International Journal of Academic Engineering Research (IJAER)
ISSN: 2643-9085
Vol. 8 Issue 5 May - 2024, Pages: 40-48

From Software Development Lifecycle to Machine Learning Lifecycle

Marine Alraqdi, Shahad Alqureisha, Reham Alzowaid, Munerah Altalasi, Maram Alsuhaibani, Aisha Al-Maalwy and Omer Alrwais

College of Computer and Information Sciences,
King Saud University, Riyadh, Kingdom of Saudi Arabia

Abstract*- Development life cycles are important for developing any system since they include the methodologies and models needed to confirm users understand the design, meet their needs, and ensure the project is on track. Software Development Life Cycle (SDLC) is one of the traditional methods of development software. It lists all the process and task necessary for developing software. But with the rapid development of machine learning that becoming vital in several industries, new methods for life cycle development needed to meet demand, such as Machine learning Life Cycle (MLLC). The main reason for needing (MLLC) due to many differences between machine learning applications and traditional information software. This paper deal with description on how SDLC differs from MLLC at different stages of the development life cycle. In this paper, we first provide brief discussion about SDLC, followed by the definition and importance of MLLC. Finally, we provide a comparison between SDLC and MLLC at different stages of the development life cycle.*

Keywords: *Software Development Life Cycle, SDLC, Machine Learning Life Cycle, MLLC, Phases*

I. INTRODUCTION

System development life cycle (SDLC) considered as one of the most robust methodologies that has been used in the past ten years. It handled and managed different problems regardless with their complexity, but recently it became difficult to use the SDLC methodology to handle "intelligent" applications and machines since it needs more simulations and intelligence.

Recently, Machine learning (ML) has been used and emerged in many fields since it can be easily used as ML components, and ML framework which enhances the productivity with preserving the quality at the same time since the employee or the user would enter the training set or the data that would be used as an inference to predict and solve the problems. Organizations started using the machine learning methods and techniques in applications that needs or deals with the artificial intelligence. At the beginning organizations were struggled because they were not having a methodology that was compatible with Machine Learning. Then the experts and researchers designed Machine Learning Life Cycle (MLLC) methodology to handle machine learning and artificial intelligence applications and frameworks [1].

SDLC focuses on the shallowed representation of the developed system were the MLLC focuses on designing a model that could automate and achieve tasks faster since it was trained by the set of data that has been entered by the system's expert which helps in designing the model and that is one of the first stages in MLLC. In fact, there are many differences between SDLC and MLLC that are helping the organizations and the experts to define which one is the suitable methodology due to the desired tasks.

In this review paper, the first part is talking about the definition of SDLC then the second apart is illustrating MLLC, the third part is discussing why MLLC is important then the last and main part of the paper which is the difference between SDLC and MLLC in details.

II. WHAT IS SDLC?

When an organization develops software, it needs a highly structured life cycle, which is important for placing process plans, principles, and specifications. Its main purpose is to define all activities related to the software development process, called the software development life cycle (SDLC). SDLC is an iterative process that includes the methods and standards required to build an information system. It brings value to the organization by addressing all necessary business requirements in the software development life cycle [2]. The first software development life cycle was presented by Herbert Benington in 1956. The first thoughts started around sequence of steps that used in development of software. Taking in consideration, the "software development" initially it formed only by coding. As programming come more sophisticated, development work requires more structured phases as the basis for project management [3]. As shown in figure.1 the software lifecycle models usually include some or all of this phases [4], which will be discussed later in this paper:

- Planning
- Requirement Analysis
- Designing
- Coding
- Documentation
- Testing
- Maintenance

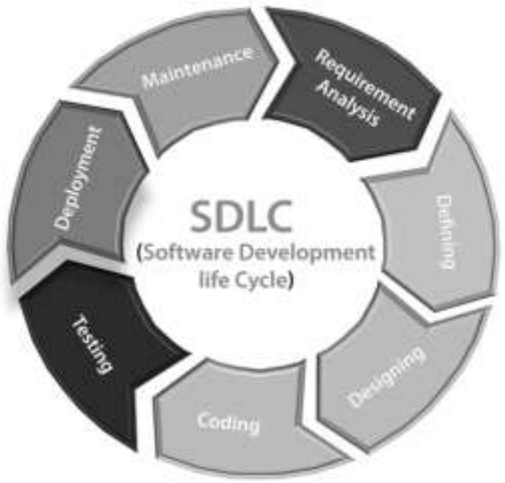

Figure.1 SDLC Phases [7]

