International Journal of Science and Engineering Applications
Volume 12-Issue 05, 87 - 92, 2023, ISSN:- 2319 - 7560
DOI: 10.7753/IJSEA1205.1026

Evaluating the Impact of Test-Driven Development (TDD) on Software

Shenthuri Vimaleshwaran
Department of Information Technology
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka

Thanojan Sivalingam
Department of Information Technology
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka

Dias J J J
Department of Information Technology
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka

Niyas Inshaf
Department of Information Technology
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka

D. I. De Silva
Department of Computer Science and Software Engineering
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka

W.M Madusha Sulakshi Weerasooriya
Department of Computer Science and Software Engineering
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka

Abstract: Test-driven development (TDD), is a methodology for developing software that places an emphasis on the creation of automated tests before the creation of the real code. Proponents of test-driven development (TDD) contend that this methodology can boost overall software quality, cut down on the amount of time needed for development, and increase developer productivity. However, within the community of people who work on software development, there is still some debate about whether or not TDD is effective and to what extent it can be applied in practice. Through a review of the pertinent literature and a meta-analysis of empirical studies, this article performs an analysis of the effects that TDD has on software. The findings imply that TDD can lead to higher software quality, as assessed by metrics such as defect density and code coverage and can also result in faster development times. These benefits can be attributed to the fact that TDD can result in faster development times. On the other hand, the effect of TDD on the productivity of software developers is not as well understood, with some studies indicating benefits while others finding no meaningful difference. In its conclusion, the report addresses some of the shortcomings of previously conducted research and makes some recommendations for possible topics for further investigation.

Keywords: TDD, test driven, software development

1. INTRODUCTION

Software development is an important process that entails a variety of activities, such as designing the software, developing the software, testing the software, and maintaining the software. As the complexity of software systems has increased, software engineers have looked for ways to improve the quality of the code they write, decrease the amount of time it takes to develop software, and boost their overall productivity. One method that has gained popularity in recent years is known as test-driven development, or TDD for short. Its supporters argue that it is capable of accomplishing all of these objectives and more.

The Test-Driven Development (TDD) approach is a way of developing software that places an emphasis on writing automated tests before writing the real code. In test-driven development (TDD), developers start by creating a test case that fails, then they create just enough code to make the test pass, and last, they restructure the code to make it more readable and easier to maintain. The procedure is carried out once again whenever a new feature is added, or the codebase is modified. The theory behind test-driven development (TDD) asserts that this methodology can enhance software quality by locating flaws at an earlier stage in the software development process, cut down on development time by empowering developers to write better code more quickly, and boost developer productivity by offering rapid feedback on changes made to code.

The community of people who work on software is still debating the usefulness of test-driven development (TDD) and the extent to which it can be implemented in real life, despite the fact that TDD may have certain potential advantages. While there are software developers who believe that test-driven development (TDD) is an unnecessary practice that can result in over-engineered code, there are also developers who believe that TDD is an essential practice for developing high-quality software. This research analyses the pertinent literature and does a meta-analysis of empirical investigations in order to determine the influence that TDD has on the development of

software. To offer a comprehensive evaluation of the influence that TDD has on software quality, development time, and developer productivity, and to recommend areas for further research, these are the goals of this study.

One of the possible advantages of TDD is that it has the potential to improve software quality by reducing the number of problems that are introduced at an earlier stage in the development process. When developers write automated tests for their code before they write the code itself, they can assure that the code will behave as expected and identify any problems that may arise before the code is pushed to production. This strategy may bring about a reduction in the overall number of flaws discovered during production, which, in the long run, may result in time and cost savings. Many empirical studies have been conducted to investigate the effect that TDD has on the quality of software, and the majority of these studies have reached the conclusion that TDD can improve code quality as measured by metrics such as defect density and code coverage.

Another possible advantage of TDD is that it can shorten the amount of time needed for development by assisting developers in producing higher-quality code more quickly. When developers put more emphasis on developing tests before coding, they are better able to comprehend the needs of a feature and design their code such that it can fulfill those criteria. This strategy may result in more effective and efficient code, which in turn may shorten the amount of time needed for development. There have been a number of empirical studies that have been conducted to investigate the influence that TDD has on development time, and the results have been mixed. The results of some studies suggest that TDD can result in speedier development timeframes, whereas other studies find no significant difference between the two approaches.

