Towards a Grounded Theory for a Development Process Model for Machine Learning Based Systems

André Meireles*
andre@crateus.ufc.br
Universidade Federal do Ceará
Brazil

Rainara M. Carvalho
rainara@ufc.br
Universidade Federal do Ceará
Brazil

Thiago Rique
thiago.rique@virtus.ufcg.edu.br
Universidade Federal de Campina Grande
Brazil

Maryzangela B. P. Cavalcante
maryzangelabessa@alu.ufc.br
Universidade Federal do Ceará
Brazil

Mirko Perkusich
mirko@virtus.ufcg.edu.br
Universidade Federal de Campina Grande
Brazil

Hyggo Almeida
hyggo@virtus.ufcg.edu.br
Universidade Federal de Campina Grande
Brazil

Angelo Perkusich
perkusich@virtus.ufcg.edu.br
Universidade Federal de Campina Grande
Brazil

ABSTRACT

The software industry has experienced the integration of artificial intelligence capabilities into applications, facing new challenges regarding software development. Despite research and industry contributions providing lessons learned and best practices, no study proposed a reference process for developing this type of software, and practitioners still struggle to establish a working process. Through a Grounded Theory study involving practitioners with experience in machine learning (ML) projects, this paper presents an emerging theory of how ML-based systems are developed. The reported results comprise key elements of a reference development process with its respective phases and activities.

CCS CONCEPTS

• **Software and its engineering → Software development process management**; • **Computing methodologies →** *Machine learning*.

KEYWORDS

Machine Learning Systems, Software Development Process, Software Engineering, Intelligent Software Engineering

ACM Reference Format:
André Meireles, Rainara M. Carvalho, Thiago Rique, Maryzangela B. P. Cavalcante, Mirko Perkusich, Hyggo Almeida, and Angelo Perkusich. 2021. Towards a Grounded Theory for a Development Process Model for Machine Learning Based Systems. In *Proceedings of Brazilian Workshop on Intelligent Software Engineering, September 27 and 28, 2021, Online Event (ISE' 21).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.5753/ise.2021.17278

1 INTRODUCTION

Software teams seek to stay up to date with development approaches and application domains, as the software industry experiences different trends. Advances in machine learning (ML) has leveraged the integration of artificial intelligence (AI) into software systems, which brings with it a set of new challenges and issues.

Different aspects which are not commonly found when developing traditional software should be considered in the development of ML-based systems. In addition, integrating AI capabilities impacts the processes companies use to deliver their products and services. The software engineering community has made efforts to deal with the particularities brought by this trend, identifying and discussing issues and challenges, and providing lessons learned and best practices [1, 3, 6].

Despite the community's contributions, developers still face difficulty in establishing a repeatable process [3]. This is where a research gap arises regarding the need for a reference development process for ML-based applications. So, the authors of this paper performed a Grounded Theory study guided by the following research question: "How do practitioners develop ML-based systems and how to integrate the activities, roles, and underlying aspects into a development process?" Based on interview transcripts of 6 participants, this study presents an emerging theory of a development process for machine learning projects. The authors report the phases, activities, and related aspects identified during the analysis, providing additional information on future steps to the development of the whole theory.

ISE' 21,
2021. https://doi.org/10.5753/ise.2021.17278

2 BACKGROUND

2.1 The need for evolving SE processes

As the dynamics of application domains change in the software industry, teams in different companies have adapted and evolved the processes used to develop software. That was how it happened when plan-driven approaches for software development proved to be inappropriate to respond to customers' changing requirements. Agile methods took place in the software engineering landscape, bringing a change-driven set of practices that help deliver valuable software faster to the customer.

With the dissemination of AI, different types of applications have been developed (e.g., integrating ML components). In this context, practitioners face new challenges and have to cope with the particularities of this type of software. To integrate ML features into the phases and activities of a development process, workflows have been proposed, having as a key characteristic their data-centered nature and feedback from one stage to another [1].

2.2 Software Engineering for Artificial Intelligence

Software engineering has been supported by tools with diverse purposes such as software versioning and project management. These tools generate a large amount of data that can be used to get insights on the different issues related to software development. As a result, an emergent area has gained interest in recent years: intelligent software engineering (ISE). In the context of their study, Perkusich et al. [5] define ISE as the application of intelligent techniques to tackle software engineering problems. By intelligent technique, the authors mean not only AI-based solutions but any technique that explores data with the purpose of knowledge discovery, reasoning, supporting decision-making, among others.