According to needs, these phases and related activities can be achieved through different ways. Such these ways known as software Development Lifecycle Model (SDLC). These models provide how should developed software through two types. First, the descriptive model: Provides a describe the history of how was developed particular software system. On the other hand, the prescriptive model used as basis for understanding and improving processes of software development or constructing models that based on it.[4] In order to be successful in the software development process, each process model follows a specific life cycle. The following is an example of the SDLC model: [5]

- Waterfall model
- Spiral model
- RAD Model
- Prototyping Model
- Shaped Model

III. WHAT IS MLLC?

MLLC is an abbreviation of the Machine Learning Lifecycle. In [8], authors described Machine learning as the extraction of automated models from data. In addition, authors mentioned that the machine learning life cycle contains four phases: Data Management, Model Learning, Model Verification, and Model Deployment. They used the machine learning workflow term for the first three phases, which includes the activities in which the machine is taught, and models are produced. The fourth phase includes the activities to deploy the ML models in operational systems with the components obtained from the traditional systems engineering process.

According to [9], authors have mentioned another division of MLDLC (Machine Learning, Development Life Cycle) stages, as it was divided into five core stages:

1. Collect Data.
2. Data Refinement.
3. Model Training.
4. Execute Model.
5. Model Refinement.

The first and second stages are data collection, validation, and documentation. In the third and fourth stages, the appropriate model is determined by analyzing the model and its suitability for the project. At the end, the model is deployed after confirming and testing the model and evaluating the validity of the results. The MLDLC (Machine Learning, Development Life Cycle) is an iterative process as shown in the figure 2.

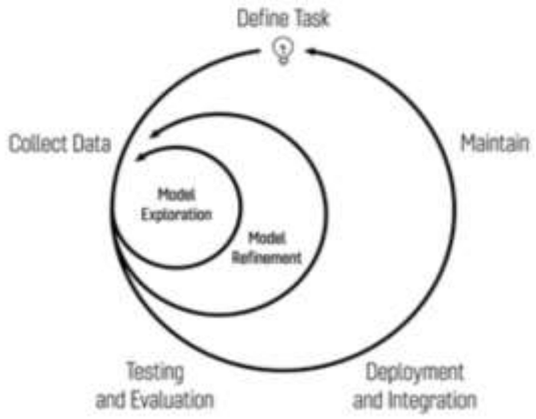

figure 2: MLLC phases [9]

There are several methodologies for MLLC, each methodology works best with specific types of data or projects. Some examples of MLLC methodologies:

- ***CRISP-DM Methodology***

CRISP-DM stands for “CRoss-Industry Standard Process for Data Mining” which commonly used in data mining and data analysis. It is an iterative process that contains six stages [12,13] There is no specific way to impose movement between the stages, the movement between them depends on what the work requires [14]. The six different phases are: (Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment) [12,13]. This methodology is most suitable for application in data mining projects when knowledge of the problem is insufficient [15].

- ***SEMMA Methodology***

This methodology developed by SAS Institute [14]. It is a non-iterative process, unlike the previous methodology, as it focuses on procedures rather than results [12]. This methodology is not considered a data mining method but rather a tool to aid in the mining process [14]. It contains four stages that make up the word SEMMA, which are (Sample, Explore, Modify, Model and Assess) [12,14]. This methodology deals with huge data to discover previously unknown patterns [12].

- ***Team Data Science Process (TDSP) Methodology***

This methodology is classified as an agile and iterative process which used in the field of data science and also used in the work of smart applications and predictive analytics. Its life cycle is similar to the CRISP-DM life cycle, as it includes five stages: (Business Understanding, Data Acquisition and Understanding, Modeling, Deployment, and Customer Acceptance) [12].

IV. WHY MLLC IS IMPORTANT?

The importance to develop lifecycle specific for machine learning programs comes from the importance and criticality of machine learning program itself. There are studies explaining the importance of applying the machine learning in several areas such as in paper [8], authors mentioned the importance of applying machine learning in safety-critical systems by making them able to understand the environment in which those systems operate and making these systems able to determine their response to the external changes that occur in their work environment. Also, it is considered an essential aspect as mentioned in [9] which says that Machine learning have an important role on the BPR (Business Process Re-engineering) which is an approach that aims to add improvements by raising the efficiency and effectiveness of the processes followed within the organizations. BPR detects and change processes to improve the Software Engineering Management (SEM).

