NOVA IMS Information Management School

# MGI

Master Degree Program in

**Information Management**

# A FRAMEWORK FOR LEVERAGING ARTIFICIAL INTELLIGENCE IN SOFTWARE DEVELOPMENT

Diogo de Oliveira Bento Martins

Dissertation

presented as partial requirement for obtaining the Master Degree Program in Information Management

**NOVA Information Management School**
**Instituto Superior de Estatística e Gestão de Informação**
Universidade Nova de Lisboa

**NOVA Information Management School**
**Instituto Superior de Estatística e Gestão de Informação**
Universidade Nova de Lisboa

# A FRAMEWORK FOR LEVERAGING ARTIFICIAL INTELLIGENCE IN SOFTWARE DEVELOPMENT

By

Diogo Martins

Master Thesis presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Information Systems and Technologies Management

**Supervisor:** Prof. Vítor Manuel Pereira Duarte dos Santos, PhD

June 2024

# STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Rules of Conduct and Code of Honor from the NOVA Information Management School.

*Diogo Martins*

*[Lisbon, Portugal, June 13, 2024]*

# ACKNOWLEDGEMENTS

I would like to thank all the people who impacted this work, directly and indirectly, and whose support was essential in this chapter of my life.

Firstly, I would like to thank my parents for always being there when I needed them most. Your unwavering support and sacrifices throughout my life could not go unmentioned in this acknowledgement. Thank you for your patience with me and helping me move forward. I would also like to thank all my grandparents, especially my late grandfather, who parted a few weeks before I started my master's degree.

My long-term friends from my hometown and those I met during my bachelor's degree also deserve recognition. Thank you for supporting me and providing much needed distractions to keep me motivated and clear-headed.

I would also like to show my gratitude to the experts who participated in the evaluation of this dissertation. Both your contribution and your willingness to dedicate your personal time on sometimes inconvenient occasions, were recognized and appreciated by me.

Lastly, but not least, I would like to thank Professor Victor dos Santos, my advisor in the preparation of this dissertation, whose advice was essential for its construction. Your consistent positive attitude, approachability and, above all, willingness to help, have been appreciated by me, in this stressful and novel time, directing me one step at a time towards the completion of a work that I hope will be successful.

# ABSTRACT

Due to recent breakthroughs in Artificial Intelligence, we have seen an increase in its capabilities, which has also bolstered an unprecedent potential for automation, as these technologies can also automate or assist in tasks innate to human intellect and creativity. As a result, these technologies are expected to redefine many professional occupations, increasing their overall productivity, along with numerous other advantages elucidated in this study. This has fueled an increase in enthusiasm for AI, both from academia and corporations, as can be seen in the contemporary increase in articles investigating the inclusion of AI in many fields, and in AI-based products available on the market, respectively. However, there is a shortage of prescriptive tools that advise entities on their inclusion in the software development process. To address this gap, this study employs a thorough literature review along with Design Science Research to uncover the way these two fields mutually interact. The main result of this dissertation is a novel framework that advises on the most appropriate algorithms for the different phases of the software development life cycle.

# KEYWORDS

Artificial Intelligence; Software Development; Software Development Lifecycle; Machine Learning; Automation

## Sustainable Development Goals (SGD):

# INDEX

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| **A1DE** | Average One Dependency Estimator |
| **AdaBoost** | Adaptive Boosting |
| **AI** | Artificial Intelligence |
| **ANN** | Artificial Neural Network |
| **BiLSTM** | Bidirectional Long Short-Term Memory |
| **BERT** | Bidirectional Encoder Representations from Transformers |
| **CDT** | Credal Decision Trees |
| **CHIRP** | Composite Hypercube on Iterated Random Projection |
| **COCOMO** | Constructive Cost Model |
| **CNN** | Convolutional Neural Network |
| **DL** | Deep Learning |
| **DSR** | Design Science Research |
| **DT** | Decision Tree |
| **GA** | Genetic Algorithms |
| **GUI** | Graphical User Interface |
| **KNN** | K-Nearest Neighbors |
| **Lin R.** | Linear Regression |
| **Log R.** | Logistic Regression |
| **LSTM** | Long Short-Term Memory |
| **ML** | Machine Learning |
| **NB** | Naïve Bayes |
| **NLP** | Natural Language Processing |
| **RAD** | Rapid Application Development |
| **RF** | Random Forest |
| **RNN** | Recurrent Neural Network |
| **SDLC** | Software Development Lifecycle |

| | |
|---|---|
| **SLRQ** | Systematic Literature Review Question |
| **SVM** | Support Vector Machine |
| **TF-IDF** | Term Frequency – Inverse Document Frequency |
| **UML** | Unified Modeling Language |
| **Word2vec** | Word to Vector |
| **XGBoost** | Extreme Gradient Boosting |

# 1. INTRODUCTION

## 1.1. BACKGROUND AND PROBLEM IDENTIFICATION

Artificial Intelligence is not a recent notion. The first proper definition of AI was made in 1956 by John McCarthy, one of its pioneers, who defined it as "the science and engineering of making intelligent machines" (McCarthy et al., 2006). Subsequently, research in artificial intelligence has been gradually progressing until new breakthroughs were achieved recently. Currently, with advances in other symbiotic fields such as an increase in computer power potentiated by cloud computing, advancements in data collection techniques, access to new and powerful algorithms, etc., permitted new AI techniques and tools to flourish (Yakubu, 2022). These technologies made experimentation cheaper and faster, which originated in an outburst of AI potential promising to transform work in a global scale (Yakubu, 2022; Yarlagadda, 2017).

In this sense, and as with many other digital technologies, AI created new opportunities for task automation within companies, providing an overall increase in efficiency (Yarlagadda, 2017). However, AI provides an unprecedented potential to revolutionize future operations within a company. Until now, the processes that could be automated, whether through the introduction of machinery or, more recently, through digitization, were repetitive low-skilled tasks associated with blue-collar or white-collar work (Acemoglu & Restrepo, 2018b). Artificial Intelligence, however, possesses the unique ability to assist or even replace high-skilled occupations, even those related to qualities innate to human intelligence, such as creativity, abstraction, judgement, and analysis (Acemoglu & Restrepo, 2018b).

The high automation potential provided by AI, along with the numerous other advantages conferred by its adoption is capturing the awareness of companies due to the transformative capabilities of AI mentioned above. According to research, AI adoption has become widespread, with a rate of thirty-five percent of companies in 2022 reporting to be using AI in their business, 4% higher than the previous year. Additionally, forty-two percent of companies reported that they are exploring the use of AI within their processes (*IBM Global AI Adoption Index*, 2022). It is reasonable to say that there is a newfound enthusiasm for its inclusion in its normal operations.

Consequently, as with all technological advances that can be used to automate processes, there is a decrease in the demand for certain jobs, but also an increase in demand for others (Acemoglu & Restrepo, 2018a). The increase in digitalization trends characteristic of industry 3.0 and 4.0 (Lasi et al., 2014), is also contributing to a greater demand for software in the market. Logically, this is also accompanied by an increase in the need for software developers, driven by technological progress that creates a demand for professionals with updated skillsets (Acemoglu & Restrepo, 2018a). Because of this, we are witnessing a worldwide shortage of software developers, with a shortfall of forty million professionals, and this trend is expected to double by 2030 (Smith, 2023; *Software Developer Shortage in the US and Global Tech Talent Shortage in 2022*, 2022). Paradoxically, one of the measures that can address this software developer shortage could be the use of software, either to automate or assist in the various phases of the development process.

