, it is possible to directly check if the change caused any issues to the system or if it is still working as intended. This increases flexibility and quality, but also trust, which is important to avoid incorrect use of the BDA solution (e.g. only using it to try to justify their own preferential decisions instead of actually building them on the data), especially in highly dynamic business environments that require more frequent adaptations and are consequently also more prone to corresponding errors (Günther et al., 2017; Staegemann et al., 2020a).

All in all, when considering the quality assurance of BD applications, there should be a synergy between TDD and the use of microservices, giving the approach proposed in (Staegemann et al., 2020b) merit. Yet, right now there appear to be only theoretical considerations, which are still to be subjected to a feasibility check. Although this is beyond the scope of a singular, exemplary project, the publication at hand aims to provide initial insights into the topic that can be built upon in the future.

2.5 Docker

Docker is a platform used to build, deploy, and manage containerized applications. Docker provides an isolated environment for applications with the operating system and dependencies required to run that application, which makes it easier to deploy the application in any environment (Cito et al., 2017). Building separate containers for each microservice will allow for the independent development and scaling for any particular microservice. Usually, each microservice contains predefined API paths, which enable it to perform actions such as a status check of other microservices or running unit tests. Most of the data transfer and connection between microservices is done using API endpoints. Therefore, any container can be replaced by modifying the source code in that particular container while keeping the API endpoints the same.

3 THE EXEMPLARY TASK

To explore the practical application of TDD in the BD domain, it is necessary to find a suitable and realistic task whose findings can be generalized at least to some extent. Therefore, for the publication at hand, the development of a movie recommendation system was chosen. Since the provisioning of recommendations is a typical big data use case (Bansal and Baliyan, 2019), this application can be seen as a rather dynamic scenario (Staegemann et al., 2020a), and the implementation of such a system can be easily broken down into small parts, it seems perfectly suitable for the expressed purpose. However, it has to be emphasized, that the developed system is not intended for productive use and the scientific interest is the primary motivator. Therefore, the application of the TDD methodology is also more important than specific choices as for specific programming languages or certain data sources.

To assure a certain degree of complexity, there are several functionalities that shall be implemented. The primary function is the visualization of information regarding movie data for a time frame chosen by the user, namely the best and worst movies by rating, the number of movies produced and the movie distribution by genre. Further, the user can have a synopsis of a chosen movie displayed. The synopsis is also used as input for a generator that provides the user with a number of tags that characterize the movie. Moreover, a recommender engine informs the user which movies might be of interest for them. Corresponding to the topic of the publication at hand, the development was to be conducted in a test driven manner, allowing for a continuous monitoring of the application's quality by applying CI. Since for TDD in a BD setting the use of microservices appears to be the most sensible choice (Staegemann et al., 2020b), the design and the opportunities provided by the system and its architecture are also influenced by the decision for TDD.

While, through the visualizer, the different capabilities are combined to provide the user with all the relevant information, the tag predictor and the recommendation engine shall also be able to be used as independent applications. This highlights the modularity of the microservice approach, allowing to utilize individual functionalities as building blocks in different contexts, adding a great degree of flexibility for the developers.

4 THE IMPLEMENTATION

For building the microservices, Python and Javascript were used, while the PostgresSQL database was chosen for data storing. The frontend user interface was designed with HTML and CSS. However, it was kept rather simple since a productive use was not the focus of the project. To containerize the application, Docker was used. For testing, the Pytest framework from Python's libraries was used and the entire project workflow is set up using Github Actions. As indicated in the previous section, the application as a whole is divided in three services that are each built from several microservices. Those main services are the *Visualizer*, the *Movie Tags Predictor* and the *Recommendation Service*. Further, there is a *Dashboard* service that allows to monitor the status of all the other microservices. The overall architecture is shown in Figure 1.