According to [1], which focused the search about the differences between ML and non-ML development. Among its findings, the software development process consists of multiple practices and also includes people from different fields with varied experiences. So, it is not like a homogeneous mass.

In addition, [10] have talked about the applying of machine learning algorithms in software development. They mentioned that it has proven important in several areas, including when the field of the problem is not well-understood with poorly human’s knowledge, when the field has a huge database must be explored, and in the domains where the programs must be adapted to the conditions of their environment. According to [11] they discussed the issue of applying machine learning specifically in the areas of software development and maintenance by using machine learning algorithms. Through their research, they found that people's awareness of applying machine learning in this area has increased due to the benefits they have obtained, also the ability to work with machine learning techniques and software engineering tools together. Whereas machine learning techniques have the ability to design and develop software artifacts based on its mathematical and logical strength.

V. How Machine Learning Life Cycle is different from System Development Life Cycle?

Software development life cycle may have some similarity with the Machine learning Lifecycle, and it can be used as a right starting point for building machine learning program. However, there are several criteria that are related to model management in Machine learning life cycle which makes ML programs and traditional software vary on their basics. As a result, the development of ML-based software applications may face some challenges that results on major technical costs when planning to follow software development life cycle methodologies with its related traditional abstractions, software design principles and practices. Consequently, specific lifecycle, frameworks and tools have been researched on the recent decades for managing the deployment and development of machine learning program. [8,16]

The typical characteristics that make MLLC different than SDLC are the following:

- Data science is less about program development and more about analyzing and getting insights from the data [12].
- The ML Lifecycle is data-driven because the model, the output of training is dependent on the data it was trained on. However, traditional software systems building, or compilation are data independent.
- Unlike the software engineering lifecycle, even the most experienced and skilled engineers will not have a clear insight about the structure and results of the final program because of the ML program are experimental, combinatorial, and data driven.
- In ML program, context is extremely critical factor, so there might be a huge space of possibilities that differs based on the context. As a result, it will be necessary to understand which data artifacts were used to train which models, and with what configurations [16].

In the following points, we have identified the major challenges more specifically for the most important phases of software development life cycle in regard of Machine learning applications to help the reader get better insight about the reasons for emerging machine learning life cycle and the need for guidelines for best practices in collaborative research in the context of for machine learning applications.

1. *Software requirements:*

As mentioned, [36] the SDLC requirements engineering means analyzing, validating, specifying, and eliciting requirements that represent a software system’s intended objective.

Data is at the core of any application of machine learning (ML). The first stage of any ML cycle is data management. This stage is responsible for the acquisition of the data supporting the synthesis of machine-learned models. This data can then be used “to predict future data, or to perform other kinds of decision making under uncertainty” as stated in paper [17]. For some scenarios, it is possible that data samples are either unavailable, too costly, time consuming, or even dangerous. In this case, the augmentation methods put forth in papers [18, 19] are used to procure more data samples and add them to the collected data sets. Additionally, the data collected from multiple sources may be heterogeneous in nature, and therefore preprocessing, as suggested in papers [20,21], may be required to produce consistent data sets. This is often required for training and verification purposes.

ML system requirements are heavily data driven. As paper [17] has noted, the ML system requirements are far more uncertain than those of the non-ML systems. Research from paper [1] outlines that ML systems aid in streamlining decision making processes within organizations. Rather than functional descriptions, it provides you with a more conceptual description of an organization's goals, after taking into account the data that has been gathered from the ML systems. Because the ML systems are so heavily data-driven, this means that the combination of different data results can result in a wide variety of different results.

While the non-ML software systems usually require functional requirements, ML system requirements are comprised mainly of quantitative measures. Information from paper [11] suggests that distinct types of quantitative measures would be leveraged to define requirements, e.g., accuracy, precision, recall, F measure and normalized discounted cumulative gain (nDCG). Paper [1] proposes that the target scores for quantitative measures could vary from one application to the next. In places where safety is of the highest value, the accuracy of the ML systems is of the greatest importance. In an environment such as this, higher scores of quantitative measures are both desired and expected for safety reasons.