A growing interest in this field, both by academia and corporations, has also contributed to many new explorations in AI, driving innovations in countless industries in the present landscape, with software development being one of such fields. Innovation in this department has allowed the surge of

numerous technologies, such as generative AI, but its applicability in the field of software development is still at an early stage. There is a lack of research regarding the recommendation of best practices for the integration of AI technologies in this area, and there is still no existence of a holistic methodology for its insertion in the various stages of software development, according to the research reviewed at this stage of this work (Korzeniowski & Goczyła, 2019).

In other words, despite the existence of a push by companies to adopt AI, as well as an interest by academics in exploring its limits, the broad use of AI in software development is still considered, as expressed by Korzeniowski & Goczyła (2019), "*terra incognita*". This marks an opportunity to investigate their applicability to the software development process as a whole.

The following study aims to formulate a framework, with the intent of assisting entities in identifying AI integration opportunities in accordance with their specific needs, enabling them to take advantage of current AI advances and allowing them to decrease the number of working hours and resources to produce a unit of output. For this purpose, numerous AI technologies are surveyed with the objective of finding suitable technologies to assist/ automate a specific step of the software development life cycle.

These facts lead to the formulation of the following research questions:

- What are the impacts of applying AI to the software development process?
- How can we help software development companies take advantage of AI?
- What AI technologies are appropriate for each phase of the software development life cycle?

## 1.2. OBJECTIVES

The end goal of the research is to help answer the research questions listed, proposing a framework to assist companies that wish to use Artificial Intelligence in their software development process.

To achieve this goal, the following objectives were defined:

- Understand and outline the different steps of the software development process.
- Outline the existing uses of artificial intelligence in software development.
- Identify AI tools and techniques compatible with the goal.
- Understand the use of AI technologies and their implementation.
- Propose a framework.
- Validate the framework.

## 1.3. IMPORTANCE AND RELEVANCE

As a rule, automation provides many benefits to those who integrate it properly, which ultimately provides an increase in productivity and an increase in business competitiveness (Korzeniowski & Goczyła, 2019). Software development, despite the methodology used, is mainly a manual process. As a consequence, it is a very expensive and error-prone process (Korzeniowski & Goczyła, 2019). Automation solutions could contribute to mitigate the problems presented, as they assist the human element or eliminate it entirely, making tasks almost immediate and providing reliable and cost-efficient solutions.

Ongoing fervor for the digital transformation of AI is also accompanied by the existence of generally recognized barriers for entities wishing to integrate it into their processes. The biggest barrier identified by companies was the lack of knowledge and expertise in projects, felt by thirty-four percent of companies that tried to adopt AI in their processes (*IBM Global AI Adoption Index*, 2022). There is also a lack of tools for effective AI integration, advocated by twenty-five percent of companies (*IBM Global AI Adoption Index*, 2022).

A framework is expected to address the previously mentioned barriers and allow companies to better understand the different technologies available and how they will be better applied, so that they can achieve the desired outcomes. The general nature of this work also provides a simple guideline that can be followed by any company wanting to integrate AI into their development process. The benefit of a simple and comprehensive work is the versatility of its application, providing parties who are interested in transforming their processes with an approach that will allow them to make the most of current technological breakthroughs, regardless the industry in which they are inserted.

Another critical factor to be considered as a way to justify the importance of this research is the current global shortage of developers. As of 2021 there was a deficit of forty million skilled workers, which is expected to double by 2030 (Smith, 2023; *Software Developer Shortage in the US and Global Tech Talent Shortage in 2022*, 2022), which could bear dire economic consequences, as ninety percent of operations in a typical organization are supported by software. If this trend continues, companies around the world risk losing around eight trillion dollars (Smith, 2023; *Software Developer Shortage in the US and Global Tech Talent Shortage in 2022*, 2022). This phenomenon can be softened by exploring automation through the use of Artificial Intelligence, as presented in this dissertation.

Finally, this study is also of interest to the scientific community. The expansion in AI capabilities has also caused an expansion in the scope of its use into new areas, but, as it is common in emerging fields, there is a scarcity of bibliography relating to numerous topics in this area, with is applicability to software development being one such topics. As such, this project will be beneficial, not only to companies wanting to adopt AI in software development, but to everyone who demonstrates an academic interest in the field, enabling them to build upon the presented work.

## 2. METHODOLOGY

The final proposition for a framework requires a systematic approach that provides guidance to achieve the proposed objectives. Note that a framework falls under the definition of an artifact and therefore requires a methodology designed specifically for creating artifacts. For this reason, the methodology chosen was Design Science Research (DSR), using both the framework provided by Hevner's (2007) research, presented in Figure 1, as well as Peffer's et al. (2006) stepwise guidelines for the research strategy.

DSR proves itself to be noticeable compatible to the case at hand, as it proposes guidelines that assist in the process of creating the framework (Brocke et al., 2020). The final artifact aims to provide support to companies; thus, it is imperative for the acknowledgement of environmental factors in order to understand the needs of their business. The importance of these factors is such that it was the main reason for the choice of this framework, in addition to the guidelines proposed by Peffers et al. (2006), which does not consider them. A further theoretical exposition of Design Science Research will be presented in the following section.

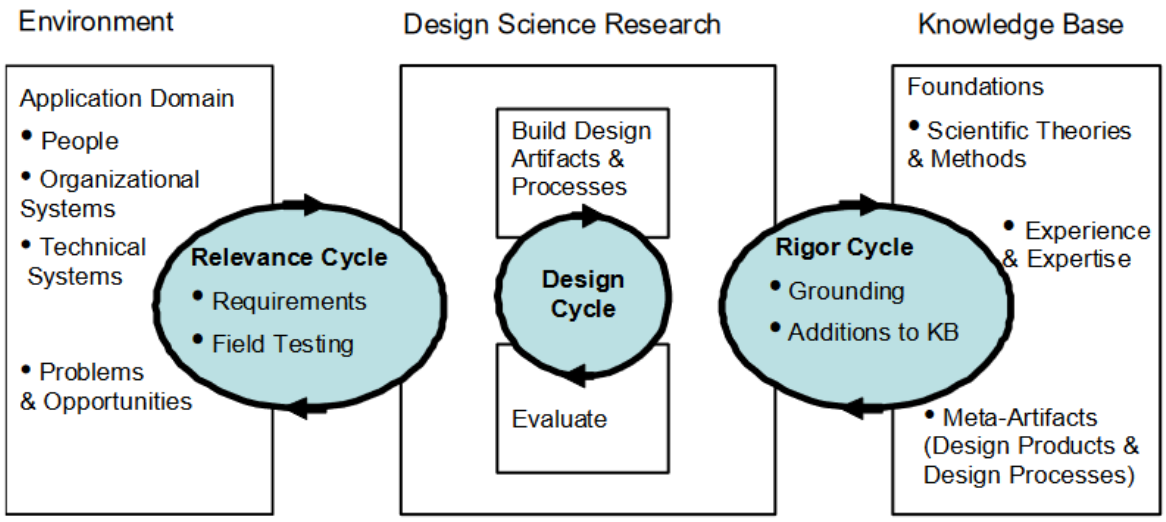


Figure 1 – Design Science Research Framework (Brocke et al., 2020)

## 2.1. Design Science Research

Design Science Research is defined as "a problem-solving paradigm that seeks to enhance human knowledge via the creation of innovative artefacts" (Brocke et al., 2020). It recognizes the need for a cyclical complementary approach between Design Science and Natural Science, as the second is mostly incapable of dealing with "wicked organizational problems" (i.e., problems that require creative and innovative solutions) (Hevner et al., 2004) without the first. This cyclical relation unrolls as follows:

- Natural science provides Design Science with the theoretical knowledge necessary to create artifacts (Hevner et al., 2004).
- Design Science, in turn, generates the artifacts, returning them to the Natural Sciences so that they can increase the general pool of knowledge (Hevner et al., 2004).