Besides considering the perspective addressed by Perkusich et al., Xie defines ISE under a different aspect of denotation (which is used throughout this paper): the development of software engineering solutions for intelligent software. With the increasing dissemination and popularity of AI, many efforts have been made to improve the productivity of intelligent software development, as well as the dependability of intelligent software [11].

Software processes have been integrated with AI workflows, providing insights about challenges and issues related to the development of AI infrastructure and applications. That is what the work of Amershi et al. [1] is about. The authors provide lessons learned from the experiences of software teams building AI-based applications and point out three fundamental aspects regarding the development of intelligent software: i) data discovery, management and versioning is inherently more complex than the same activities in other application domains; ii) skills different from the ones in typical software teams are required to develop models. That means that only software engineering skills are not enough: a consistent knowledge of machine learning foundations is necessary; iii) handling AI components as distinct modules does not work the same way as with other software components. Models can affect one another due to their inherent complexity [1].

So, several aspects should be considered to adapt and evolve software processes to meet the needs of developing AI-based applications. Many times these aspects are not well understood, not even clearly stated, and practitioners still struggle to operationalize processes to support machine learning workflows [3], which reinforces the need for a reference development process for intelligent software.

3 METHOD

Grounded Theory is a research method that supports theory building from empirical data in a systematic way that includes analyzing data through coding and categorizing, supported by a process of constant comparison [10]. The choice of Grounded Theory is justified by the interest of the researchers towards generating a theory for a reference development process for AI applications. Rather than validating any theory from the literature on the topic, the authors of this paper aimed at uncovering aspects for the development of intelligent software from empirical data, focusing on understanding the activities, the roles, how they are related and underlying factors and conditions.

The Grounded Theory method is composed of the following phases: planning, data collection, coding, and reporting results. The planning, collecting and coding are described in the next subsections. Results are reported in Section 4.

3.1 Planning

The planning phase aims to identify the area of interest and the research question that will drive the work. In this work, the area of interest is Software Engineering for Machine Learning, and the research question is: "How do practitioners develop ML-based systems and how to integrate the activities, roles, and underlying aspects into a development process?".

As mentioned in previous sections, practitioners still struggle to establish working processes in which ML workflows take place. So the authors planned to build a theory from empirical data regarding the experience of practitioners involved in the development of ML-based systems. To obtain a picture of the participants' perspective on the topic, interviews were designed and conducted, containing in-depth questions about several aspects related to intelligent software development, and the researchers had the opportunity to follow-up on the topic with the interviewees [10].

This work adopts the Straussian grounded theory since its position is aligned with the aim of the authors regarding aspects such as the research question and the role of the literature [8], and it also provides clear guidelines to researchers.

3.2 Data Collecting

To collect data about how engineers develop systems with machine learning, we interviewed 6 developers. We selected them by the convenience sampling technique [4]. The criteria to select them were:

- The interviewee should be a member of a machine learning development team
- The interviewee should have more than three years of experience with machine learning projects

Moreover, the authors selected the samples in order to guarantee different contexts of ML projects execution (academy and industry, different ML areas, different business domains) to reduce data bias. Table 1 presents the samples attributes details.

As soon as the first data became available, data analysis started, feeding a simultaneous process of collecting and analyzing data. The interviews were designed and executed as intensive interview that permits an in-depth exploration of particular topics [2] that the interviewer consider more relevant. The interview questions are accessible online[1].

#	Context	Qualitif.	Exp. Time	ML Area	Business Domain
1	Academy	Master	7 years	Time series, Prediction	Market Product Demand Prediction
2	Academy	PhD	6 Years	Time series, Prediction	Hardware components fails
3	Industry	PhD	4 years	Computing Vision	Document Classification
4	Industry	Technic	4 years	Recomendation Systems	E-commerce retail
5	Industry	PhD	7 years	Computing Vision	Agro Industry
6	Industry	PhD	4 years	Classification and Prediction	Hardware components analysis

Table 1: Data Collecting - Samples details

3.3 Coding

Coding means extracting concepts from raw data and relating them to each other until reaching a core concept [9]. In the case of this study, the idea is to extract and relate concepts that characterize how engineers develop machine learning applications.

The coding process is performed in three tasks: open, axial and selective coding. Each one of them is better explained by showing what it means and its outcomes.