Requirements for ML systems typically involve a large number of preliminary experiments, which varies from the requirements of non-ML systems. As paper [1] has noted, business stakeholders might suggest leveraging a number of emerging machine learning algorithms to solve their business problems. The largest consequence of this is that it is then necessary that the requirement specialists already have a strong background in machine learning. The other consequence is that the requirement validation process involves a larger number of preliminary experiments, as outlined in paper [1]. Those preliminary experiments are conducted by software engineers that set out to validate and select machine learning algorithms from different sources.

As paper [1] has suggested, it is imperative that take into consideration the possible degradation in the lifespan of the ML systems over time. The findings from papers [1, 17, 11] emphasize that there are two very important requirements of a ML system: firstly, that they should be capable of becoming aware of any degradation of their own performance. Secondly, if degradation in performance occurs, the ML system needs to be able to perceive that degradation and be able to adapt to it. The way that the ML system would be able to adapt to it is either by sending new data to the learning algorithm, or it must use brand new data to train a new model.

2. *Software design:*

The non-ML software systems architecture always described by software design where it represents "how software is decomposed and organized into components" [22]. While in ML systems the architecture and design of model done in model learning stage, it typically starts by selecting the type of model according to type of problem, structure and volume of training data and good personal experience [17].

As noted in [1], the architecture of ML systems differs than the architecture of non-ML software systems. ML architectures consist of data collection, data cleaning, feature engineering, data modeling, model execution and model evaluation [23]. Generally, this architecture relatively fixed in a lot of ML systems [1]. While non-ML software systems implements different software components structures and also generate the descriptions of these components behavior like: drawing data flow diagram and activity diagram for these components [1,17]. The main issue for data design is the generation of huge amount of data [22,36]. The distributed architectural style is the better solution of these issue [1,27] which may lead to complexity in architectural design of ML systems [22].

ML systems consist of different components each component with its own function but all of these components are dependent on each other [1,17]. For example: the performance of model highly dependent on pre-processing of data, if it is done weakly that lead to failure of produced model. So, pre-processing of data considered as the most important stage in ML systems that effect on the performance of model [22]. If there is more than one model, often each model work separately without any dependent on another model [24]. In contrast, the high dependency occurs between components of non-ML software systems [10]. From that we can conclude the components of non-ML software systems are highly coupled than the components of ML systems [1].

As mentioned in [9] the detailed design of ML systems considered more flexible than non-ML software systems. For each model of ML systems, we can use different numbers of algorithms [22] as such each algorithm can be used in different models [1,17]. As a result, the construction of a good model with detailed design can be applied in multiple iterations until reach to a proper model [9] which are taken time and considered as time consuming [22].

3. *Software Construction:*

Usually, software construction mostly refers to coding and debugging [22]. Coding in non-ML software systems done in coding phase where it responsible about translating the design of the software into code by using particular programming language [8]. While coding in ML systems involves data processing, feature analysis and data modeling by using data and algorithms [1,36]. Non-ML software systems highly depends on coding and it needs high workload to perform the codes [1,8,26]. In contrast, ML systems highly depends on data and its workload of coding is low as compared to non-ML software systems [9,17]. In non-ML software systems code can be reused significantly between different software systems [1] to increase productivity and reduce time and cost of development [25]. The code in ML systems considered as low little reused between ML systems because ML systems significantly emphasis on performance. Data considered as the basic effecter on the performance of model [1]. Libraries in ML systems can be reused also commonly between different models [36]. The data also in ML systems can be reused from external domain [36].

Debugging generally related to find bugs and errors [8]. In non-ML software systems debugging responsible about fix bugs and errors in the code [26]. While in ML systems debugging responsible about performance improvement. The performance of ML systems cannot be evaluated until the data model is finished and finalized [25]. From that we can know the importance of finalizing data model in the ML systems construction [1].

During to the multiple iterative training for data modeling [17,22,36], training a high volume of data require a long time to complete training [1]. So, [25,36] suggested that the data should be divided into multiple datasets with different size. The training start with small dataset then take a large dataset after reaching a good result from training of small datasets to save time of training, improve training efficiency and mitigate risk may occurred from incomplete training.

The practice of debugging in non-ML software systems done in step-by-step execution of program by using different methodologies that depend on determining breakpoints [25]. The aim of debugging of ML systems is "to translate ideas from developer's head into code" in more straightforward way [1].