As noted earlier, this approach consists of providing criteria that merit consideration during the design process, as shown in Table 1. It also focuses on three cycles of research: the relevance cycle, the rigor cycle, and the design cycle. The Rigor and Relevance Cycles are included in the domain of Natural Sciences and the Design Cycle resides in Design Science (Hevner, 2007). Each cycle is further explained in the following subsections.

Table 1 – Design Science Research Criteria (Hevner et al., 2004)

| *Guidelines* | *Descriptions* |
|---|---|
| 1. Design as an Artifact | *"Design science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation"* |
| 2. Problem relevance | *"The objective of design science research is to develop technology-based solutions to important and relevant business problems"* |
| 3. Design evaluation | *"The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods"* |
| 4. Research contributions | *"Effective design science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design"* |
| 5. Research rigor | *"Design science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact"* |
| 6. Design as a search process | *"The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment"* |
| 7. Communication of research | *"Design science research must be presented effectively to both technology-oriented and management-oriented audiences"* |

### 2.1.1. Relevance Cycle

The Relevance Cycle originates from the DSR's desire to "improve the environment by the introduction of new and innovative artifacts and the process of building these artifacts" (Hevner, 2007). In this setting, the environment encompasses all stakeholders who will benefit from the use of the final artifact, including people, organizations, and even technologies, that interact with it. Therefore, the relevance cycle is considered the starting point for design research, as it is pivotal for the identification of the needs of these stakeholders. Through its study it is possible to gain a better understanding of the research problem and define requirements for the research itself (Hevner, 2007).

The initial outputs of design science derive from the needs and constraints of the environment, needing to be released into the environment before being claimed as the final artifact. The results of the artifact's testing in the environment will determine whether the requirements have been met, the companies' needs have been met, and whether its performance is satisfactory. If these conditions are not met, then the artifact is considered inadequate, and the design of another iteration is then initiated (Hevner, 2007).

### 2.1.2. Rigor Cycle

In the rigor cycle, past knowledge is collected, studied, and employed to serve as a basis for building the DSR output. This knowledge is composed of two categories of resources. This is known as foundations, which includes theoretical knowledge that justify the drawn conclusion in the DSR process, frameworks, methods, instantiation, and others. The methodologies used and which provide the ground rules for the design process are also considered essential knowledge acquired at this phase (Hevner, 2007).

The presence of a cycle that continually seeks relevant academic knowledge to support DSR contributes to the inclusion of relevant information as justification for the new knowledge created, and to improve existing parts of a solution. In other words, it contributes to the creation of innovative solutions to perceived problems, as well as to the improvement of existing solutions from other sources or iterations arising from the design process (Hevner, 2007).

### 2.1.3. Design Cycle

The design cycle is fundamental to Design Science Research. It is through this process that various design science iterations are produced through inputs from collected requirements and factual knowledge gathered from the rigor and relevance cycles, respectively. Such iterations are subjected to a meticulous testing and evaluation process that provides relevant feedback (Hevner, 2007).

The design cycle requires multiple iterations, which rightly implies that initial iterations are rejected. In this case, both the rigor and relevance cycles may be revisited according to the acquired feedback so that an alternative iteration may be generated. Then, the newly acquired design knowledge is again subjected to the design cycle. This process will be repeated until a satisfactory iteration is found, with the aim of ensuring the formulation of knowledge that assures maximum levels of effectiveness and efficiency (Hevner, 2007).

## 2.2. Research Strategy

For the development of the proposed artifact, the aforementioned DSR framework and the research criteria of Hevner (2007) and Hevner et al. (2004) will be considered for the entire process. However, as shown in Figure 2, Peffers et al. (2006) provide a step-by-step guideline that will be followed as the strategic approach for the making of this master's dissertation.

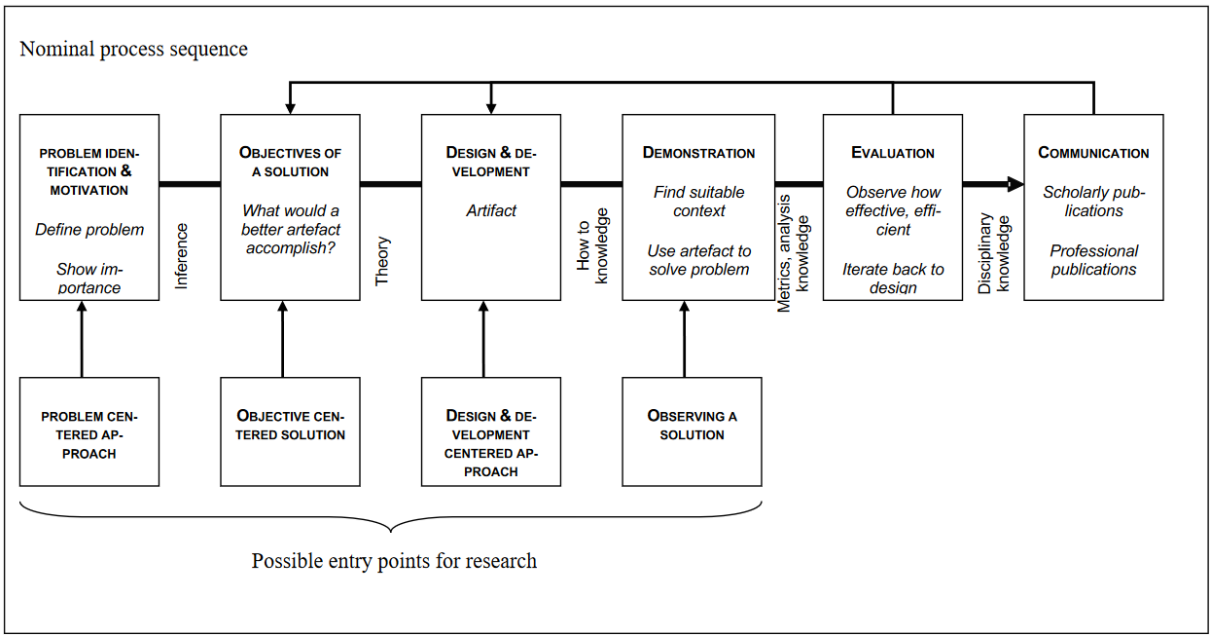


Figure 2 – DSR Methodology Process Model (Peffers, Tuunanen, Gengler, & Rossi, 2006)

The complementary use of both approaches is expected to be beneficial in the making of this study. Design Science Research guidelines provide a systematic step-by-step process for the artifact generation process in its entirety. The DSR framework contributes with three cyclical perspectives for gathering information and building iterations.