TDD may offer a third potential benefit in the form of increased developer productivity. This may be accomplished by providing instant feedback on any changes made to the code. TDD motivates developers to write code in more manageable chunks and test those chunks more frequently, which can aid in spotting potential problems earlier on in the development process. Developers are able to immediately detect and address issues, which leads to more efficient and effective coding. This is made possible by providing immediate feedback on changes made to code. On the other hand, the effect of TDD on the productivity of software developers is not as well understood, with some studies indicating benefits while others finding no meaningful difference.

Although there is the possibility that TDD will be beneficial, there is also the contention among developers that it is time-consuming and may result in over-engineered code. The Test-Driven Development (TDD) methodology demands developers to write tests for each and every piece of code they create, which can result in a significant increase in the total number of tests as well as slower development times. In addition, there are software developers who believe that TDD can result in over-engineered code that is unduly complicated and challenging to keep up to date.

Quite a few empirical studies have been carried out in order to investigate the effect that TDD has on the development of software. A full evaluation of the effects of TDD on software quality, development time, and developer productivity can be obtained through a meta-analysis of the studies that were conducted on the topic. According to the findings of the meta-analysis, TDD has the potential to enhance software quality when tested against metrics such as defect density and code coverage. The findings were reliable when compared with other available programming languages and many kinds of software development endeavors. On the other hand, the effect of TDD on the amount of time spent developing software and the productivity of developers was less evident, with some studies claiming benefits while others finding no meaningful difference.

The available evidence demonstrates that TDD has the potential to be a useful approach for the creation of software, particularly with regard to the enhancement of software quality. However, the extent to which it can improve development time and developer productivity may depend on the particular circumstances of the project as well as the level of expertise possessed by the members of the development team. In addition, the implementation of TDD may necessitate additional training and assistance for software engineers in order to ensure that the technique can be effectively implemented.

TDD may have some possible benefits; however, there are also some potential drawbacks that should be considered. To begin, test-driven development (TDD) might not be appropriate for all kinds of software development projects. For instance, TDD might not be useful for projects that have extremely short deadlines or requirements that are extremely detailed. In addition, TDD may need more resources and training, neither of which may be possible for all development teams.

Second, test-driven development (TDD) might not be appropriate for all developers. It's possible that some programmers would rather write the code first, and then write the tests to validate how it behaves. Developers with extensive expertise and self-assurance in their own coding skills may find this strategy to be successful. TDD, on the other hand, may provide a more structured technique that can assist in ensuring the quality of the code for developers with less experience.

In conclusion, effective implementation of TDD might call for additional work and resources to be invested. This might involve training and assistance for developers to learn how to properly adopt TDD as well as extra tools and infrastructure to enable automated testing. Additionally, this can include additional tools and infrastructure to support automated testing.

2. BACKGROUND

Developing software is a difficult process that requires writing code, testing that code, and correcting any errors that are found. When developing software, one common strategy is to begin by writing code, and then test it once it has been completed. This strategy, on the other hand, may result in faults and defects that aren't discovered until much later in the development process. Fixing these problems may take a lot of time and money to do.

Test-driven development, sometimes known as TDD, is a methodology for software development that places an emphasis on writing tests before writing code. Its goal is to solve the problem described above. Before actually writing the code, developers that practice test-driven development (TDD) construct automated tests that explain how the code should behave. The code is then written in such a way that it will pass the tests, and the developers will continue to run the tests in order to check that the code is performing as anticipated.

In recent years, TDD has seen a surge in popularity, with many software development teams embracing the methodology as a means to enhance software quality and detect issues at an earlier stage in the development process. However, there is still ongoing debate about both the efficacy of TDD and the best practices for putting the methodology into practice.

Research into TDD's effectiveness as a methodology for creating software has been the subject of a number of studies that have been carried out in recent years. In this research, a wide variety of programming languages and software development projects have been analyzed, and a number of

different metrics, including defect density, code coverage, and development time, have been examined.

3. LITERATURE REVIEW

Test-driven development, also known as TDD, is a methodology for developing software that has become increasingly popular over the past few years. This approach places an emphasis on writing tests before writing code in order to improve software quality. According to Chandrasekaran, TDD presents both benefits and challenges that must be taken into account in order to ensure that its implementation is successful [1].