To obtain the data and assure data heterogeneity, several sources were used, namely the Internet Movie Database (IMDb - https://www.imdb.com/), Kaggle (https://www.kaggle.com/) and MovieLens (https://movielens.org/).

4.1 Visualizer

The visualizer is built from three microservices, *Data Collector*, *Database* and *Webpage*. The Data Collector gathers the data from the IMDb homepage and then sends it to the database after cleaning the data. Apart from the IMDb movie dataset, this service also uses the MovieLens dataset containing ratings given by users to the movie. In order to avoid downloading data every time a user opens a webpage,

the Database service is used to store the cleaned movie data that can be accessed by the Webpage service when needed. The latter contains a web page that users can interact with. Its landing page shows four different charts. Data shown in the charts can be modified based on different parameters given to the charts. For example, in the "Highest rated movies chart", users can change year and number of movies using a slider to display the top-rated movies according to their settings. Apart from this, users can also get more details about any particular movie shown in the chart by clicking on the bar related to that particular movie. Upon clicking, a new webpage opens with additional details of that movie such as synopsis, tags (from the tags predictor service), and recommendations (from the movie recommender service). In order to get the synopsis, the IMDb website is scraped.

Upon starting the Visualizer service, it runs integration tests. First of all, it checks whether all the required microservices are online. If this test passes, it checks the Database for a copy of older data. If old data is found, the Webpage microservice opens the port for users showing visualizations from old data and sends an API request in the background to the Data Collector microservice to download new data and updates the visualizations as soon as the new data is stored in the Database. If old data is not found in the Database, it sends an API request to the Data Collector and waits until the data is stored to the Database before opening a port for users.

4.2 Movie Tags Predictor

Since the actual movie tag prediction is not the main content of the project, but only a means for exploring the application of TDD, the implementation is not a new development and instead aligned with an already preexisting endeavor (Panda 2020). The data used in the Movie Tags Predictor service is obtained from the Kaggle website. Kaggle is an online community with a focus on data science and machine learning, where, inter alia, relevant datasets can be downloaded. The used dataset comprises approximately 14.000 movies and different attributes explaining the movies. An attribute containing the plot synopsis (`plot_synopsis`) is the primary independent attribute, and *Tags* is the dependent target variable. The plot synopsis data is used for training a Naive Bayes Classifier (Ting et al. 2011).