The making of this dissertation will function as follows:

1. **Problem Identification and Motivation** – The initial step of building the framework involves defining the research problem and rationalizing the benefit of a solution. This was accomplished in the introductory section of this study, where a brief exposition of the problem was demonstrated as context, as well as how the study can benefit the many stakeholders who may interact with it.

2. **Objectives of a Solution** – The definition of general objectives is essential to offer an *ad hoc* normative criterion that, when met, can mean the success of the research. In the first section, it was described what the final framework is expected to achieve, the importance of this study and the various generalized guidelines are listed in the "Objectives" subsection.

3. **Desing and Development** – The objective of this phase is straightforward: the generation of the new artifact. This includes gathering knowledge before the design cycle is performed. For this effect, a literature review will be performed with the aim of researching and collecting all relevant information. During the writing of the literature review, the aforementioned Rigor and Relevance Cycles will be continuously consulted. For the production of the artifact itself, the Design Cycle must be used. Note that during the "Problem Identification and Motivation" step an environmental analysis was initiated in the introductory section of this study. However, the information presented in this section is only a fraction of what is needed, used only as a justification for the research.
   Firstly, for the literature review, a more detailed overview of software development and artificial intelligence will be conducted individually, in the context of the rigor cycle. To this end, information will be searched throughout the web, preferably from trustworthy sources, such as Google Scholar. Then, a systematic literature review will be conducted to discern both environmental factors that need to be taken into consideration and existing AI algorithms that are applicable to the software development process. These algorithms will be researched, elaborated, and analyzed to better understand their potential placement in the future framework.
   Both the relevance and rigor cycles are not linear, as it might be perceived in this strategic exposition, but rather interdependent. During the research of factual information, as a basis for the creation of new knowledge and justify new theories, new unforeseen environmental factors may arise, requiring further investigation in the relevance cycle and vice versa.
   After an acceptable level of information has been acquired through the Literature Review, the construction of the artifact will begin, as explained in the design cycle.

4. **Demonstration** – The demonstration phase consists of applying the artifact to the environment for testing and evaluating purposes. Unfortunately, this step cannot be applied as intended, due to network and time constraints. Alternatively, it will be presented to various experts for evaluation purposes. Through their expertise they will be able to theoretically predict the success the artifact would acquire if it was applied to a real case study.

5. **Evaluation** – During the evaluation, as described in the previous step, the proposed framework will be scrutinized. At this phase, even if the artifact is denied, the provided feedback will contribute to refine the final proposal, in order to provide the best possible degree of effectiveness and efficiency to the work. If this happens, further information collection might be necessary, which will require a review of the third step.

6. **Communication** – The final phase of the DSR process consists of the communication, acceptance and publication of this study in the NOVA IMS public repository (run.unl.pt). Furthermore, it is aimed to make a scientific indexed publication with this paper's results.

# 3. LITERATURE REVIEW

In this section, as previously stated in the Methodology chapter, a Literature Review will be carried out with the aim of retrieve appropriate information regarding the topics under study. The creation of a framework that helps the integration of artificial intelligence into software development represents an overlap between these two individual fields. Therefore, these two subjects require an individual exploration, in order to accumulate sufficient understanding of each to substantiate the theories to be presented in the section of this work where the solution is.

## 3.1. SOFTWARE DEVELOPMENT PROCESS

### 3.1.1. Concepts

Software development is a multidisciplinary process through which the desired information system output is generated (Zelkowitz, 1978). However, to create an effective, consistent and high-quality system in a timely manner, it is necessary to use a methodological approach to software development (Acharya & Sahu, 2020). Such methodological approach is called “Software Development Lifecycle”, or SDLC, and comprises all stages of software development, from its inception to its launch and subsequent maintenance (Ruparelia, 2010).

There is a lack of a universally accepted SDLC for all scenarios, existing instead a plethora of models available for use, each suited for particular conditions (Acharya & Sahu, 2020). However, after research, it was concluded that many of the similarly named phases of each SDLC model operate in a similar way. Therefore, to avoid future redundancy in the “Main Approaches” subsection, where each SDLC model will be outlined, in this section only the phases of the Waterfall Model will be conceptually described, as it was the first proper SDLC model, underpinning all subsequent models since its inception (Ruparelia, 2010). Any conceptual deviation from the defined phases is addressed in the corresponding exposition of the SDLC model in question. The waterfall phases are:

1. **Requirements Gathering and Analysis** – This phase consists of information representing the specifications of the commissioned software, provided by the client and agreed with the contracted entity (Alshamrani & Bahattab, 2015). It generates documentation that contains such requirements, providing information about the software needs, goals, and constraints to be taken into consideration in the development of the system.
   The unlikelihood of the requirements being fully stated before the software is concluded must merit consideration. Some requirements or constraints are only identified during the many phases of the model, given the difficulty in predicting some challenges and opportunities in a given project. Other changes originate from the client’s fickle nature who often demands changes mid-project, and also often has unrealistic expectations and provides ambiguous requirements (Alshamrani & Bahattab, 2015; Kramer, 2018).

2. **Design** – The planning for a proper implementation of the acquired information is done according to the requirements, resulting in an architecture for the system. This involves: High-level design, which concerns the creation of an architecture for the general software solution, through the division of the software into modules and the correlation between them. It also concerns a specification of the necessary resources, such as software, hardware and the need for skills/ a team (Alshamrani & Bahattab, 2015; Kramer, 2018).
   There is also low-level design at a component level, thus detailing the components of the high-level design, containing interface details, error message listings, dependency issues, inputs and outputs for each module (*SDLC Waterfall Model*, n.d.).

3. **System Implementation and Coding** – This phase, as the name implies, is the one in which the code is written, in accordance with the asserted specifications and system design. Although this phase seems straightforward, this is the longest phase of the SDLC (Kramer, 2018). This is where the developers create the code from the two other steps and the approved design. Programmers can use different tools, such as compilers, interpreters, and debuggers, to generate the code, as well as the programming languages asserted in the previous phase (Kramer, 2018).

4. **Testing** – In order to decide if the system is ready to launch, it is tested according to predefined performance measures, in order to determine whether it provides the desired efficiency and quality. It also determines whether the system is faithful to the requirements and the planned design, and also identifies errors to be corrected before launch (Petersen et al., 2009).

5. **System Operation and Maintenance** – After its release, the system may need post-release modifications to provide bug fixes, improvements, or even system refinements. Thus, this phase is the process of addressing such concerns (Alshamrani & Bahattab, 2015; Kramer, 2018).

Furthermore, it is worth highlighting that the Waterfall model, as with most of the models presented, yields a multitude of interpretations, depending on the author who reports it. The chosen interpretation, as displayed in this study, is perceived to be an accurate and comprehensive depiction of the software development process, supported by the following authors: Kramer (2018), Alshamrani & Bahattab (2015), Adenowo & Adenowo (2020).

