# Towards Accountability Driven Development for Machine Learning Systems *

Chiu Pang Fung[1,3], Wei Pang[1,2], Iman Naja[2], Milan Markovic[2], and Peter Edwards[2]

[1] School of Mathematical and Computer Sciences, Heriot-Watt University
Edinburgh, EH14 4AS, UK
w.pang@hw.ac.uk
[2] School of Natural and Computing Sciences, University of Aberdeen
Aberdeen, AB24 3UE, UK
{iman.naja, milan.markovic, p.edwards}@abdn.ac.uk
[3] School of Computing, University of Leeds, Leeds, LS2 9JT, UK
C.P.Fung@leeds.ac.uk

**Abstract.** With rapid deployment of Machine Learning (ML) systems into diverse domains such as healthcare and autonomous driving, important questions regarding accountability in case of incidents resulting from ML errors remain largely unsolved. To improve accountability of ML systems, we introduce a framework called Accountability Driven Development (ADD). Our framework reuses Behaviour Driven Development (BDD) approach to describe testing scenarios and system behaviours in ML Systems' development using natural language, guides and forces developers and intended users to actively record necessary accountability information in the design and implementation stages. In this paper, we illustrate how to transform accountability requirements to specific scenarios and provide syntax to describe them. The use of natural language allows non technical collaborators such as stakeholders and non ML domain experts deeply engaged in ML system development to provide more comprehensive evidence to support system's accountability. This framework also attributes the responsibility to the whole project team including the intended users rather than putting all the accountability burden on ML engineers only. Moreover, this framework can be considered as a combination of both system test and acceptance test, thus making the development more efficient. We hope this work can attract more engineers to use our idea, which enables them to create more accountable ML systems.

**Keywords:** Behaviour Driven Development · Machine Learning · Model Card · Accountability

* This research is supported by the RAInS project funded by EPSRC (EP/R033846/1). Corresponding Author: Wei Pang (w.pang@hw.ac.uk)

## 1 Introduction

There are many definitions of accountability in AI systems, and following our previous work [10], we define accountability as follows: "The ability to inspect, review or otherwise interrogate an AI system with the goal of (*i*) making processes associated with each of its life cycle stages transparent; (*ii*) demonstrating compliance with hard laws (*i.e.* laws and regulations), and soft laws (*i.e.* standards and guidelines); and (*iii*) aiding investigations into the cause(s) of failure or erroneous decisions and supporting the identification of responsible parties." [10]. This definition can be adopted to the narrower scope - ML systems. Currently ML system development is a straight forward process [17, 4] (Figure 1). In this kind of processes, the relevant personnel within the project are in charge of different tasks based on their roles. Decision makers care about the purpose; stakeholders are concerned with the business value of the potential ML application; domain experts explain what the data means; data scientists and ML engineers focus on technical perspective, including data processing, model training and evaluating. This division of labor may cause ambiguity for accountability of ML systems. Many data scientists and ML engineers are keen on fulfilling the system performance requirements rather than considering the accountability issues. The Model Card framework [9] offers a standardized documentation procedure to capture useful information such as model purpose, owners, data information, algorithms' details. But the information they collected is not comprehensive regarding to accountability of ML systems. The lack of information about system's explainability, transparency, fairness and performance makes it difficult to audit and investigate the potential failures of ML systems.

ML system's life cycle can be divided into four high level stages: *Design*, *Implementation*, *Deployment*, and *Operation* & *Maintenance* [10]. In this paper, we focus on illustrating how to ask the ML system to generate those information in our ADD framework in the implementation stage, and we also give examples based on a simulation using our framework to develop an auto decision making system for mortgage applications.

## 2 Background and related work

Artificial Intelligence (AI) systems which employ ML models are being put to use in various applications affecting our daily lives. However, few companies and organizations are dedicating resources to mitigate AI risks despite some of them having to increasingly manage such risks related to AI [8]. In AI/ML research, the same situation happens that many researchers focus on improving the performance of ML algorithms rather than looking into the accountability of these systems. Research about ML accountability has been emerging in recent years. Hajian *et al.* [5] propose an idea on how to process raw data to prevent discrimination in AI based intrusion and crime detection. Datta *et al.* [3] introduce the Quantitative Input Influence (QII) metric to improve the transparency of