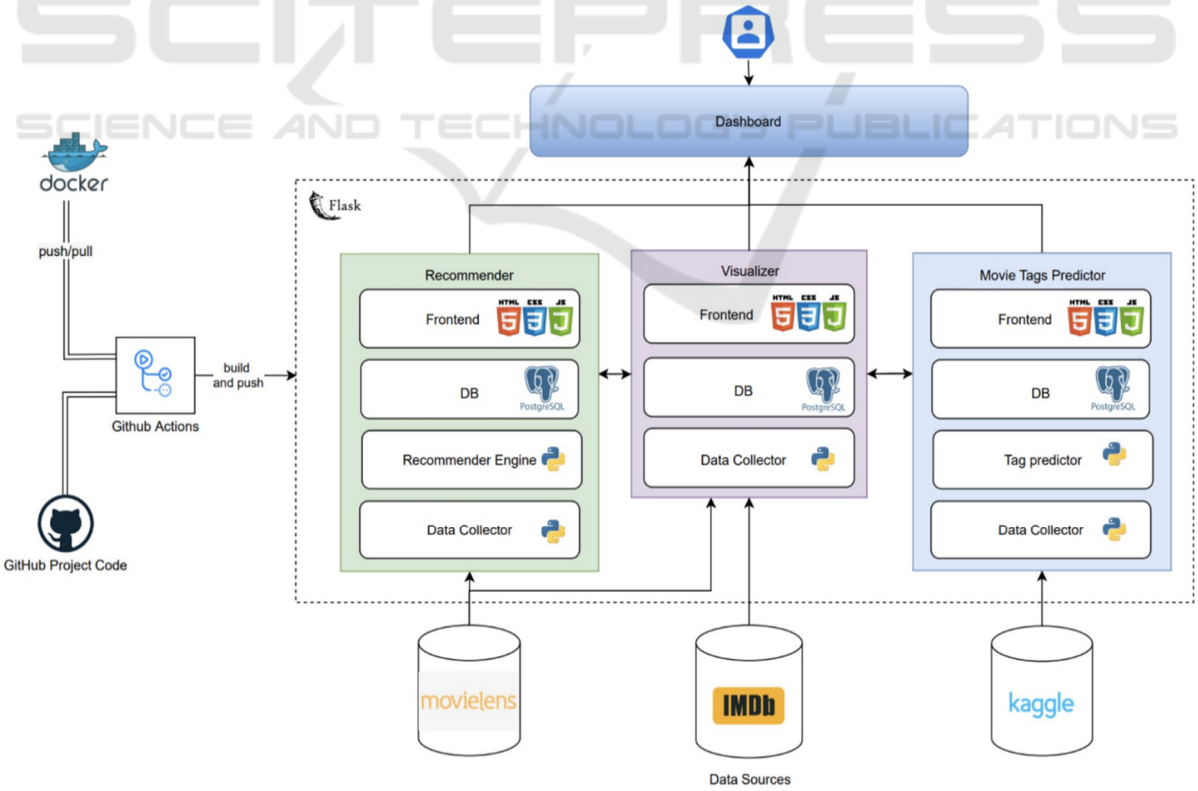

Figure 1: The implemented application's architecture.

The idea behind this service is to collect the plot synopsis of any movie and return hash tags that summarize the overall theme of the movie. When using the service not in the context of the comprehensive application but as a standalone solution, the user can input any text (in this case the plot of any movie) into a text field and hit the submit button to receive hash tags representing the movie, which are subsequently displayed below the text field.

The movie tags predictor service is divided into several other microservices such as *Processor*, *Predictor* and *Webpage*. Those microservices communicate with each other using REST APIs. When the movie tags predictor service is started, all three containers (processor, predictor, and web page) go live. During this process, the Processor downloads the labelled data (containing several movie variables with 'tags' as a target variable) from Kaggle and then performs some data preprocessing. Subsequently, the Predictor gets the processed data from the Processor using the API and builds a Machine Learning model on this data.

The Webpage service contains UI files such as HTML and CSS files but also the core Flask app logic. The Preprocessor service is programmed to source the labelled data from Kaggle and then perform text processing techniques on this main input variable that is `plot_synopsis` such as lemmatization, stop word removal and stemming. Once the synopsis is clean and ready, it is loaded into Postgres tables for storage purposes. At the first instance when an application is live, the Predictor service builds a machine learning model (Naive Bayes) on top of the processed data received from the Preprocessor and saves the model. When it receives test data i.e., every time a user inputs a new synopsis, it gets this data from the Preprocessor and gives it as an input to the model and outputs the generated prediction.

4.3 Recommendation Service

The movie recommender application suggests the user the next movie to watch after being provided the name of a movie they enjoyed as an input. While this is a huge simplification when regarding a productive use context, for the purpose of the conducted project it is sufficient, since the quality of the algorithms isn't the focus. The movie Recommendation Service consists of four microservices: *Data Collector*, *Database*, *Recommender*, and *Webpage*. The Data Collector downloads the data from MovieLens and sends it to the database after performing basic data cleaning tasks. The Database microservice uses PostgresSQL and stores the data from the Data Collector. For the Recommender microservice, the purpose is to give recommendations based on the input title of any movie the user liked. The recommendations are provided using collaborative filtering methods. The MovieLens dataset contains information about movies and ratings given by each user. With help of this information, a matrix is created, and the recommendations are provided using the k nearest neighbor method. Finally, the Webpage microservice provides an interface for users to get the recommendations for any movie. This microservice takes all the movies stored in the Database into account. Those are then used for auto-completion. When the user presses the recommend button, it sends an API request to the Recommender microservice and returns all the received recommendations. The results can be removed using the reset button.