[1]Interview Script: https://drive.google.com/file/d/1VuDcdq7otVuSmHQgS42izroicXgavM53

3.4 Open Coding

In this task, the researcher inspects the data to understand the essence of what is being expressed. In the case of this research, the authors inspected the data extracted from the interviews. Then, the researcher (first author) created a conceptual name (code) to represent his understanding. Codes can represent a single word, a phrase or a whole paragraph. The MAXQDA12 tool was used to support open coding.

Figure 1 (A) illustrates the application of the open coding procedure using some examples. As presented, 'eda – exploratory data analysis', 'data visualization', 'intelligence modelling' and 'intelligence integration' emerged as codes from the interview transcripts. By applying a constant comparison process (a key component of the Grounded Theory method [8]), these codes and codes from other interview transcripts were integrated into a higher level of abstraction, categories. In a particular case, the codes 'eda – exploratory data analysis' and 'data visualization' were grouped into the category 'feasibility analysis'.

3.5 Axial Coding

Axial coding is the process of relating concepts, or grouping them by creating categories (a high-level concept that represents a group of codes) [9]. These relations between concepts can be defined by the researcher, although there are already existing relations that can be reused, such as "is a", where a concept is a kind of another concept.

To illustrate an example of axial coding, Figure 1 (B) shows how the subcategories 'intelligence development' and 'service development' relate to each other, since they represent similar concepts (with particular characteristics) into the phenomenon under study. Given these subcategories, a category called 'development' was created to group them, establishing a relationship that emerged from the analysis.

3.6 Selective Coding

Finally, when all codes and categories can be related to a core category, it means the researcher is doing the selective coding. As Figure 1 (C) indicates, the authors related the subcategories and categories derived from the open and axial coding to identify the core category. In selective coding, the emerging theory started to take shape as the subcategories and categories captured a part of the whole process. The core category represents the central phenomenon and, in the case of this study, builds the theory of a development process for intelligent software.

2

4 RESULTS

This study's execution involved, until here, 8 hours and 33 minutes of interviews from which 312 codes and categories were generated. These codes were mapped by one researcher and reviewed by two other researchers.

The emerging theory until then is "Development Process for Machine Learning Projects". The following sections present details about the theory formulation based on the data collection and analysis.

4.1 Emerging Theory: Development Process for Machine Learning Projects

The authors observed that ML projects life cycle works quite similarly to traditional softwares, where almost all the activities (requirement elicitation, analysis design, implementation, tests, deployment and monitoring) are performed on each project's phase, but the effort applied for each activity depends on the project's phase. This same scheme, with small differences, can be found in the Unified Process Model and Rapid Application Development [7].

In this emerging theory, a ML project is composed by three phases: *Feasibility Analysis*, *Development* and *Operation*. Although the proposed phases look similar or can be mapped to the phases defined by other classic software development models, such as *Inception*, *Elaboration*, *Construction*, *Transition* and *Production*, from *Unified Process Model* [7], the key findings of this work are related to new identified activities, the effort applied to each activity for each phase, and the criticality and duration of each phase. The reasons why the authors concluded that are detailed in the next sections.

It is important to highlight that this emerging theory does not only intend to represent how ML projects are executed but also to reduce the issues and risks identified by the interviewees' declarations.

4.2 Phase 1 - Feasibility Analysis

ML projects are often started from a problem that someone believes can be solved by using machine learning or other data science techniques. The problem is that, different from traditional software, the client does not comprehend a ML solution enough. The interviews showed that this problem happens even in big companies with a large experience with ML solutions. See in Table 2, reports from 3 different interviewees that were coded as "client's high or unreachable expectation".

#4 "Often, in the case of the company, there is a definition from the marketing team, they want something and that this problem could be solved by using machine learning or data science in general. But there is still not much maturity and the definitions are quite open, causing some difficulties"
#1 "Because, when working with machine learning, the client is very anxious, he wants to see the result right away, even if it's right at the beginning, for example, if it's a prediction problem, he already wants it to be working 95% well"
#2 "They [clients] have an expectation that ML will solve all the problems, that it's like a silver bullet that will deliver everything they expect"

Table 2: Text Segments related to Client's Expectations

Due to that, some interviewees reported that their ML teams, based on the lessons learned, defined a feasibility analysis phase in order to solve or reduce problems related to the following risk points that commonly occur in ML projects:

- Make the client problem clear;
- Authorization to access the needed data;
- Verify the available data quality;
- Identify existing techniques to solve the problem;
- Identify needed improvements to have suitable data to work;
- Client expectation alignment.