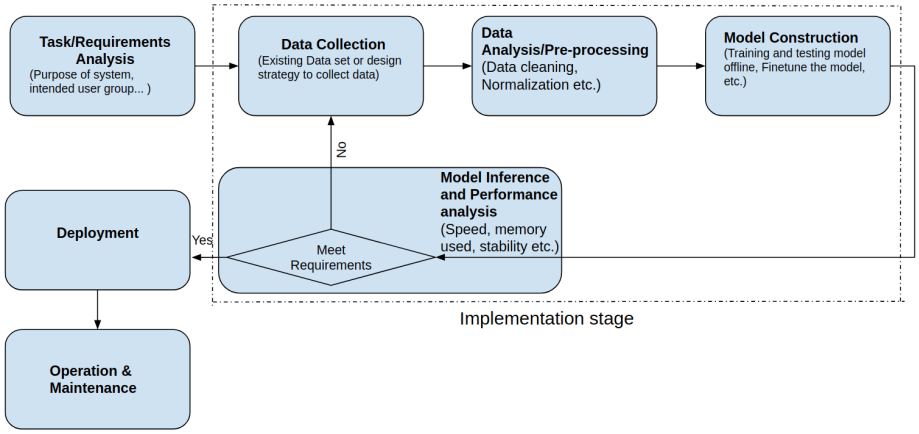


**Fig. 1.** A typical Machine Learning system Development Workflow.

decision-making systems by measuring the influence of the input features. Bach *et al.* [1] propose Layer-Wise Relevance Propagation to explain deep neural networks. Ribeiro *et al.* [13] propose the called Local Interpretable Model-Agnostic Explanations (LIME) algorithm to locally explain predictions of classifiers and regressors by training an interpretable model to approximate the original model. Afterwards, Shrikumar *et al.* propose [14] Deep Learning Important FeaTures (DeepLIFT). Finallu, Lundberg *et al.* [7] propose the SHapley Additive exPlanations (SHAP) method. These articles started the upsurge of ML's explainability research. All the above researches make it feasible for the information of ML system's explainability to be collected. For the other aspect, transparency, Mitchell *et al.* introduce Model Card [9] to create documentations for ML models. Their framework allows people to look deeply into the model's development and its performance.

On the other side, in software development there is a methodology called Behavior Driven Development (BDD) [12, 6, 16], which is developed from Test Driven Development (TDD) [2, 11]. Same as TDD, BDD is still a test driven development approach, but BDD designs user story scenarios instead of designing test cases; it uses natural language to describe software system's behaviours, forcing the behaviours fulfill the corresponding requirements in all scenarios to test the system. As a test-first approach, in BDD the scenarios are designed at the beginning of the development when there is no existing code, so that the first test must fail until developers start to write code. The use of natural languages to describe user story scenarios in BDD creates a bridge among software designers, developers and users, encouraging all the development team members such as developers and non-technical collaborators to work together with less technical barriers. Inspired by BDD, we use natural language to describe the ML system's behaviours in the form of user story scenarios that are designed according

to both technical and accountability requirements, making the ML system fulfill those requirements. That means the ML system must generate information about its performance, explainability, transparency *etc.* For example, there is a performance requirement-*Accuracy*, when the ML system passes the corresponding scenario test, it must generate information on accuracy. Those information can be collected to improve system's accountability. If there is a failure about the system's accuracy after its deployment, we can trace back what kind of accuracy test was done and what condition had be set in the corresponding testing scenario, and this information will help the failure investigation and auditing purpose.

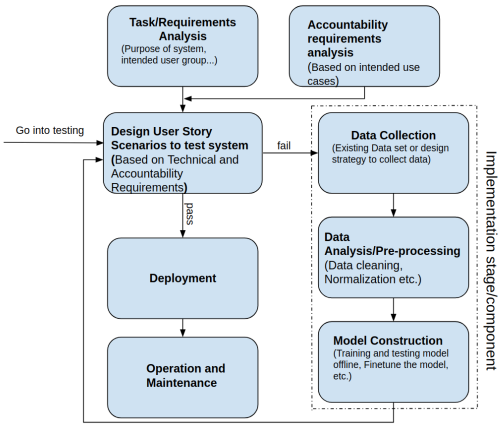


**Fig. 2.** Machine Learning System Development workflow in ADD framework. Three blocks (Data Collection, Data Analysis/Pre-processing and Model Construction) in the dotted rectangle are considered as an implementation component, which creates the ML system. The system can go to the deployment stage if it passes all the scenarios tests or it must be improved if it fails the test in any of those scenarios.