4.4 Dashboard

The Dashboard is independent of all the other services. Its webpage shows all the microservices along with their status, whether they are online or offline. Each microservices can also be manually checked, using the check button. The dashboard is also designed to automatically re-check the status of every microservices after a specific time interval. Currently it is set to 5 Minutes. Apart from this, every 24 hours, the microservice will also update the data in the database by sending a request to the respective microservice. All the Webpage microservices that can be accessed by the user are hyperlinked and can be accessed from the dashboard.

4.5 The Testing

In the given project, unit testing and integration testing are applied, which corresponds to the basic testing techniques in TDD (Kum and Law 2006).

4.5.1 Unit Testing

Each of the developed microservices contains testing scripts to validate each container's results before it is executed. To illustrate, Figure 2 shows a simple test function that tests the functionality of the function *decontracted(phrase)*. It converts each contraction of an input phrase to its expanded, original form, thereby

```
def test_decontracted():
    phrase = "I won't do it"
    phrase = decontracted(phrase)
    assert phrase == 'I will not do it', 'decontracted function
not working'
```

Figure 2: Example of a unit test for text processing.

helping with text standardization. Accordingly, *test_decontracted()* is a test function that validates *decontracted(phrase)*.

A non-exhaustive list of further tests used in the course of the project comprises:

- checking whether the correct data is downloaded from a provided link
- checking whether the link is up and running and has no issues from the server on which it is hosted
- checking whether the connection to the database is established
- checking whether all the tables are created in the database to store the downloaded data
- checking whether the correct data is present in the tables created

4.5.2 Integration Testing

Integration tests are implemented in in all the Webpage microservices. In each of the three main services, the Webpage service will start after all the other microservices related to that main service have started. Therefore, it is important to have integration tests for the Webpage microservice. They will check if all the microservice required for the Webpage microservice to work properly are running. Then, using dummy data, the functionality is tested. If the results from these tests match the expectations, then the port for that main service is opened for the users. For example, the Webpage microservice in the Recommender service requires the Database, Collector, and Recommender microservices to be up and running. So first the Webpage microservice will check the status for these microservices. If all of them are running, then it will send a dummy movie name and try to get movie recommendations as a response. If such recommendations are received, the service is ready for users and the port for users will be opened.

5 DISCUSSION

While the application itself was scaled down in terms of aspects like usability and design, it is still adequate for the publication at hand's purpose since the underlying architecture is of sufficient complexity. Consequently, it can be considered to constitute a representative use case and, therefore, allows to answer the research questions.

As the preceding illustrations have shown, the first part of RQ1 can be positively answered, since the test driven methodology has proven to be feasible. The second part of RQ1 as well as RQ2 are discussed in the following. However, the corresponding answers are not sharply separated from each other since they oftentimes build upon each other.

The whole application has been developed in a highly modularized form, using microservices, with the corresponding tests being created before the actual implementation. Through this modularity it is also possible to scale and deploy the services independently of each other, which allows to react on volatile and imbalanced demands with a great degree of flexibility.

In the given example, there were two types of tests applied. Unit tests are used to assure that the separate functions are properly working, which is the pillar of a high-quality system. One of the findings was that for this purpose multiple assert statements should be avoided as they could lead to confusion where the test failed. The unit tests are complemented by integration tests to verify that the interplay between related services is playing out as it is expected. During the project, especially the latter have proven to be extremely helpful, since they led to the discovery of many errors that pertained to the interplay between the distinct services. Therefore, based on the project's findings, integration testing should not be neglected, and sufficient resources need to be devoted to it. The actual implementations were kept as simple as possible, making it easier to keep an overview. Further, adhering to naming conventions, using a consistent terminology for items with the same functionality, and choosing informative names have proven to be crucial for clarity. Additionally, during the project, no function dependencies were introduced between the tests, keeping each of them standalone. One issue that arose was that sometimes, small test cases may contain the actual code of the functionality that needs to be implemented and it is therefore somewhat being checked against itself. This, however, defeats the purpose of the TDD approach. For prospective researchers, exploring this topic further might therefore be a promising avenue.