The bugs are considered as a result of debugging process. Bugs differs between ML systems and non-ML software systems. Many bugs can be found in code of non-ML software systems [26] while the bugs in ML systems is a data bug [36]. Also, bugs in ML systems can be generated as a result from the integration of two frameworks with different order of dimensions of data [1]. Bugs in ML systems are hardly diagnosing, the developers must be going in each line of code and attempt to find the causes of bugs [25]. While in non-ML software systems diagnosis of bugs is straightforward and easier than ML systems [20,26].

4. *Software Testing:*

In SDLC, software testing means checking if a program for specified inputs gives correctly and expected results. Software testing is a critical component of software quality assurance, especially for life-critical software such as flight control testing which can be highly expensive. Software testing is a process of executing a program with the goal of finding errors. So, testing means that one inspects behavior of a program on a set of test cases i.e., a set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement, for which valued inputs always exist [27]. However, the testing of ML systems poses new challenges for software testing community. Although software quality is important in both ML and non-ML systems, the practice of testing appears to differ significantly [1]. As a result, new guidelines are needed to help ML developers cope with the challenges they face to validate the ML programs [28].

Some of the main challenges is that traditional software systems are constructed deductively by writing down the rules that govern the behavior of the system as program code. However, with ML, these rules are inferred from training data which means they are generated inductively resulting in systems that are fundamentally challenging to test because they do not have complete specifications or even source code corresponding to some of their critical behaviors. In fact, ML models learn from the input data in an adaptive and iterative manner. The model verification stage of the ML lifecycle is concerned with the provision of auditable evidence that a model will continue to satisfy its requirements when exposed to inputs which are not present in the training data. Consequently, the inductive nature of ML programs makes it difficult to reason about their behavior [29] and the generated rules might even be unknown to the developers. Thus, it is harder to identify the erroneous system behaviors and to detect the source of the bugs [28].

Moreover, ML systems lack a reliable explicit oracle to detect what is the correct output should be for arbitrary input. As a result, it will be difficult to detect subtle errors, faults, defects, or anomalies in the ML applications [30]. In addition, machine learning programs do not have control flow like traditional programs and cannot be tested with traditional software testing techniques which rely on explicit oracle and program control flow.

Another difference is that in traditional software testing it is possible only to show the presence of bugs but not their absence. Usually when input or output equivalence classes are applied to developing test cases so, the expected output for a given input is known in advance. In ML, the problem is how to devise test cases that are likely to show bugs, and how one can know whether a test is revealing a bug, given that we do not know what the output should be in the general case [31]. The functionality of ML systems depends on the set of data input to them and a minor change in the training data can have major influence on the behavior of the system and the results of the learning process [28].

According to [1] in ML programs, the testing outputs are expected to be a range rather than a single value[1]. Compared with traditional software, the dimension and potential testing space of a ML programs is much larger. Current existing software development techniques must be revisited and adapted to this new reality [29].

Furthermore, ML algorithms might sometimes exhibit robustness to some bugs and produce reasonable outcomes by compensating for noisy data or implementation errors. Thus, bugs in ML application can be tricky to detect and fix. Furthermore, testing of ML applications may involve large scale training data and manual labeling of such data which is a highly cost task. Moreover, a random selection of subsets of data is likely to fail in identifying many cases [28].

One significant difference also exists in the reproducibility of test results. In contrast to non-ML software systems, the testing results of ML systems is hard to reproduce because of several sources of randomness. As the outcomes of machine learning models are stochastic in nature there might not be unique results to compare and verify machine learning applications [28]. Test case generation for ML systems is more challenging, compared to non-ML systems. In addition, automated testing tools are not used as frequently as non-ML systems [1] such as the existing unit testing frameworks PyUnit for Python cannot be readily used to test ML applications [28].

All these issues make the testing and fixing of erroneous behavior in ML applications a very challenging task and different than the software systems. According to [30], such systems have been classified as "non-testable" by traditional methods because it is a hard and complex process to explain the model behavior. As a result, some researchers leading to the requirement of developing novel strategies to test machine learning applications. Several techniques are recently proposed in the literature for testing ML models such as the evaluation of the model accuracies with the evaluation data set that must be complete enough to represent all possible use-case scenarios [28]. In addition, these techniques can be used to analyze and test the scalability, generality, and robustness of ML program [30].