### 3.1.2. Main Approaches

#### *3.1.2.1.* Waterfall Model

The Waterfall model, as portrayed in Figure 3, was the first SDLC model and therefore the most influential. This model is characterized by its cascading processes, flowing linearly from one completed phase to another, ending in a single big bang release (Kramer, 2018; Ruparelia, 2010). Royce (1970), the first to define this model, also identified the need for feedback loops, with the intent of revisiting the previous phase of the process, if necessary, requiring expensive rewrites of documentation (Ruparelia, 2010).

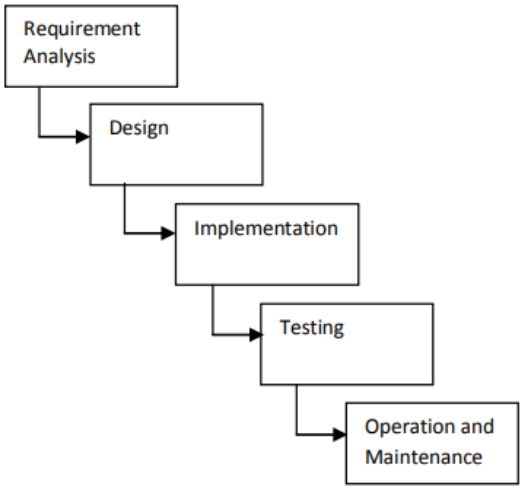


Figure 3 – Waterfall Model (Adenowo & Adenowo, 2020)

The main advantage of this model is that it emphasizes the creation of documentation at every stage, allowing better team communication and quality assurance (Adenowo & Adenowo, 2020; Kramer, 2018). This model is also stronger when considerable time is spent validating the earlier phases of the cycle, since a bug found in these earlier phases requires less capital, time, and effort to fix than in the later stages (Adenowo & Adenowo, 2020).

On the other hand, the Waterfall Model possesses some systematic flaws: with the lack of iteration, this model is inefficient in addressing changes in requirements mid-project, making much of the completed work obsolete in such cases, which results in increased completion time and costs (Kramer, 2018). The lack of prototyping also makes final software delivery difficult, as testing is done at the end of the project, making it difficult to predict challenges and risks in the early phases (Kramer, 2018).

As this SDLC represents the genesis of the field, its flaws have inspired many subsequent development models, with these later SDLCs being preferred over this model. However, it is still used in creating software that provides back-end functionality (Ruparelia, 2010), as well as in smaller projects or if the requirements are testable and well defined (Kramer, 2018).

### *3.1.2.2.* Evolutionary Model

The Evolutionary Model uses the same sequential phases as the waterfall model. However, this model's differentiating characteristic is the segmentation of the "System Implementation and Coding" phase into shorter waterfall model applications, as shown in Figure 4, which also provides a slightly different interpretation of the waterfall phases. Each of these brief waterfall models generates an iteration of the software, which is then presented to the customer (May & Zimmer, 1996). Subsequently, the customer provides feedback that is incorporated into the construction of the subsequent iteration, allowing the software to evolve in the process (Chandra, 2015). This sequential process is repeated until a satisfactory iteration is created, asserting itself as the final product (May & Zimmer, 1996).

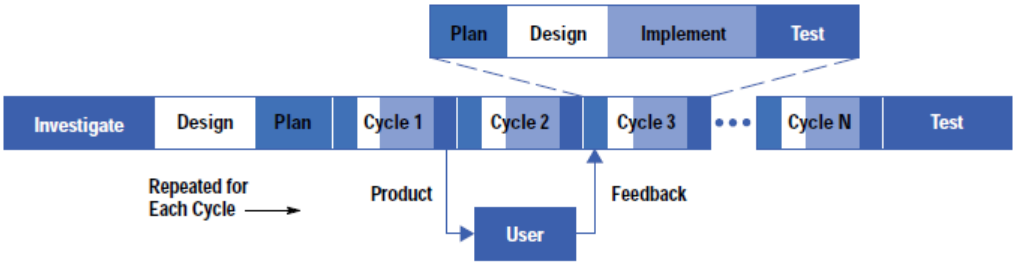


Figure 4 – Evolutionary Model (May & Zimmer, 1996)

The evolutionary model addresses issues related to changes in requirements mid-project and the unpredictability of problems through the iterative building of the system. This, combined with regular customer interaction, results in a significant reduction in risk and higher software quality when compared to the waterfall model (May & Zimmer, 1996). However, its differentiating characteristics can be considered a double-edged sword. Applying several waterfall cycles to develop the final system can make the application of this model time-consuming, causing an increase in costs (Chandra, 2015). Applying this model also requires effective supervision and control, as well as an overall focus on management (May & Zimmer, 1996).

The evolutionary model is appropriate for use in long projects and when the customer prefers to use the main features of the software instead of waiting for a final release (*Evolutionary Model - Software Engineering*, 2019). Its flexibility allows its application in projects where requirements are expected to be particularly volatile. It is also useful when the application requires innovation that is unfamiliar and new to the company and its teams (Adesina et al., 2020).

### *3.1.2.3.* Spiral Model

The spiral model was created as a response to flaws in the waterfall model. In the study that first proposed this model, Boehm (1986) expressed frustration at the lack of evolutionary iteration of the waterfall model, as well as the lack of accommodation to change and difficulty in managing risks in its application. Therefore, he attempts to correct these flaws by creating a risk-oriented approach. In this model, development is achieved through the iterative application of the phases outlined in Figure 5, which reflect the waterfall phases, as previously demonstrated (except the waterfall maintenance phase) (Alshamrani & Bahattab, 2015). In the first iteration, teams start building software using a limited subset of requirements, yielding software that encompasses only essential functionalities (Alshamrani & Bahattab, 2015; Boehm, 1986). It is then repeatedly subjected to this model's phases, evolving and gaining complexity, as the spiral progresses outwards. The spiral design of this model also provides an opportunity for requirements to be added and reviewed, and risk to be evaluated as each layer of complexity is planned (Alshamrani & Bahattab, 2015; Boehm, 1986).

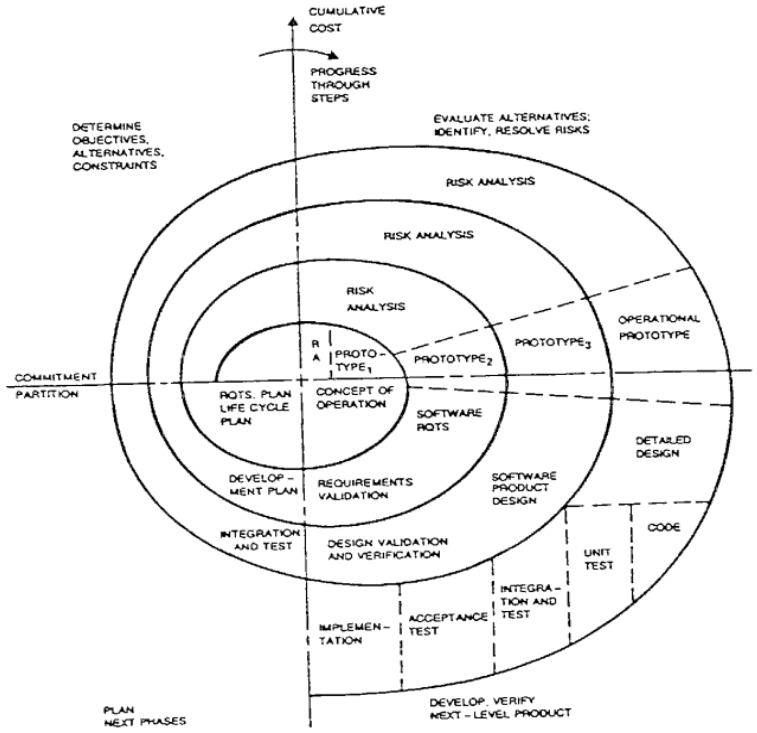


Figure 5 – Spiral Model (Boehm, 1986)