According to the TDD methodology, whenever there were modifications in the development, all relevant tests were re-run (regression testing) to assure that no new errors were introduced. However, those tests only pertained to the pure operability. This means that it was only assured that everything generally works, e.g., the recommendation service is actually giving recommendations when being fed with a movie title. Yet, it was not regarded how good the quality of those movie recommendations actually was. For the future, putting emphasis on the quality assurance beyond the pure operability might be another promising research area.

6 CONCLUSION

Even though BD is applied in many areas of today's society, the testing of the corresponding systems is still rather immature. Since their quality is, however, one of the determining factors for the success of BDA, this is a huge issue, which is consequently addressed by numerous researchers, who explore new methods, tools and techniques to facilitate the quality assurance of BD systems. One rather recent proposition was the application of the TDD methodology to the big data domain. To assess its feasibility and gain practical insights concerning its actual application, as an exemplary case, a movie recommendation system was implemented in a test driven manner. The obtained insights and findings of this endeavour were discussed and, besides showing the approach's feasibility, recommendations for its actual application have been given. However, since this study is only based on a single case, its validity still needs to be strengthened by ancillary experiments. Additional avenues for future research have been identified in the investigation of the overlapping of test cases and productive code as well as in the facilitation of a stronger focus on the utilized algorithms' contentual quality instead of a pure operability perspective.

REFERENCES

Al-Mekhlal, M., and Ali Khwaja, A. (2019). "A Synthesis of Big Data Definition and Characteristics," in *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC),* New York, NY, USA. 01.08.2019 - 03.08.2019, IEEE, pp. 314-322 (doi: 10.1109/CSE/EUC.2019.00067).

Bahri, S., Zoghlami, N., Abed, M., and Tavares, J. M. R. S. (2019). "BIG DATA for Healthcare: A Survey," *IEEE Access* (7), pp. 7397-7408 (doi: 10.1109/ACCESS.2018.2889180).

Bansal, S., and Baliyan, N. (2019). "A Study of Recent Recommender System Techniques," *International Journal of Knowledge and Systems Science* (10:2), pp. 13-41 (doi: 10.4018/IJKSS.2019040102).

Beck, K. (2015). *Test-Driven Development*: *By Example*, Boston: Addison-Wesley.

Bronson, K., and Knezevic, I. (2016). "Big Data in food and agriculture," *Big Data & Society* (3:1) (doi: 10.1177/2053951716648174).

Chang, W. L., and Grady, N. (2019). "NIST Big Data Interoperability Framework: Volume 1, Definitions," *Special Publication (NIST SP),* Gaithersburg, MD: National Institute of Standards and Technology.

Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S., and Gall, H. C. (2017). "An Empirical Analysis of the Docker Container Ecosystem on GitHub," in *Proceedings of the 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR),* Buenos Aires, Argentina. 20.05.2017 - 21.05.2017, IEEE, pp. 323-333 (doi: 10.1109/MSR.2017.67).

Crispin, L. (2006). "Driving Software Quality: How Test-Driven Development Impacts Software Quality," *IEEE Software* (23:6), pp. 70-71 (doi: 10.1109/MS.2006.157).

Davies, K., Keet, C. M., and Lawrynowicz, A. (2019). "More Effective Ontology Authoring with Test-Driven Development and the TDDonto2 Tool," *International Journal on Artificial Intelligence Tools* (28:7) (doi: 10.1142/S0218213019500234).

Davoudian, A., and Liu, M. (2020). "Big Data Systems: A Software Engineering Perspective," *ACM Computing Surveys* (53:5), pp. 1-39 (doi: 10.1145/3408314).