5. *Software Maintenance and configuration management*

As mentioned in [32], software maintenance is about the activities required to provide the effective support to software. Maintenance activities are performed during the pre-delivery stage and during the postdelivery stage. Pre-delivery activities is about the planning for post-delivery operations, maintainability, and determination for transition activities. Postdelivery activities include software modification, training, and operating or interfacing to a help desk.

The ML program model management phase focus on the maintenance, deployment and monitoring of ML models which is a challenging task in building ML program. ML models as mentioned before are data-driven and are based on different assumptions on the distributions and patterns of data. Therefore, initial characteristics of data may not hold due to frequent changes in the data which might affect the model behavior [28]. ML systems are highly predicted to degrade in performance across time and in provide constantly robust results. Thus, most ML systems usually support automatic maintenance. Once performance degradation occurs, ML systems is designed to recognize the degradation and trains new data models in an online or offline way using the latest emerged data. "Health factors" or quantitative indicators are often used to check the status of a machine learning system as part of ML monitoring process. The indicators help machine learning system monitor its performance in the specific application context [1]. In fact, robust, continuous monitoring models will help in detecting changes and carefully adapt to changing conditions.

In machine learning programs, one of the major threads is that uncertain events might occur in the deployment phase because the environment of machine learning production might largely differ from the environment the ML models were trained and evaluated. In many cases, the ML models may be frequently retrained with concept drift which means the data characteristics may depend on some hidden context, not given explicitly in the form of predictive feature at the time the model have been built, ex: the patterns of customers' buying preferences may change with time. As a result, the behavior of the model will change independently without considering the real-world environment changes [32,33]. Therefore, ML post-deployment monitoring of models is important and ML systems needs more monitoring efforts more than traditional software. Monitoring in machine learning applications includes:

- **Data monitorin**g: monitoring the data characteristics provided to the model because it could negatively impact the quality of a model's predictions. Monitoring tools for data dependencies, automatic data validation and cleaning during runtime, and concept drift adaptation must be implemented.
- **Operation environment monitoring:** machine learning production environment might differ from the environment the ML models were trained and evaluated which leads to add new functional modules to the application. Over time, operation environment might add some challenges such as compatibility, portability and scalability that needs to be considered.
- **Model monitoring:** monitoring the internals of the model and its performance including decisions on model retraining, adversarial settings, and backwards compatibility of trained models.
- **Runtime Monitoring:** Selection of the metrics used for monitoring, live monitoring of system behavior that allows automated responses without direct human intervention, and dynamic monitoring for runtime verification and certification [8, 30].

To apply changes to the machine learning models with the objective of maintaining the application, some procedures are needed to be implemented to control the change process. Software configuration management is a supporting-software life cycle process that benefits project management, development and maintenance activities, quality assurance activities, as well as the customers and users of the product [32]. However, configuration management for ML systems involves a larger amount of content compared to non-ML software. One reason is that machine learning models include not only code but also data, model files, model dependencies, hyperparameters, and parameters that must be considered on the configuration management [1,28]. Machine learning is all about data. The amount of effort it takes to discover source, manage, and version data is inherently more complex and different than doing the same with software code [32]. In addition, it is important to keep track of the model versions, dataset, and configurations to allow reproducibility of ML models and easier management of ML workflow. Model reproducibility helps us to analyze and compare model behavior and performance and supports deployment or roll out decisions. We defined policies to version data, keep track of models and configurations for comparative analysis of our ML models [28].

In fact, all previously mentioned challenges on ML model management and configuration management for ML applications needs to be considered which leads to new tools and techniques to be designed[32].

6. *Software Engineering Processes and Management*

The developmental lifecycles for an application that is not ML would not be adequate when applied to ML systems as the need for data-oriented or model-oriented is not considered, this is especially the case when managing lifecycles [36].

They are both very different from each other in terms of the process involved when developing them. A paper [9] suggests considering the importance of communication with every stakeholder, regarding the capabilities of machine learning. Another paper [20] mentions the importance of data processing if one wishes to successfully carry out the entire process. The data visualized in paper [9] helps understand how the features are distributed and how they correlate with each other. They also help to identify any trend or pattern that may exist within the data. These steps aim to determine how various aspects can be balanced using a trial and error method. Even the most sophisticated machine learning programs use their mistakes as something to learn from.