The evolutionary model encourages contact with the stakeholders after each iteration, providing room for customization of the system and additional functionality, depending on provided feedback. It also provides strong documentation as a consequence of its risk-centric characteristic (Alshamrani & Bahattab, 2015). However, a strong risk analysis can have negative outcomes, such as high costs and a significant amount of time devoted to risk analysis. Another disadvantage of this model is the need for high expertise of the involved teams, since the success of the project depends on a complete and accurate risk analysis (Alshamrani & Bahattab, 2015; Boehm, 1986).

The spiral model is recommended for medium to high-risk projects with long-term commitment, in which user needs need to be constantly reviewed (Alshamrani & Bahattab, 2015).

#### *3.1.2.4.* Agile

Agile software development is not a model in the traditional sense, but a development philosophy that inspired many other models (Gheorghe et al., 2020). It was created with the intention of changing many of the predefined assumptions of SDLCs that have bound software development since the inception of the waterfall model. These paradigm shifts can be summarized in four fundamental values (Fowler & Highsmith, 2001):

- **"Individuals and interactions over processes and tools"** – Involved teams and stakeholders drive software development, therefore, they should not be overshadowed by the processes and tools used (Fowler & Highsmith, 2001; Gheorghe et al., 2020).
- **"Working software over comprehensive documentation"** – Researchers involved in the origin of Agile noted that many developers had to expend too much time and effort in documenting their code, shifting focus from implementing requirements. Such documents are still important, but they are recognized as mostly unimportant to customers, as they value working software as a progress measure above anything else (Fowler & Highsmith, 2001; Gheorghe et al., 2020).
- **"Customer collaboration over contract negotiation"** – Frequent face-to-face collaboration between customers and project managers are central to the agile model, as it allows for both parties to reach consensus on needs and capabilities, creating a system that is more accurate to the customer's needs and grounding their expectations in reality (Fowler & Highsmith, 2001; Gheorghe et al., 2020).
- **"Responding to change over following a plan"** – In Agile, change is perceived as an opportunity to add value to a project and, therefore, is always welcome. Through its preference for building short iterations and its "release frequently" principle, it provides enough flexibility for additions and changes to requirements (Fowler & Highsmith, 2001; Gheorghe et al., 2020).

As previously stated, Agile is a philosophy for software development, providing principles for the creation of many other models such as SCRUM, Extreme Programming (XP) and Feature-Driven Development (FDD) (Gheorghe et al., 2020; S. Sharma et al., 2012). Consequently, the depiction of a single generalized phased Agile process is unfeasible. Despite this, the phase's definition and work follow familiar waterfall patterns, as presented in the "concepts" subsection, even if adapted to meet Agile principles and applied in a cyclical manner.

The overall disadvantages of Agile models are linked to some of their greatest strengths: if communication is not effective, despite constant interaction with the customer, the project can deviate from the intended outcomes, just as with any SDLC under these conditions (S. Sharma et al., 2012). Less documentation can be a barrier for newer developers joining the project at latter stages. Constant changes in requirement may also cause an increase in costs (S. Sharma et al., 2012). Agile models are effective in instances where a high level of uncertainty is exhibited and changes to requirements mid-project are not only possible, but likely. Its use is also advised in large and complicated projects where its division into small functional increments is viable and stakeholders agree to regularly engage in the project (Adesina et al., 2020).

#### *3.1.2.5.* RAD

Rapid Applications Development, exemplified in Figure 6, is a model focused on rapid cost-efficient building of functional software, that is, at the same time, able to retain high levels of quality (Beynon-Davies et al., 1999). A project based on this model provides the customer entity with functional software equivalent to a subset of the product in a relatively short timeframe (Berger & Beynon-Davies, 2009). To achieve this goal, this model employes a parallel development of prototypes by distinct teams. Such functional prototypes are later delivered to customers, to check whether the requirements were understood, followed, and fully meet the intended objectives. New functional increments will continue to be developed iteratively and delivered to users based on feedback. Once a prototype is accepted, it will be refined into a complete solution to be delivered to the customer (Berger & Beynon-Davies, 2009).

Through the previous explanation, one might notice the similarities between this model and Agile. The primary differences between the two is that while Agile is a philosophy, RAD is a proper model that does not share some of Agile principles. Some of the Agile characteristics not shared by RAD are face-to-face communication, projects being built around motivated individuals, simplicity, etc. (Berger & Beynon-Davies, 2009; Fowler & Highsmith, 2001).

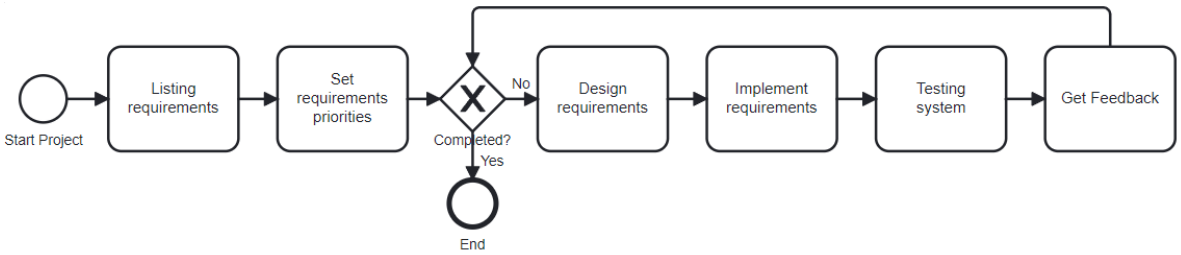


Figure 6 – Rapid Application Development Model adapted from (Daud et al., 2010)

In addition to the main advantages of this model already explained, there are other benefits to its use. Its cyclical and iterative design allows requirements to be reviewed in later stages of the development process. The regular delivery of working prototypes with low development costs also contribute to a remarkably high level of customer satisfaction (Daud et al., 2010). Modularized functions also facilitate development (Adesina et al., 2020). The RAD model also has some problems in its design. One of these disadvantages is the lack of scalability, meaning that an increase in requirements, scope and complexity of the project is unfeasible for this model. It also relies heavily on effective synergetic teams and experienced professionals with solid modeling skills (Adesina et al., 2020).

RAD is especially efficient in small and medium-sized projects with low complexity and a high degree of commitment from both developers and customers. A project using this model must also be of relatively low complexity and with stable, well-known requirements (Berger & Beynon-Davies, 2009).