Such points, in this theory, are verified by the execution of a set of activities that were coded and related to the Feasibility Analysis category, as presented in Figure 2. The codes and their respective text segments that grounded the definition of these activities and the existence and importance of the feasibility analysis phase can be accessed on the online sheet of codes[2]. The same approach[3] was applied for all the next concepts presented hereafter. The online sheets show only part of the samples of text segments coded during this work.

[2]https://drive.google.com/file/d/1upVutG6AvOadcClv12L5qhsWziH80D8Q
[3]https://drive.google.com/file/d/1zqQZ9VAsHddMjEXHd1yd3OWLFqztX853

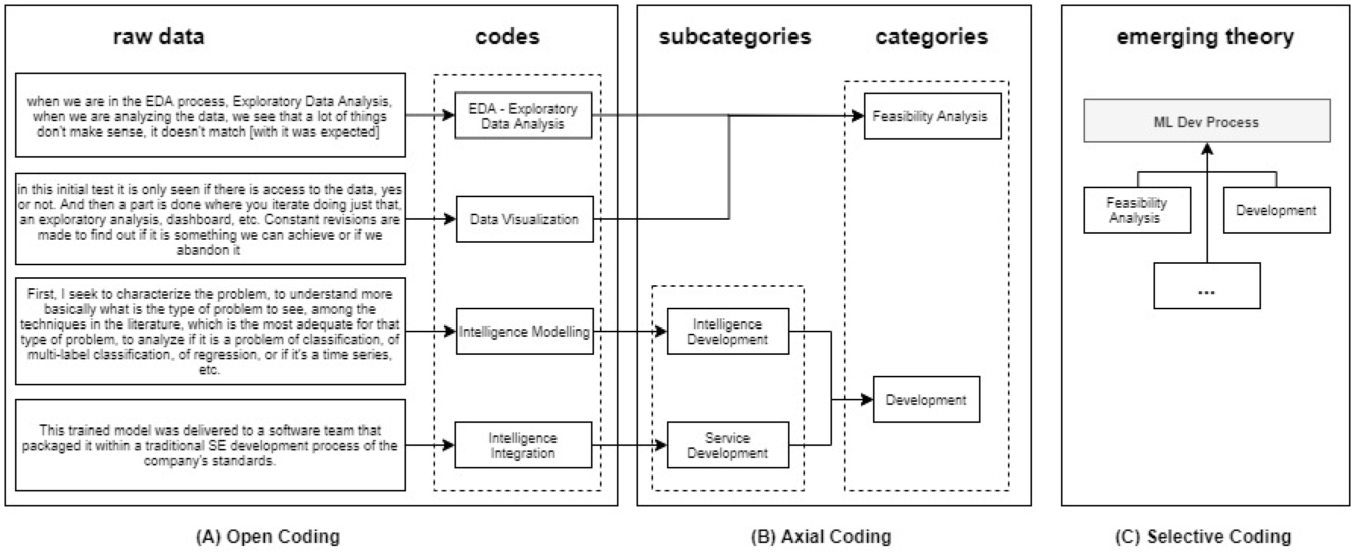
raw data
codes
subcategories
categories
emerging theory
when we are in the EDA process, Exploratory Data Analysis, when we are analyzing the data, we see that a lot of things don't make sense, it doesn't match [with it was expected]
in this initial test it is only seen if there is access to the data, yes or not. And then a part is done where you iterate doing just that, an exploratory analysis, dashboard, etc. Constant revisions are made to find out if it is something we can achieve or if we abandon it
First, I seek to characterize the problem, to understand more basically what is the type of problem to see, among the techniques in the literature, which is the most adequate for that type of problem, to analyze if it is a problem of classification, of multi-label classification, of regression, or if it's a time series, etc.
This trained model was delivered to a software team that packaged it within a traditional SE development process of the company's standards.
EDA - Exploratory Data Analysis
Data Visualization
Intelligence Modelling
Intelligence Integration
Intelligence Development
Service Development
Feasibility Analysis
Development
ML Dev Process
Feasibility Analysis
Development
...
(A) Open Coding
(B) Axial Coding
(C) Selective Coding