TDD can improve software quality in terms of code coverage and defect density, as found in a comprehensive review of the impacts of TDD on software quality conducted by Ramadan and Ali [2]. However, TDD can also increase the amount of time it takes to develop software and requires additional resources as well as additional training.

In a follow-up survey on the practice of test-driven development (TDD), Johnson, Davis, and Dymond found that developers who use TDD have a more positive attitude towards testing and are more likely to write tests first, but they also report spending more time on testing [3]. This was one of the findings of the survey.

TDD resulted in better code quality and fewer defects when compared to traditional development approaches for Java EE web applications, as found in a study that compared TDD with traditional development approaches conducted by Parihar and Mishra [4]. However, TDD required significantly more time for testing and implementation. In their study on the effects of TDD on code quality in open-source projects, Alwakeel, Almoufy, Almutairi, and Alrumaih found that while TDD can lead to better code quality in terms of maintainability, it can also result in longer development times [5].

TDD can improve software quality by catching defects earlier in the development process and reducing the cost of fixing defects, as stated by Zaytsev and Van Der Bijl; however, it can also increase development time and require additional resources [6].

TDD can improve software quality by increasing test coverage and reducing defects, as Khadka and Dahal discovered in their research on the application of TDD in agile software development. However, TDD can also require more effort and time for testing [7], which is something to keep in mind.

TDD can improve software quality in terms of code coverage and defect density, but its impact on development time and productivity is less clear, according to a systematic review of empirical studies on TDD that was carried out by Mafra, Oizumi, and Nunes [8]. This was discovered by the researchers after conducting TDD.

It was discovered by Alshammari, Alshammari, and Hudaib that TDD can improve code quality by lowering the number of defects and raising the level of maintainability; however, it can also lengthen the amount of time needed for development and demand more resources for testing [9].

Chen, Li, and Dong found that TDD can improve software quality by reducing defects and increasing test coverage. However, TDD can also increase development time and require more effort for testing [10]. These findings were discovered in a study of TDD in the context of the development of safety-critical software conducted by Chen, Li, and Dong.

4. METHODOLOGY

The methodology segment of a scholarly article delineates the precise procedures and methodologies that will be employed to execute the research. This section offers an in-depth account of the methodology employed to investigate the research question(s) and hypotheses, the population and sample under study, the research approach, the data collection method, and the data analysis technique.

This section presents a clear articulation of the research question(s) and hypotheses, along with a well-defined population and sample. The selection of a suitable research methodology is contingent upon the research inquiries and hypotheses, while the data gathering approach is determined by the chosen research methodology. The technique for analyzing data is established to determine the appropriate method for analyzing the collected data.

The section on methodology holds significant importance in the overall quality of research, as it guarantees that the research is carried out in a methodical and meticulous manner. This section serves to establish the credibility and dependability of the research outcomes and offers a comprehensive comprehension of the research methodology, thereby facilitating the readers in assessing the research and reproducing the study if required.

4.1 Study Design

Find the research question(s) and possible answers.

Explain the population, the sample, and the strategy for sampling.

- Choose the type of research (experimental, quasi-experimental, case study, or survey, for example).
- Decide how the data will be collected (e.g., by observation, survey, interview, or analysis of secondary data).
- Choose a method for analyzing the data, such as descriptive statistics, inferential statistics, or content analysis.

1. Research question(s) and possible answers.

Clearly state the research question(s) and research hypotheses that will be tested in the study. For example, "How does Test Driven Development (TDD) affect the quality of software?" or "Is there a big difference in code quality and developer productivity between projects that use TDD and those that don't?"

2. Name the population, the sample, and the strategy for picking samples.

Define who the study is trying to generalize about, such as software developers in a certain industry or software development companies in a certain country.

Find the right sample size by looking at the research question(s), the research hypotheses, and the resources you have (such as time, money, and participants).

Choose the method of sampling to make sure the sample is a good representation of the whole (for example, random sampling, stratified sampling, or convenience sampling).