The addition of any ML component into software requires a few actions that must be taken first. The development of an ML system must first be guided using software that is altered depending on what the development team requires [35]. Data lifecycles involve a variety of steps that are critically and compellingly designed so that every project would be successful [20]. Thus, Agility becomes something that is extremely essential when deciding on an approach as a result of the evolution and the increased demand within the ML system industry that fosters newer ideas and innovations. ML models and their lifecycles help provide a second example of the efficient integration of processes that help with the success of the project [9]. Apart from the two lifecycles mentioned, each team would also require assistance on several aspects, including: Engineering processes, data-handling, tests, as well as generally maintaining the same work process.

The uncertainty attributed to the various ML process makes it difficult to provide an accurate estimation of the labor required and make a plan of the same [34]. There can be instances where teams aren’t allowed to find out the achievability of their target quantities till there is a final ML model [36]. Since estimation is not possible, an impatient would be likely to begin the cancellation of the developments for ML components although it would have an intermediate result that shows a great deal of progress.

The papers [9] and [20] not that real-world applications for ML projects within companies would face a great dealing of difficulty when developing, deploying, and operating in a way that complies with the privacy-policies for those organizations as well as their legal framework. Such a demand would be challenging from a technical as well as a management perspective.

VI. CONCLUSION

In this paper we focused on defining the SDLC and MLLC terminologies then illustrating the differences between ML development and non-ML development in details by defining the differences according to ML development steps. Apparently, machine learning is the new methodology that would adapt with the new technologies. This decade is depending on big data and data science, which makes MLLC useful since it is data driven and deals with uncertainty which helps in avoiding risks, damages and increase the business profits as well. Machine learning algorithms and applications are already existing in our phones. In the future, understanding and having knowledge about the machine learning would be one of our life's basics.

REFERENCES

[1] Z. Wan, X. Xia, D. Lo and G. C. Murphy, "How does Machine Learning Change Software Development Practices?," in IEEE Transactions on Software Engineering, doi: 10.1109/TSE.2019.2937083.

[2] Murch, R. (n.d.). The software development lifecycle - a complete guide. Retrieved March 20, 2021, from https://books.google.com.sa/books?id=Q7utBQAAQBAJ&lpg=PA11&hl=ar&pg=PA16#v=onepage&q&f=false

[3] Kneuper, R. (2017). Sixty years of software development life Cycle Models. IEEE Annals of the History of Computing,39(3), 41-54. doi:10.1109/mahc.2017.3481346

[4] Rastogi, V. (2015). Software Development Life Cycle Models- Comparison, Consequences. International Journal of Computer Science and Information Technologies.

[5] Ragunath, P., Velmourougan, S., Davachelvan, P., Kayalvizhi, S., & Ravimohan, R. (2010). Evolving A New Model (SDLC Model-2010) For Software Development Life

Cycle (SDLC). IJCSNS International Journal of Computer Science and Network Security.

[6] Sujit, K., & Pushkar, D. (2013). SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) ANALYTICAL COMPARISON AND SURVEY ON TRADITIONAL AND AGILE METHODOLOGY. National Monthly Refereed Journal of Research in Science and Technology.

[7] SDLC - software development life cycle - JAVATPOINT. (n.d.). Retrieved March 22, 2021, from https://www.javatpoint.com/software-engineering-software-development-life-cycle

[8] Paterson, C., Calinescu, R., & Ashmore, R. (2021). Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges. *ACM Computing Surveys*.

[9] Bhavsar, K., Shah, V., & Gopalan, S. (2020). Machine Learning: A Software Process Reengineering in Software Development Organization. *International Journal of Engineering and Advanced Technology (IJEAT)*, *9*(2), 4492-4500.

[10] Zhang, D. (2000, June). Applying machine learning algorithms in software development. In *Proceedings of the 2000 Monterey workshop on modeling software system structures in a fastly moving scenario* (pp. 275-291).

[11] Zhang, D., & Tsai, J. J. (2003). Machine learning and software engineering. *Software Quality Journal*, *11*(2), 87-119.

[12] Karamitsos, I., Albarhami, S., & Apostolopoulos, C. (2020). Applying DevOps practices of continuous automation for machine learning. *Information*, *11*(7), 363.

[13] Wirth, R., & Hipp, J. (2000, April). CRISP-DM: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining* (Vol. 1). London, UK: Springer-Verlag.