Figure 1: Application of Strauss-Corbinian Grounded Theory procedures

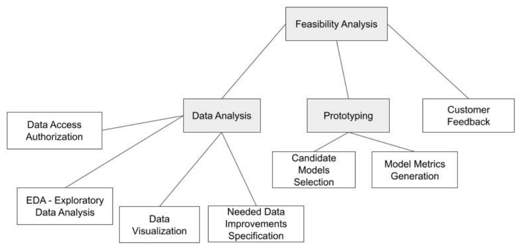
Feasibility Analysis
Data Access Authorization
Data Analysis
Prototyping
Customer Feedback
Candidate Models Selection
Model Metrics Generation
EDA - Exploratory Data Analysis
Data Visualization
Needed Data Improvements Specification

Figure 2: Feasibility Analysis - Subcategories and Codes

As the most of the development process models, the proposed phases and activities may be adapted and used or not according to the project needs. The main point is that ML projects are often so much riskier and, for that reason, the feasibility analysis should take more time and attention than most traditional software projects.

4.3 Phase 2 - Development

As soon as the project feasibility risk is reduced and the project is considered feasible, the Development Phase starts. This phase is composed of three branches: *Data Pipeline Development*, *Intelligence Development*, and *Service Development*, organized according to the technical skills needed to perform the activities that compose each branch.

Figure 3 represents the *Development Phase* branches and activities. Although many activities can be executed parallelly, the arrow from *Data Provisioning* to *Intelligence Development* represents that the provisioning of new data should trigger some *Intelligence Development* activities such as (but not only) model training and model test. The other arrows represent that for new models generated by the *Intelligence Development*, when requested, they should be validated by the QA and, if approved, integrated into the service. Before service deployment, the service with the new integrated model should also be validated by the QA.

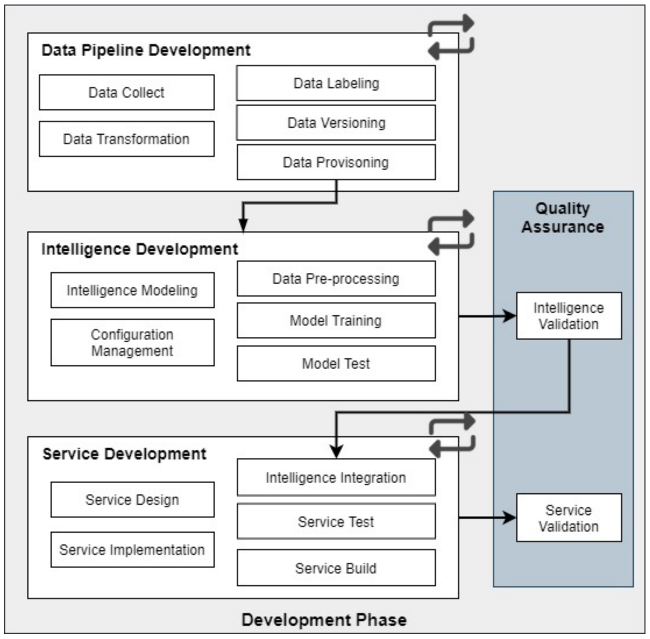

Figure 3: ML Developement Process - Development Phase

This scheme of branches was designed to be parallelly executed and with low coupling. Among them, some activities can generate inputs that will necessarily trigger the execution of activity of another branch. However, each branch has an internal cyclic behavior, independent of the other phases, as an iterative and incremental process. For instance, in the *Intelligence Development* branch, it is possible to run the *Intelligence Modeling*, *Model Training* and *Model*

Test using the same data set version. The next paragraphs detail each branch and its respective activities.

Data Pipeline Development branch consists of all the activities related to collect and manage data, and finally set them available if they meet the expected quality for ML solution modeling and models training. It is commonly called **pipeline** because the sequence of tasks performed from the first access to the raw data until achieving the ideal data format and composition, is often modeled as a Directed Acyclic Graph (DAG) in which the next task processes the data resulting from a preceding task processing.