Faitelson, D., Heinrich, R., and Tyszberowicz, S. (2018). "Functional Decomposition for Software Architecture Evolution," in *Model-Driven Engineering and Software Development*, L. F. Pires, S. Hammoudi and B. Selic (eds.), Cham: Springer International Publishing, pp. 377-400 (doi: 10.1007/978-3-319-94764-8_16).

Freymann, A., Maier, F., Schaefer, K., and Böhnel, T. (2020). "Tackling the Six Fundamental Challenges of Big Data in Research Projects by Utilizing a Scalable and Modular Architecture," in *Proceedings of the 5th International Conference on Internet of Things, Big Data and Security,* Prague, Czech Republic. 07.05.2020 - 09.05.2020, SCITEPRESS - Science and Technology Publications, pp. 249-256 (doi: 10.5220/0009388602490256).

Fucci, D., Erdogmus, H., Turhan, B., Oivo, M., and Juristo, N. (2017). "A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last?" *IEEE Transactions on Software Engineering* (43:7), pp. 597-614 (doi: 10.1109/tse.2016.2616877).

Goes, F. R., Meerhoff, L. A., Bueno, M. J. O., Rodrigues, D. M., Moura, F. A., Brink, M. S., Elferink-Gemser, M. T., Knobbe, A. J., Cunha, S. A., Torres, R. S., and Lemmink, K. A. P. M. (2020). "Unlocking the potential of big data to support tactical performance analysis in professional soccer: A systematic review," *European journal of sport science*, pp. 1-16 (doi: 10.1080/17461391.2020.1747552).

Günther, W. A., Rezazade Mehrizi, M. H., Huysman, M., and Feldberg, F. (2017). "Debating big data: A literature review on realizing value from big data," *The Journal of Strategic Information Systems* (26:3), pp. 191-209 (doi: 10.1016/j.jsis.2017.07.003).

Häusler, R., Staegemann, D., Volk, M., Bosse, S., Bekel, C., and Turowski, K. (2020). "Generating Content-Compliant Training Data in Big Data Education," in *Proceedings of the 12th CSEdu,* Prague, Czech Republic. 02.05.2020 - 04.05.2020, SCITEPRESS - Science and Technology Publications, pp. 104-110 (doi: 10.5220/0009513801040110).

Hazen, B. T., Boone, C. A., Ezell, J. D., and Jones-Farmer, L. A. (2014). "Data quality for data science, predictive analytics, and big data in supply chain management: An introduction to the problem and suggestions for research and applications," *International Journal of Production Economics* (154), pp. 72-80 (doi: 10.1016/j.ijpe.2014.04.018).

Herschel, R., and Miori, V. M. (2017). "Ethics & Big Data," *Technology in Society* (49), pp. 31-36 (doi: 10.1016/j.techsoc.2017.03.003).

Janzen, D., and Saiedian, H. (2005). "Test-driven development concepts, taxonomy, and future direction," *Computer* (38:9), pp. 43-50 (doi: 10.1109/MC.2005.314).

Ji, S., Li, Q., Cao, W., Zhang, P., and Muccini, H. (2020). "Quality Assurance Technologies of Big Data Applications: A Systematic Literature Review," *Applied Sciences* (10:22), p. 8052 (doi: 10.3390/app10228052).

Karlesky, M., Williams, G., Bereza, W., and Fletcher, M. (2007). "Mocking the Embedded World: Test-Driven Development, Continuous Integration, and Design Patterns," in *Embedded Systems Conference,* San Jose, California, USA. 01.04.2007 - 05.04.2007, UBM Electronics.

Keet, C. M., and Ławrynowicz, A. (2016). "Test-Driven Development of Ontologies," in *The Semantic Web. Latest Advances and New Domains*, H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S. P. Ponzetto and C. Lange (eds.), Cham: Springer International Publishing, pp. 642-657 (doi: 10.1007/978-3-319-34129-3_39).