[14] Dåderman, A., & Rosander, S. (2018). Evaluating frameworks for implementing machine learning in signal processing: A comparative study of CRISP-DM, semma and kdd.

[15] Palacios, H. J. G., Pantoja, G. A. H., Navarro, A. A. M., Puetaman, I. M. A., & Toledo, R. A. J. (2016, September). Comparativa entre crisp-dmy semma para la limpieza de datos en productos modis en un estudio de cambio de cobertura y uso del suelo: Comparative between crisp-dm and semma for data cleaning of modis products in a study of land use and land cover change. In *2016 IEEE 11th Colombian Computing Conference (CCC)* (pp. 1-9). IEEE.

[16] Garcia, R., Sreekanti, V., Yadwadkar, N., Crankshaw, D., Gonzalez, J. E., & Hellerstein, J. M. (2018). Context: The missing piece in the machine learning lifecycle. In *KDD CMI Workshop*.

[17] Ashmore, Rob, Radu Calinescu, and Colin Paterson. "Assuring the machine learning lifecycle: Desiderata, methods, and challenges." arXiv preprint arXiv:1905.04223 (2019).

[18] German Ros, Laura Sellart, Joanna Materzynska, et al. 2016. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In IEEE Conf. on computer vision and pattern recognition. 3234–3243.

[19] Sebastien CWong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. 2016. Understanding data augmentation for classification: when to warp?. In Int. Conf. on digital image computing: techniques and applications. IEEE, 1–6.

[20] SB Kotsiantis, Dimitris Kanellopoulos, and PE Pintelas. 2006. Data preprocessing for supervised leaning. Int. Journal of Computer Science 1, 2 (2006), 111–117.

[21]Shichao Zhang, Chengqi Zhang, and Qiang Yang. 2003. Data preparation for data mining. Applied artificial intelligence17, 5-6 (2003), 375–381.

[22] Kumeno, F. (2019). Sofware engneering challenges for machine learning applications: A literature review. *Intelligent Decision Technologies*, *13*(4), 463-476.

[23] Washizaki, H., Uchida, H., Khomh, F., & Guéhéneuc, Y. G. (2019, December). Studying software engineering patterns for designing machine learning systems. In *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)* (pp. 49-495). IEEE.

[24] Werkt, P. (2019, September 05). Software engineering for machine learning applications. Retrieved April 14, 2021, from https://fontysblogt.nl/software-engineering-for-machine-learning-applications/

[25] Fisher, D., DeLine, R., Czerwinski, M., & Drucker, S. (2012). Interactions with big data analytics. *interactions*, *19*(3), 50-59.

[26] Istyaq, S., & Zargar, A. (2010). Debugging, Advanced Debugging and Runtime Analysis. *IJCSE) International Journal on Computer Science and Engineering*, *2*(02), 246-249.

[27] Bourque, P., Dupuis, R., Abran, A., Moore, J. W., & Tripp, L. (2004). Guide to the software engineering body of knowledge.

[28] Rahman, M. S., Rivera, E., Khomh, F., Guéhéneuc, Y. G., & Lehnert, B. (2019). Machine learning software engineering in practice: An industrial case study. *arXiv preprint arXiv:1906.07154*.

[29] Braiek, H. B., & Khomh, F. (2020). On testing machine learning programs. *Journal of Systems and Software*, *164*, 110542.

[30] Sherin, S., & Iqbal, M. Z. (2019). A systematic mapping study on testing of machine learning programs. *arXiv preprint arXiv:1907.09427*.

[31] Murphy, C., Kaiser, G. E., & Arias, M. (2007). An approach to software testing of machine learning applications.

[32] Kumeno, F. (2019). Sofware engneering challenges for machine learning applications: A literature review. *Intelligent Decision Technologies*, *13*(4), 463-476.

[33] Tsymbal, A. (2004). The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, *106*(2), 58.

[34] Arpteg, A., Brinne, B., Crnkovic-Friis, L., & Bosch, J. (2018, August). Software engineering challenges of deep learning. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 50-59). IEEE.

[35] Ishikawa, F., & Yoshioka, N. (2019, May). How do engineers perceive difficulties in engineering of machine-learning systems?-questionnaire survey. In *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)* (pp. 2-9). IEEE.

[36] Giray, G. (2020). A Software Engineering Perspective on Engineering Machine Learning Systems: State of the Art and Challenges. *arXiv preprint arXiv:2012.07919*.