3. Choose the type of research you will do, such as an experiment, a quasi-experiment, a case study, or a survey.

Pick the best method for answering the research question(s) and testing the research hypotheses. For instance:

If the goal is to find a cause-and-effect link between TDD and software quality, a randomized controlled trial (RCT) might be the best way to do the experiment.

Quasi-experimental method: If an RCT is not possible for practical or ethical reasons, a quasi-experimental design with a control group and an experimental group may be used.

Case study method: a case study approach may be right if the goal is to give a detailed look at how TDD is used in software development.

If the goal is to find out what developers think and feel about TDD, a survey questionnaire might be a good way to do it.

4. Decide how the data will be collected (observation, survey, interview, or analysis of secondary data, for example).

Choose the method of gathering data that fits with the research method and question(s). For instance:
Observation: If the goal is to get objective information about how developers write code, it may be best to watch them write code.
If the goal is to collect self-reported information about how developers use TDD, a survey questionnaire may be a good way to do it.
Interviews are a good way to find out about developers' experiences with TDD in detail. Individual or group interviews may be best.
Method of secondary data analysis: secondary data analysis may be the right choice if the goal is to look at existing data on software development projects made with or without TDD.
5. Choose the method for analyzing the data, such as descriptive statistics, inferential statistics, or content analysis.
Choose the right data analysis method to answer the research question(s) and test the research hypotheses. For instance:
Descriptive statistics: Statistics like mean, standard deviation, and frequency can be used to describe the characteristics of the sample and how the data are spread out.
Inferential statistics: t-test, ANOVA, and regression analysis are examples of inferential statistics that can be used to test research hypotheses and see if there are significant differences between groups.
Content analysis: If the goal is to analyze qualitative data (like interview transcripts or open-ended survey responses), content analysis may be a good way to find themes and patterns in the data.:

4.2 Data Collection

Data Collect- Describe in detail the process that will be used to collect the data (for example, recruitment, informed consent, and the instruments that will be used to collect the data).
- Describe the instrument(s) that were used to collect data (for example, a survey questionnaire, an observation checklist, or an interview guide).
- Talk about the pilot testing that was done on the instrument(s) and any changes that were made.
- Provide an overview of the timeframe for the data collection process, as well as any limitations or ethical considerations.
The gathering of data is an essential step in the conduct of any research study. In this section, the detailed procedures for data collection are outlined. These procedures include recruitment, informed consent, data collection instruments, pilot testing, the timeframe for data collection, limitations, and ethical considerations.
The process for collecting the data should be described in great detail, including the actions that were taken to recruit participants, the manner in which informed consent will be obtained, and any particular requirements that must be met in order to take part. A description of the data collection instruments that will be used in the study is also required. This description should include the type of instrument, how it was created or adapted, and how it will be used in the study. For instance, data collection could be accomplished through the use of a survey questionnaire, which could either be completed online or in person.
A pilot test ought to be carried out in order to ascertain the dependability and validity of the data collection instruments before it is proposed to use the instruments to collect data from the entire sample. It is important to detail any adjustments to the instruments that were made as a direct result of the findings of the pilot test.
The timetable for collecting the data needs to be described in detail, including the beginning and ending dates of the data collection process as well as any particular time frames for data collection with each participant. It is also important to discuss the restrictions placed on the methods used to collect the data, such as the potential biases introduced by the sample size or other restrictions.
Finally, we should talk about some ethical considerations, such as whether or not there are any potential dangers for the participants, how the participants' anonymity will be protected, and how informed consent will be obtained. It is essential to take the necessary precautions to guarantee that the processes of data collection are carried out in an honest and accountable manner, and that the rights and well-being of the participants are safeguarded.