Krylovskiy, A., Jahn, M., and Patti, E. (2015). "Designing a Smart City Internet of Things Platform with Microservice Architecture," in *2015 3rd International Conference on Future Internet of Things and Cloud (FiCloud 2015)*, I. Awan (ed.), Rome, Italy. 24.08.2015 - 26.08.2015, Piscataway, NJ: IEEE, pp. 25-30 (doi: 10.1109/FiCloud.2015.55).

Kum, W., and Law, A. (2006). "Learning Effective Test Driven Development - Software Development Projects in an Energy Company," in *Proceedings of the First International Conference on Software and Data Technologies,* Setúbal, Portugal. 11.09.2006 - 14.09.2006, SciTePress - Science and and Technology Publications, pp. 159-164 (doi: 10.5220/0001316101590164).

Lee, I. (2017). "Big data: Dimensions, evolution, impacts, and challenges," *Business Horizons* (60:3), pp. 293-303 (doi: 10.1016/j.bushor.2017.01.004).

Müller, O., Fay, M., and Vom Brocke, J. (2018). "The Effect of Big Data and Analytics on Firm Performance: An Econometric Analysis Considering Industry Characteristics," *Journal of Management Information Systems* (35:2), pp. 488-509 (doi: 10.1080/07421222.2018.1451955).

Musik, C., and Bogner, A. (2019). "Book title: Digitalization & society: A sociology of technology perspective on current trends in data, digital security and the internet," *Österreichische Zeitschrift für Soziologie* (44:S1), pp. 1-14 (doi: 10.1007/s11614-019-00344-5).

Nadareishvili, I., Mitra, R., McLarty, M., and Amundsen, M. (2016). *Microservice architecture*: *Aligning principles, practices, and culture*, Beijing, Boston, Farnham, Sebastopol, Tokyo: O´Reilly.

Nagorny, K., Lima-Monteiro, P., Barata, J., and Colombo, A. W. (2017). "Big Data Analysis in Smart Manufacturing: A Review," *International Journal of Communications, Network and System Sciences* (10:03), pp. 31-58 (doi: 10.4236/ijcns.2017.103003).

Panda, S. K. (2020). "Movie Tags Prediction Using Machine Learning Models.," available at https://medium.com/analytics-vidhya/movie-tag-s-prediction-using-machine-learningmodels-d5fde119db6d, accessed on Jan 24 2022.

Sangwan, R. S., and Laplante, P. A. (2006). "Test-Driven Development in Large Projects," *IT Professional* (8:5), pp. 25-29 (doi: 10.1109/MITP.2006.122).

Shahin, M., Ali Babar, M., and Zhu, L. (2017). "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," *IEEE Access* (5), pp. 3909-3943 (doi: 10.1109/ACCESS.2017.2685629).

Shakir, A., Staegemann, D., Volk, M., Jamous, N., and Turowski, K. (2021). "Towards a Concept for Building a Big Data Architecture with Microservices," in *Proceedings of the 24th International Conference on Business Information Systems,* Hannover, Germany/virtual. 14.06.2021 - 17.06.2021, pp. 83-94 (doi: 10.52825/bis.v1i.67).

Shull, F., Melnik, G., Turhan, B., Layman, L., Diep, M., and Erdogmus, H. (2010). "What Do We Know about Test-Driven Development?" *IEEE Software* (27:6), pp. 16-19 (doi: 10.1109/MS.2010.152).

Slaats, T., Debois, S., and Hildebrandt, T. (2018). "Open to Change: A Theory for Iterative Test-Driven Modelling," in *Business Process Management*, M. Weske, M. Montali, I. Weber and J. Vom Brocke (eds.), Cham: Springer International Publishing, pp. 31-47 (doi: 10.1007/978-3-319-98648-7_3).