## 3 ML System Development with ADD

### 3.1 The workflow of ADD

In our framework (Figure 2), starting from the Task/Requirements analysis, the development team and potential users should not only discuss technical requirements (*e.g.* performance requirements such as accuracy and efficiency), but also discuss the accountability requirements according to model's transparency, explainability, fairness and document all the requirements. Then based on the requirements, user stories and test scenarios are designed to start the testing. Similar to BDD, the first test must fail as no ML system has been developed. Afterwards, the developers start to implement the ML system and test it again,

until the developed system passes all the tests. Eventually the system can be considered to have fulfilled all the requirements and ready to be deployed. In this case, the testing in our framework can be considered combining system test and acceptance test.

### 3.2 Technical and Accountability Requirements

Technical requirements normally refer to performance characteristics. Some of these requirements originate from potential users, such as performance requirements, safety requirements, hardware architecture requirements. However, to improve the entire technical requirements, the development team should proactively propose some extra technical requirements in ML Systems, such as metrics for specific algorithms that do not violate customer's requirements. For example, a customer asks for a 90% accuracy for a classifier, and the development team should offer a confusion matrix analysis. According to transparency, a document which is extended from Model Card [9] to collect more information should be established to record details of the entire model development process such as requirement analysis and useful information generated when the system passes the tests. Corresponding to explainability, there should be requirements to explain how and why this output is generated by the system. For example, in the auto decision making system for mortgage applications, when an output is obtained, the weights of all the features (features in this paper refer to applicant's income, occupation, age *etc.*) of the corresponding applicant should be listed and recorded. Regarding fairness, it can be proposed in accordance with law that the system cannot be discriminatory based on ethnicity or gender *etc.*

### 3.3 Design user story scenarios

In BDD, the user story scenarios tests relate to some sub-modules of the whole software system. For example, in a supermarket management system, selling a product relates to the cash flow management module and stock management module, so that the scenarios descriptions are more comprehensive. In ADD, we consider three blocks (Data Collection, Data Analysis/Pre-processing and Model Construction. Figure 2) inside the dotted rectangle as a component in development. That will simplify the design. For example, we can ask this component to provide data distribution information rather to ask the Data Analysis/Preprocessing block to do so. Not only the normal requirements, but also the edge use cases must be covered in the scenario design. For example, when there is an edge use case that an applicant with no features goes into the system, an error message "***Wrong input given, can not create proper output***" should be given rather a normal output as "***Mortgage application is approved***" or "***Mortgage application is declined***". Also, all details of scenarios design must be recorded as they contain what situations have been considered for the use of the system which are useful for accountability.

### 3.4 Syntax for describing scenarios and system's behaviour

The ML technical terms may be difficult to understand, and obscure statistical methods, evaluation standards and metrics keep non-technical personnel out of the loop. These non-technical collaborators may have better understanding of other accountability factors, such as laws and ethical requirements, which can help improving a system's accountability. Removing this technical barrier is necessary to gain contributions for accountability from both technical and non-technical personnel in the development stages. In ADD, a method similar to BDD is used, and we use natural language to describe user story scenarios and the behaviors of the system. We give two examples below to explain how to transfer one fairness requirement into scenarios and how to describe system's behaviour in that scenario. In the second example, the system is forced to generate explanation information. (It is pointed out that in these examples, we just want to demonstrate how ADD can be used, and we do not consider the profit requirement of the intended users, the banks. We also point out that the below examples are the starting point to engage all parties, and they will be further refined after discussions among related people, including users and ML developers/designers. The final version of the scenarios may be produced after several iterations until all parties reach a consensus. If a consensus cannot be reached, the information on disagreement also needs to be recorded. )

**Title**: *Producing non-discriminatory output*

**As a** *Bank.*
**I want** *the system that produces fair outputs for different applicants regardless of their ethnicity.*
**so that** *the System is not racially biased.*

**Scenario 1**: *The System should produce fair outputs for applicants from different ethnic groups.*
**Given** *that a mortgage application approved for applicant A,*
**When** *another applicant B, with the same features but different ethnicity applies for the same mortgage amount,*
**Then** *the mortgage application should be approved for applicant B.*