According to the reports, the most important concepts involved in the data pipeline are:

- **Data Collection**: Access the needed raw data to be processed by the data pipeline.
- **Data Transformation**: Apply changes on data in order to obtain new data with specific formats and characteristics (e.g.: *Data Augmentation* by applying filters on images, *Synthetic Data Generation* based on existing data).
- **Data Labeling**: For supervised learning approaches, in special, labeled data are essential for models construction. Data labeling can be performed by humans or using automatic techniques (e.g.: Active Learning). In some cases, the collected data comes labeled, anyway, improvements can be necessary.
- **Data Versioning**: Rigid control of the dataset versions that allow retrieving the data as they were in a specific point of time. Depending on the size of the datasets, it is highly recommended a careful definition of tools and control points in order to reduce the storage consumption.
- **Data Provisioning**: Consists in control of datasets regarding collecting method, integration to organizational processes, and accessibility. Commonly some quality criteria such as data quality, reliability, integrity, security, availability, accessibility, recoverability are defined and, based on that, datasets can be classified at different levels (e.g. Bronze, Silver, and Gold is a common convention for this classification)

The presented activities should be analyzed and used to base the design of the data pipeline. Ideally, the data to be provided to Intelligence Development activities should have the expected quality, should be versioned, and a clear mechanism to access the data ready to use should be defined.

Intelligence Development is the branch of the ML development process that embraces the production, management, and provisioning of ML models. It has a high dependency on data that should be provided by the data pipeline. When a minimum data management policy is not defined with, at least, a data version control system, critical issues such as feasibility to reproduce training can occur. The following activities were identified by the reports of the interviewees:

- **Data Pre-processing**: Embraces data manipulation that results in data changes, but that can not be done by the data pipeline for any reason. Commonly, the data that will be processed by the resulting model in production needs to be modified before being processed by the model (e.g.: *feature extraction*).
- **Intelligence Modelling**: Defines all the aspects related to the intelligent model construction, involving the definition of the most suitable approaches and techniques to solve the problem, the needed data, features selection, and other needed parameters based on the project's requirements (e.g.: problem type, expected accuracy, inference time)
- **Model Training**: Consists on the implementation of a procedure (using programming language or automatic tools) following the specifications defined by the intelligence modeling. Using the provisioned data, the developer trains ML models using specific techniques and follows the best parameter for that context. As soon as you realize that changes in the training procedure are not needed to retrain the model, this activity is often automated without human interventions.
- **Model Test**: Commonly executed together or immediately after each training, this activity in this phase consists in verifying models improvements based on the generated metrics. When the tests indicate that a better model was achieved, the generated model should be versioned and provisioned for a quality assurance (QA) process.
- **Configuration management**: A horizontal discipline related to control of environments and dependencies. It deserves a great relevance for intelligence development in order to guarantee the reproducibility of the model generation. For most cases, it is extremely important to guarantee that all the artifacts, parameters, and dependencies necessary to rebuild the model are versioned and related to the model version.

Organizing these activities sequentially by the level of dependency among them, the intelligence modeling and the configuration management policy should be defined initially. Once there are data to perform training, it is possible to train and test models. When a model is considered ready, it should be versioned and, in an ideal scenario, sent to the quality assurance process for validation.

Quality Assurance (QA) for ML projects was reported as a quite immature area by the interviewees. For most of them, although they had reported it as a necessary and important process, it is no formally defined or executed in their contexts. For that reason, this emerging theory does not establish yet the concepts and practices that compose such area. Considering the role of the QA on traditional software development, it should determine if a model is ready or not for deployment in a specific environment (e.g. stagging or production, alpha, beta, or release to market). Being approved by the QA process, the model is ready to be deployed. However, it is often embedded into a service that is managed in the context of the Service Development branch.

Service Development branch works very similarly to traditional software because it is a common software that receives input data, uses the ML model to analyze those input data, and produces an output based on the output produced by the model. In this context, there is a large variety of data types (e.g. text, audio, video), protocols (e.g. HTTP for Rest Services, RTMP for video streaming), and architectures (e.g. synchronous request, event-based system, batch processing) that can be used depending on the solution objective.

The identified activities of this branch presented below, as well as the previous ones, are independent of the technical aspects of the project:

- **Service Design**: Defines all the aspects related to the intelligent service, involving the definition of components communication, technologies to be used, communication protocols, and user interface based on the project's requirements related to how the intelligence should be provided and how to embed and use the intelligent model.
- **Service Implementation**: It is the construction of the service by the implementation of the specifications defined by the service design.

- **Intelligence Integration**: An intelligent service should be prepared to load intelligent models with low effort. For every new approved model, it is necessary to integrate the model into the service and provide the service now using the new model;
- **Service Test**: Service tests are traditional software tests created and executed during the development such as unit tests and integration tests. These tests should focus on ensuring the correct functioning of the service, excluding aspects related to the ML model (that should be guaranteed by the intelligence tests and validation);
- **Service Build**: Also quite similar to traditional software build that consists in generating one or more artifacts ready to run on a target environment (e.g. compile a C code and generate an executable file for Linux). In this case, it is possible that a new build is necessary for every new model or that a new build is only necessary if the service itself has changed.