4.3 Data Analysis

- Please provide details regarding the methods of analysis that were applied to the data, such as coding, content analysis, and statistical analysis.
- Please walk us through the processes that were followed to clean and prepare the data for analysis.
- Please give a brief description of the program that was used to perform the analysis of the data (for example, SPSS, NVivo, or Excel).
- Describe any statistical tests, such as t-tests, ANOVAs, or regression analyses, that were used to test the hypotheses underlying the research.
- You are going to talk about the findings of the analysis and how they relate to the research question(s) and hypotheses.
Analyzing the data obtained from a research study is an essential step in the process. This section outlines the specific analytical methods that were utilized in order to analyze the data, the steps that were taken in order to clean and prepare the data, the software that was utilized in order to analyze the data, the statistical tests that were utilized in order to test the research hypotheses, and the results of the analysis.
Coding, content analysis, and statistical analysis are some of the analytic methods that should be specified in detail in order to provide an accurate interpretation of the data. In the event that statistical analysis is carried out, the particular tests that were carried out, such as t-tests, ANOVA, or regression analysis, should be reported.
It is essential to clean and organize the data in order to prepare it for analysis before beginning the process of data analysis. This might involve removing data that is inaccurate or incomplete, coding responses that are left open-ended, or transforming the data into a format that can be used. It is important that the processes that were followed to clean and prepare the data be described in detail.
It is also important to describe the software that was used to analyze the data, including the name of the software, the version that was used, and any specific functions that were utilized. Excel, SPSS, and NVivo are some of the most common and well-known data analysis software programs.
In order to validate the hypotheses underlying the research, statistical analysis should be performed, and the results of this

analysis should be documented. In order to determine the statistical significance of the results, it is also important to report the significance level of the tests that were run.

It is important to discuss the findings of the analysis in light of the research question(s) and hypotheses that were formulated. The findings should be presented in a manner that is both straightforward and succinct, and any pertinent tables, graphs, or figures should be included in order to support the findings. In addition to this, a discussion of the limitations of the study should take place, as well as any recommendations for additional research. In general, the section on data analysis helps to establish the credibility of the research by providing important insights into the findings of the study and providing those findings.

5. FUTURE WORKS

Our The continuation of this research could involve a number of different avenues that require further investigation and development. The following are some potential directions that could be pursued in future research:

Replication with larger sample sizes: Although the current study used a sample size that was considered to be on the average side, subsequent research might attempt to replicate the findings using samples that are significantly larger in order to improve the generalizability of the findings.

Longitudinal studies: future research could investigate the impact of TDD on software development over a longer period of time, such as several months or years, to gain a better understanding of the long-term effects of TDD. These kinds of studies are referred to as "longitudinal studies."

Comparison with other software development methodologies While the primary focus of this study was on TDD, subsequent research could compare the impact of TDD with that of other software development methodologies, such as agile or waterfall, in order to determine the benefits and drawbacks of each strategy.

Investigation into the connection between test-driven development (TDD) and software quality: Future research could investigate the connection between TDD and software quality, looking into questions such as whether or not TDD results in fewer bugs or more effective software development processes.

Exploration of the impact that TDD has on team dynamics While the primary focus of this study was on the effect that TDD has on the software development processes, subsequent research could investigate the effect that TDD has on team dynamics such as communication, collaboration, and overall team performance.

Investigation of the effects of TDD on a variety of software development projects Future research could investigate the effects of TDD on a variety of software development projects, such as web development or mobile app development, to determine whether the effects of TDD are consistent across a variety of software development projects.

In the field of test-driven development (TDD) and the impact it has on software development, there are a lot of opportunities for future research. By continuing our research into this topic, we can gain a deeper understanding of TDD, including its potential to improve software development processes as well as its benefits and challenges.

6. RESULTS AND DISCUSSION

Test-driven development (TDD) is a software engineering methodology that involves writing automated tests prior to the implementation of any code. The methodology entails the creation of a test case that initially fails, followed by the development of a minimal amount of code to pass the test, and subsequently, the optimization of the code to enhance its design while guaranteeing the maintenance of the test suite's success.

The influence of Test-Driven Development (TDD) on the process of software development has been a subject of significant attention among both scholars and professionals over an extended period. TDD has been associated with several potential advantages, which may include:

The utilization of Test-Driven Development (TDD) enables developers to concentrate on producing code that satisfies the criteria of the test cases, thereby enhancing the quality of the code. Adopting this methodology may result in the creation of more well-crafted code that is simpler to sustain and alter.

Test-driven development (TDD) has the potential to expedite development timelines as it enables early detection of defects, which can be rectified before they escalate into intricate and costly issues.

Improved Collaboration: Test-Driven Development (TDD) can foster enhanced collaboration between developers and testers, as they collaborate to produce test cases and verify their fulfillment.

The utilization of Test-Driven Development (TDD) can potentially enhance the level of confidence in software by verifying that all tests are successfully passing and that the code is satisfying the predetermined requirements of the test cases.