#### *3.1.2.6.* Unified Process

The Unified Process, as shown in Figure 7, originates from the contemporary desire for bigger and more sophisticated software. Jacobson et al. (1999) proposed the need for a model capable of facilitating the management of all strands of these projects. Consequently, they advocate that their proposed model be used in conjunction with the UML, outlining three distinctive features central to its functionality (Jacobson et al., 1999):

- **Use-case driven** – A use-case refers to all individuals and systems interacting with the software. In this context, this means the SDLC needs to unravel how each individual use-case interacts with the system when identifying the project's functional requirements (Jacobson et al., 1999).
- **Architecture-centric** – Architecture is highly influenced by the defined requirements; however, software is also influenced by a multitude of other factors. To address this, the model calls for a parallel and balanced evolution of use cases and software architecture, allowing use cases to fit the architecture. The architecture, in turn, must encourage the discovery of new, previously undetected use cases (Jacobson et al., 1999).
- **Iterative and Incremental** – This model divides the work into increments, which result in an iteration when completed, that will be tested by developers. If it meets predefined goals, then they proceed to the development of the next iteration (Jacobson et al., 1999).

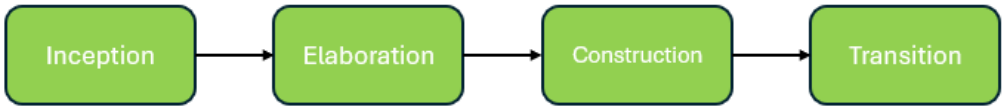


Figure 7 – Unified Process phases adapted from (Jagli & Temkar, 2013)

This model's phases are different than the ones presented at the beginning of this section. Its phases are: the Inception phase, Elaboration phase, Construction phase, and Transition phase.

The Inception phase consists of a preliminary system planning phase that attempts to grasp a vision for the final product. A discussion proceeds in an attempt to forecast the goals, risks, and use cases of the system, while a provisional architecture is built as a blueprint, containing the most critical subsystems (Jacobson et al., 1999; Pressman, 2010).

The Elaboration phase investigates most use cases, describing them in detail. This information is used to create a primitive architecture, both to assert a relationship between all increments of the system, allowing management to allocate resources, and to validate the executability of the model (Jacobson et al., 1999; Pressman, 2010).

In the Construction phase, the product is built and tested, resulting in an iteration ready to be delivered to users (Jacobson et al., 1999; Pressman, 2010).

Finally, in the Transition phase, the product moves to the beta release, in which it will be further tested, errors corrected, and training will be provided on how to use the software. This SDLC is cyclical, so this process restarts with the intention of building new iterations (Jacobson et al., 1999; Pressman, 2010).

## 3.2. ARTIFICIAL INTELLIGENCE

### 3.2.1. Concepts

Artificial intelligence, as explained previously, is a broad field of study that seeks to create intelligent machines that emulate, or even surpass, human intelligent behavior (Acemoglu & Restrepo, 2018b). According to Pannu (2015), "Artificial intelligence is the study and developments of intelligent machines and software that can reason, learn, gather knowledge, communicate, manipulate and perceive the objects". For another author, Wang (2019), artificial Intelligence differs from traditional computational software, as the second is designed to carry out actions according to predefined orders and criteria, while the former is generated through techniques that allow the creation of intelligent machines. In both definitions, the notion of "intelligent machines" is presented, which, in this context, refers to the system's ability to "adapt to its environment while operating with insufficient knowledge and resources" (Wang, 2019). For Wang (2019), adaptability is one the central characteristics that a software needs to demonstrate to be considered "intelligent", as exemplified by algorithms whose output depends on past experience, evidenced in machine learning.

All existing Artificial Intelligence algorithms fit the definition of Weak AI, also referred to as Artificial Narrow Intelligence, including, naturally, all AI presented in this study (*What Is Strong AI?*, 2021). Weak AI algorithms focus on performing a single specific or narrow set of tasks, requiring human intervention to provide input, train them, and set the parameters of their learning algorithm (*What Is Strong AI?*, 2021). Weak AI's definition exists in opposition to Strong AI, or Artificial General Intelligence, which describes a more advanced form of AI that can produce a self-aware and conscious mind indistinguishable from that of a human, having the ability to eventually solve problems, learn and plan for the future, independently of most human intervention. Such an AI system is a theoretical concept, not existing any available example of strong AI produced as of yet (*What Is Strong AI?*, 2021). Strong AI was considered a high-risk AI system and therefore restricted by the new "Artificial Intelligence Act" passed in the European Union (*Artificial Intelligence Act*, 2023).

### 3.2.2. Areas and pathways

As previously demonstrated, continuous developments in artificial intelligence, since its inception, have allowed its application to be a positive outcome in most fields, contributing to the emergence of many areas within its scope that provide broader functionality in their applications (Yarlagadda, 2017). According to Pannu (2015), the most notable areas of study in the field of AI are defined as follows:

- **Language Understanding** – Systems that have the ability to comprehend, interpret, and respond to natural language inputs. This does not simply include systems that are able of accepting natural language inputs, but also systems that are able of providing an output written in natural language from non-natural language inputs (Pannu, 2015).
- **Learning and adaptive systems** – Systems with this capacity are capable of dynamically adjusting their conduct, as well as formulating rules about a given environment, according to past experience (Pannu, 2015).
- **Problem solving** – The ability of a system to articulate a problem, develop a strategy for its solution, and detect areas where new information will be required, as well as understand where that information can be acquired. It includes inference, interactive problem solving, automatic problem solving, and heuristic search (Pannu, 2015).
- **Perception** – Systems with this attribute have the ability to analyze a given environment and represent it, as well as entities within the environment, in models. The relationships between environmental entities are structurally represented in these models. Perception, in this context, is related to AI pattern recognition and scene analysis (Pannu, 2015).
- **Modeling** – In this area, commonly used AI is defined as the method of building an internal representation, together with a set of transformation rules, that facilitates the prediction of behaviors and relationships present in data. It includes perceptual and functional representations of data, representation of models for problem solving, and modeling of natural systems (Pannu, 2015).
- **Robotics** – It consists of software able of interacting and manipulating the movement of physical objects through a combination of AI areas and capabilities previously presented (Pannu, 2015).
- **Games** – This area consists of formulating rules for games, later converting them into a representation or structure that enables the application of learning and problem-solving techniques, aiming to achieve the desired performance outcomes (Pannu, 2015).

### 3.2.3. Machine Learning

Advancements in machine learning, a subfield of AI, correlate with the recent explosion in AI capabilities (Janiesch et al., 2021), and are contemporaneously integrated into most AI work, including many of the technologies presented in this chapter (N. Sharma et al., 2021). Machine Learning builds self-learning models and algorithms capable of detecting intricate patterns in data. Such algorithms are created through the use of artificial neural networks, which use layered perceptrons (i.e., artificial neurons) to emulate the functioning of the human brain (N. Sharma et al., 2021).

One of the most notable subfields of machine learning is deep learning. This subfield focuses on building ML models through the use of deep neural networks, that correspond to neural networks with more than one hidden layer (Janiesch et al., 2021). Deep learning is very scalable, and its use is recommended when high-dimensional data is present (N. Sharma et al., 2021).