**Scenario 2**: *The system should produce fair outputs for applicants from different ethnic groups.*
**Given** *that a mortgage application from applicant A is declined,*
**When** *another applicant B with the same features but different ethnicity is applying for the same mortgage amount,*
**Then** *mortgage application from applicant B should be declined.*

**Title**: *Proving explanations for outputs*

**As a** *Bank.*

**I want** *the system that provides explanation for every mortgage application decision.*
**so that** *we know why the output is produced and can explain it to our clients.*

**Scenario 1**: *The system should provide explanation for a successful application.*
**Given** *an applicant with all necessary features,*
**When** *mortgage application is approved,*
**Then** *the system should create a report with all the weights corresponding to the features of the applicant.*

**Scenario 2**: *The system should provide explanation for a failed application.*
**Given** *an applicant with all necessary features,*
**When** *mortgage application is declined,*
**Then** *the system should create a report with all the weights corresponding to the features of the applicant.*

## 4 Discussion and Future work

There is an urgent need to improve ML system's accountability during its development stage. In this paper, we briefly introduce our ADD framework to develop ML systems and explain how the ADD can facilitate accountability information capture regarding the system's performance, explainability, fairness and transparency. We believe ADD can help reduce misunderstanding among users, ML system designers and developers by engaging them and confirming accountability requirements.

In the future, we are going to further improve this framework by making a deeper study in the requirements and scenarios design sections (Sections 3.2, 3.3), developing an example model and giving a full report by extending the Model Card framework [9]. Further more, we point out that ADD is a generic methodology for facilitating accountability in ML, and therefore, we will extend our framework to the whole life cycle of ML systems, making it more widely used. Finally, we are considering developing a Cucumber-like [15] (cucumber is a tool that supports BDD.) tool that is specifically for ML system development.

## References

1. Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PloS one **10**(7), e0130140 (2015)

2. Beck, K.: Test-driven development: by example. Addison-Wesley Professional (2003)
3. Datta, A., Sen, S., Zick, Y.: Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In: 2016 IEEE symposium on security and privacy (SP). pp. 598–617. IEEE (2016)
4. Google: Machine learning workflow. https://cloud.google.com/ai-platform/docs/ml-solutions-overview, online; Accessed Mar 25, 2021
5. Hajian, S., Domingo-Ferrer, J., Martinez-Balleste, A.: Discrimination prevention in data mining for intrusion and crime detection. In: 2011 IEEE Symposium on Computational Intelligence in Cyber Security (CICS). pp. 47–54. IEEE (2011)
6. Haring, R., de Ruiter, R.: Behavior driven development: Beter dan test driven development. Java Magazine p. 29 (2011)
7. Lundberg, S., Lee, S.I.: A unified approach to interpreting model predictions. arXiv preprint arXiv:1705.07874 (2017)
8. Mckinsey: The state of ai in 2020. https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/global-survey-the-state-of-ai-in-2020, online; Accessed Mar 28, 2021
9. Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I.D., Gebru, T.: Model cards for model reporting. In: Proceedings of the conference on fairness, accountability, and transparency. pp. 220–229 (2019)
10. Naja, I., Markovic, M., Edwards, P., Cottrill, C.: A semantic framework to support ai system accountability and audit (2021)
11. Newkirk, J., Vorontsov, A.A.: Test-driven development in Microsoft .Net, vol. 1. Microsoft Press Redmond, WA (2004)
12. North, D.: faster organizations, faster software. http://dannorth.net/introducing-bdd, online; Accessed Mar 30, 2021
13. Ribeiro, M.T., Singh, S., Guestrin, C.: " why should i trust you?" explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 1135–1144 (2016)
14. Shrikumar, A., Greenside, P., Kundaje, A.: Learning important features through propagating activation differences. In: International Conference on Machine Learning. pp. 3145–3153. PMLR (2017)
15. SmartBear: what is cucumber. https://cucumber.io/docs/guides/overview/, online; Accessed Mar 28, 2021
16. Solis, C., Wang, X.: A study of the characteristics of behaviour driven development. In: 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications. pp. 383–387. IEEE (2011)
17. Wang, M., Cui, Y., Wang, X., Xiao, S., Jiang, J.: Machine learning for networking: Workflow, advances and opportunities. IEEE Network **32**(2), 92–99 (2017)