Staegemann, D., Volk, M., Daase, C., and Turowski, K. (2020a). "Discussing Relations Between Dynamic Business Environments and Big Data Analytics," *Complex Systems Informatics and Modeling Quarterly* (23), pp. 58-82 (doi: 10.7250/csimq.2020-23.05).

Staegemann, D., Volk, M., Jamous, N., and Turowski, K. (2019). "Understanding Issues in Big Data Applications - A Multidimensional Endeavor," in *Proceedings of the Twenty-fifth Americas Conference on Information Systems,* Cancun, Mexico. 15.08.2019 - 17.08.2019.

Staegemann, D., Volk, M., Jamous, N., and Turowski, K. (2020b). "Exploring the Applicability of Test Driven Development in the Big Data Domain," in *Proceedings of the ACIS 2020,* Wellington, New Zealand. 01.12.2020 - 04.12.2020.

Staegemann, D., Volk, M., Lautenschlager, E., Pohl, M., Abdallah, M., and Turowski, K. (2021a). "Applying Test Driven Development in the Big Data Domain – Lessons From the Literature," in *2021 International*

Conference on Information Technology (ICIT), Amman, Jordan. 14.07.2021 - 15.07.2021, IEEE, pp. 511-516 (doi: 10.1109/ICIT52682.2021.9491728).

Staegemann, D., Volk, M., and Turowski, K. (2021b). "Quality Assurance in Big Data Engineering - A Metareview," *Complex Systems Informatics and Modeling Quarterly* (28), pp. 1-14 (doi: 10.7250/csimq.2021-28.01).

Ting, S. L., Ip, W. H., and Tsang, A. H. C. (2011). "Is Naïve Bayes a Good Classifier for Document Classification?" *International Journal of Software Engineering and Its Applications* (5:3).

Volk, M., Staegemann, D., Bosse, S., Häusler, R., and Turowski, K. (2020a). "Approaching the (Big) Data Science Engineering Process," in *Proceedings of the 5th International Conference on Internet of Things, Big Data and Security,* Prague, Czech Republic. 07.05.2020 - 09.05.2020, SCITEPRESS - Science and Technology Publications, pp. 428-435 (doi: 10.5220/0009569804280435).

Volk, M., Staegemann, D., Pohl, M., and Turowski, K. (2019). "Challenging Big Data Engineering: Positioning of Current and Future Development," in *Proceedings of the 4th International Conference on Internet of Things, Big Data and Security,* Heraklion, Crete, Greece. 02.05.2019 - 04.05.2019, SCITEPRESS - Science and Technology Publications, pp. 351-358 (doi: 10.5220/0007748803510358).

Volk, M., Staegemann, D., Trifonova, I., Bosse, S., and Turowski, K. (2020b). "Identifying Similarities of Big Data Projects–A Use Case Driven Approach," *IEEE Access* (8), pp. 186599-186619 (doi: 10.1109/ACCESS.2020.3028127).

Volk, M., Staegemann, D., and Turowski, K. (2020c). "Big Data," in *Handbuch Digitale Wirtschaft*, T. Kollmann (ed.), Wiesbaden: Springer Fachmedien Wiesbaden, pp. 1-18 (doi: 10.1007/978-3-658-17345-6_71-1).

Williams, L., Maximilien, E. M., and Vouk, M. (2003). "Test-driven development as a defect-reduction practice," in *Proceedings of the 14th ISSRE,* Denver, Colorado, USA. 17.11.2003 - 20.11.2003, IEEE, pp. 34-45 (doi: 10.1109/ISSRE.2003.1251029).

Zhu, L., Yu, F. R., Wang, Y., Ning, B., and Tang, T. (2019). "Big Data Analytics in Intelligent Transportation Systems: A Survey," *IEEE Transactions on Intelligent Transportation Systems* (20:1), pp. 383-398 (doi: 10.1109/TITS.2018.2815678).