5 CONCLUSION

Developing software that integrates AI capabilities requires a process which includes a set of activities, issues and aspects not commonly found in traditional software development. Based on the analysis of empirical data regarding the experiences of practitioners involved in machine learning projects, the emerging theory presented in this paper comprises a set of key elements organized in 3 phases with their corresponding activities.

The most important findings of this study so far are (1) , due to the high risk inherent to ML projects, there is a strong indicative that the project's conception phase is more complex and the feasibility risk is higher than in tradicional software projects because of the many contextual problems like the high customer expectation and the needed datasets, (2) the traditional quality assurance practices are not suitable for ML projects and there is a lack of maturity comparing to the same area in traditional software projects, and (3) the understanding that the development process for ML projects can not be modeled as a unique cycle of activities that are executed by the single team.

The development process presented in this paper intends to solve those problems by the definition of *Feasibility Analysis* phase, which is an intense conception phase composed by a set of activities that approaches the most common problems and risks of ML Projects, and a *Development Phase* composed by separated development branches that aggregate activitivies with high dependency and allow independent iterative activity cycles for each branch.

This paper summarizes the results of the study considered mature enough to be published. The authors' decision about which elements of the emerging theory would be reported relied on the level of details shared by the participants. One of the phases identified in the study (the operation phase) was not approached because additional data need to be collected and analyzed to enable a complete understanding of this aspect regarding how ML-based applications are developed.

With regard to limitations and threats to validity, the coding procedures were executed by the first author and revised by two other researchers to prevent bias. The limited number of interviews also influences the results. The next paragraph better explains what this means for theory developing and points out future actions to mitigate this limitation.

As Grounded Theory guidelines determine, data collection and analysis are performed simultaneously until theoretical saturation is reached, meaning that no new concepts or insights emerged, which indicates that the researcher can stop collecting data. So, as future steps, new interviews should be conducted to enable the rise of new insights on the theory until theoretical saturation is reached. Also, a more in-depth discussion of the findings in light of the literature on the topic is required.

Finally, the contribution of this study becomes relevant as a theory starts to take shape from empirical data analyzed in a systematic way that makes it possible to understand the peculiarities of how intelligent software is developed. Despite the efforts made by the software engineering community, no study proposed a reference development process for ML-based applications.

REFERENCES

[1] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 291–300.

[2] Kathy Charmaz. 2006. *Constructing grounded theory: A practical guide through qualitative analysis*. SAGE Publications, London.

[3] Charles Hill, Rachel Bellamy, Thomas Erickson, and Margaret Burnett. 2016. Trials and tribulations of developers of intelligent systems: A field study. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 162–170.

[4] Rukayya Sunusi Alkassim Ilker Etikan, Sulaiman Abubakar Musa. 2015. Comparison of Convenience Sampling and Purposive Sampling. *American Journal of Theoretical and Applied Statistics* 5, 1 (2015), 1–4. https://doi.org/10.11648/j.ajtas.20160501.11

[5] Mirko Perkusich, Lenardo Chaves e Silva, Alexandre Costa, Felipe Ramos, Renata Saraiva, Arthur Freire, Ednaldo Dilorenzo, Emanuel Dantas, Danilo Santos, Kyller Gorgônio, et al. 2020. Intelligent software engineering in the context of agile software development: A systematic literature review. *Information and Software Technology* 119 (2020), 106241.

[6] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2017. Data management challenges in production machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1723–1726.

[7] Nayan B. Ruparelia. 2010. Software Development Lifecycle Models. *SIGSOFT Softw. Eng. Notes* 35, 3 (May 2010), 8–13. https://doi.org/10.1145/1764810.1764814

[8] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded theory in software engineering research: a critical review and guidelines. In *Proceedings of the 38th International Conference on Software Engineering*. 120–131.

[9] Anselm Strauss and Juliet Corbin. 1990. *Basics of qualitative research*. Sage publications.

[10] Claes Wohlin and Aybüke Aurum. 2015. Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Software Engineering* 20, 6 (2015), 1427–1455.

[11] Tao Xie. 2018. Intelligent software engineering: Synergy between AI and software engineering. In *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*. Springer, 3–7.