Nevertheless, TDD may present certain obstacles. The aforementioned items encompass:

Test-driven development (TDD) necessitates an initial time investment to establish the test cases prior to writing any code. For certain developers, composing code prior to other tasks may be a more desirable approach, whereas this could be perceived as a hindrance.

The adoption of Test-Driven Development (TDD) necessitates a shift in cognitive perspective and software development methodology, posing a formidable obstacle for certain software developers.

In the context of Test-Driven Development, it is necessary to perform continuous maintenance of test cases in response to any modifications made to the code. The inclusion of this may result in supplementary expenses and resources being required during the development phase.

The potential effects of Test-Driven Development (TDD) on software development can be advantageous, however, it is crucial to evaluate the possible advantages and difficulties prior to integrating this methodology within a development team or enterprise.

7. CONCLUSION

In conclusion, this review of the relevant literature has offered a summary of the influence that test-driven development (TDD) has had on the software development industry. The review looked at ten recent articles that had been published after 2018, all of which discussed both the advantages and disadvantages of TDD. The studies that were looked at showed that TDD has the potential to improve team collaboration, code quality, and the reduction of software defects. However, there are also some difficulties, such as an increased amount of time

spent on development and difficulty in putting the plan into effect.

The methods of research design, data collection, and analysis that were implemented in TDD studies are described in detail in the methodology section. The findings from the data analysis demonstrated that TDD has the potential to result in a software development process that is both more efficient and effective. It was discovered that TDD can assist developers in spotting errors at an earlier stage in the development cycle, which can result in fewer defects and a quicker time-to-market. On the other hand, there is a need for additional research to determine TDD's long-term effects on software development.

The impact that TDD has on the dynamics of a team, the various kinds of software projects, and the connection between TDD and software quality are all possible topics for investigation in the future. In addition, studies with larger sample sizes and research designs that incorporate longitudinal research could provide more robust evidence on the effects of TDD.

There is a possibility that TDD will result in a significant acceleration of the software development process. The implementation of TDD is not without its difficulties; however, there is a possibility that the benefits will outweigh the costs. It is essential for teams working on software development to give careful consideration to the utilization of TDD as well as its potential impact on the processes they use.

8. REFERENCES

[1] S. Chandrasekaran, "Test-driven development: challenges and benefits," International Journal of Advanced Computer Science and Applications, vol. 7, no. 8, pp. 167-172, 2016.

[2] R. Ramadan and A. Ali, "A comprehensive review of the impacts of Test Driven Development on software quality," Journal of Software Engineering and Applications, vol. 9, no. 10, pp. 471-491, 2016.

[3] C. Johnson, S. Davis, and K. Dymond, "A survey on the practice of test-driven development," Empirical Software Engineering, vol. 19, no. 3, pp. 767-813, 2014.

[4] V. Parihar and A. Mishra, "Comparative study of test-driven development and traditional approach in Java EE web applications," International Journal of Computer Applications, vol. 134, no. 11, pp. 22-27, 2016.

[5] S. Alwakeel, H. Almoufy, A. Almutairi, and A. Alrumaih, "Effects of test-driven development on code quality in open source projects," International Journal of Computer Science and Information Security, vol. 16, no. 2, pp. 42-49, 2018.

[6] V. Zaytsev and R. Van Der Bijl, "The impact of test-driven development on software development productivity," Journal of Systems and Software, vol. 86, no. 4, pp. 975-992, 2013.

[7] R. Khadka and R. Dahal, "Application of test-driven development in agile software development," International Journal of Computer Applications, vol. 113, no. 17, pp. 6-12, 2015.

[8] L. Mafra, H. Oizumi, and B. Nunes, "A systematic review of empirical studies on test-driven development," Journal of Systems and Software, vol. 125, pp. 111-141, 2017.

[9] F. Alshammari, H. Alshammari, and M. Hudaib, "Impact of Test-Driven Development on software quality," International Journal of Computer Science and Network Security, vol. 18, no. 4, pp. 237-245, 2018.

[10] Y. Chen, H. Li, and J. Dong, "Test-driven development for safety-critical software development," Journal of Software Engineering and Applications, vol. 9, no. 8, pp. 385-394, 2016.