ML can be divided into the following three categories:

- **Supervised Learning** – In this category of machine learning, input data is labelled, meaning that for each input data, the target value/ output is provided in model training. In this way, it allows the calibration of the model, making it viable to predict target values using unseen data (i.e., data not used in model training) (Janiesch et al., 2021).
- **Unsupervised Learning** – What distinguishes supervised learning from unsupervised learning is that the latter does not use labeled data. Instead, it only includes input features in model training, making it ideal for detecting structural information present in data (Janiesch et al., 2021).
- **Reinforcement Learning** – Instead of training a model using test labels, it describes the current state of an environment without any prior information, establishes a goal, allowed actions and constraints (Janiesch et al., 2021; N. Sharma et al., 2021). Subsequently, the ML model experiences the environment by executing these actions, receiving feedback. This feedback updates and optimizes the strategy used to achieve the predefined goal. In short, reinforcement learning applies the principle of trial-and-error in building ML models (N. Sharma et al., 2021).

### 3.2.4. AI algorithms

As previously established, AI has the potential to detect patterns in data, predict outcomes and assist in decision-making. However, there are a plethora of AI algorithms available to perform a given task. Therefore, a brief study of them, focusing on the advantages and disadvantages of each one, is considered essential to generate the proposed solution:

- **Artificial Neural Networks (ANN)** – Set of interconnected nodes that simulate neural structures, learning through adjusting the weights of these neurons (Brar & Nandal, 2022). ANN is a robust AI tool, capable of dealing with non-linear, noisy, incomplete, and high-dimensional data, therefore presenting robust and adaptive characteristics (M. Mijwil, 2021). ANN also allows parallel computing, which, although hardware dependent, allows the execution of multiple computational tasks simultaneously (M. Mijwil, 2021). The limitations of ANN are its black box nature (hard to understand the internal workings and calculations that return a certain output). It is also computationally intensive, prone to overfitting, and its models are difficult to adjust (M. Mijwil, 2021).
- **Support Vector Machine (SVM)** – A supervised algorithm capable of performing linear and non-linear regression, as well as performing classification into two or more categories (Quba et al., 2021). SVM achieves accurate results, is efficient with high-dimensional data, especially in text classification (Boateng et al., 2020), and its outputs are not heavily influenced by outliers (Boateng et al., 2020). However, this algorithm exhibits lengthy training times, and its use is not recommended for handling high volumes of training data (Boateng et al., 2020).
- **K-Nearest Neighbors (KNN)** – In this algorithm, prediction is performed by assigning objects to the same class of closest and most common training examples, with K meaning the number of nearest training examples to be considered (Boateng et al., 2020). In regression, it considers the distance to the nearest neighbors and calculates their average (Quba et al., 2021). The strength of this algorithm is that it is simple, versatile, easy to implement, robust to noisy training data, effective for large quantities of training data, and as K increases, so does its accuracy (Boateng et al., 2020). However, its computation time can be time-consuming, especially as its K and training data volume increases (Boateng et al., 2020).
- **Naïve Bayes (NB)** – Naïve Bayes is a simple algorithm that assigns instances to a class through probabilistic calculations (Brar & Nandal, 2022). It demonstrates high training speed, it is insensitive to noisy features, and requires a small amount of training data, while having the ability of handling multiple classes (Kalcheva et al., 2020). This algorithm assumes the independence of each feature and that they offer an equal contribution to the outcome, a condition that is not met most of the time, which can cause outputs to not be optimal, and is sensitive to how input data is prepared (Kalcheva et al., 2020).
- **Decision Trees (DT)** – This algorithm builds tree-like structures that rank attributes through feature values (Brar & Nandal, 2022). This is an intuitive algorithm that benefits from easy internal visualization, speed, scalability, it is computationally cheap, and missing or erroneous values and irrelevant features do not offer a considerable impact on the result (Kalcheva et al., 2020). However, this algorithm is prone to overfitting, sensitive to changes in data, and needs large volumes of training data to obtain accurate results (Kalcheva et al., 2020).

- **Random Forests (RF)** – Random Forests is an ensemble of many generated decision trees grown through randomization, and the outcome is predicted through majority voting for classification tasks or an average of the prediction values for regression tasks (Boateng et al., 2020). It is also characterized by its good performance, being versatile, user-friendly, resistant to overfitting, with a low outlier impact and able to deal with large datasets (Boateng et al., 2020). Its disadvantages is that it is time consuming, especially considering that the number of trees grow with the number of features (Boateng et al., 2020).
- **Logistic Regression (Log R.)** – This supervised learning algorithm is used for binary classification or predictive tasks. This method is easy to train and very efficient when dealing with linear data. This algorithm is, however, inefficient when dealing with non-linear data and highly correlated features (Burns et al., 2022).
- **Linear Regression (Lin R.)** – This supervised learning algorithm can be used for predictive tasks. The advantages of Linear Regression are its interpretability, ease of implementation and understanding of its outputs. It is also computationally efficient, quick to train, and robust to outliers, having a less impact on model performance. The limitations are the assumption of a linear relationship between dependent and independent variables, that might impact performance if this is not the case in reality, a susceptibility to overfitting, and a sensitivity to multicollinearity (*Linear Regression in Machine Learning - GeeksforGeeks*, n.d.).
- **Adaptive Boosting (AdaBoost)** – AdaBoost is an ensemble method that combines multiple weak learners into a stronger learner, weighing each instance to provide a solution. The advantages of AdaBoost are its low generalization error, its flexibility as it is easily modifiable and can be combined with other algorithms, it is easy to code, computationally efficient and applicable to complex tasks (Kalcheva et al., 2020). The disadvantages of this algorithm are that it is prone to overfitting, the presence of considerable noise during training, and requiring large training samples (Kalcheva et al., 2020).
- **Convolutional Neural Networks (CNN)** – CNN is a DL neural network used to solve tasks related to images, video, and NLP (Abdel-Jaber et al., 2022). Models built through this algorithm are easy to train, scalable due to parallelization, practical in their implementation, efficient in feature extraction and have an acceptable performance (Abdel-Jaber et al., 2022). The disadvantage of this model is that it requires a lot of memory to store intermediate results, being therefore hardware dependent, often confusing images, and is of no use if data is completely unstructured (Abdel-Jaber et al., 2022).
- **Recurrent Neural Networks (RNN)** – RNN is also a DL neural network that is characterized by having an internal memory and is recommended for tasks that involve text, speech or video (Abdel-Jaber et al., 2022). It is capable of processing inputs of any length, and the model size does not increase due to increasing inputs (Abdel-Jaber et al., 2022). However, it demonstrates slow computation times, high training difficulty, limited scalability, and issues with exploding and vanishing gradients (Abdel-Jaber et al., 2022).
- **Long short-Term Memory (LSTM)** – This RNN algorithm is useful to capture long-term dependencies and store them over long periods, that is useful for capturing context between events when such conditions are met. Additionally, it deals with RNN's vanishing and exploding gradients issues. The main issues of this algorithm are scalability, it requires high numbers of training data, training is time-consuming, it is computationally expensive, and it is hard to parallelize ("Deep Learning | Introduction to Long Short Term Memory," 2019).

## 3.3. Systematic Literature Review on Artificial Intelligence in Software Development

### 3.3.1. PRISMA Methodology

In this subsection, the PRISMA statement is used for the purpose of carrying out a Systematic Literature Review, that is, a planned literature review performed through the analysis of the information collected, using predefined criteria. This search criteria must be applied to the appropriate search engines with the aim of filtering the most relevant articles and further contributing to a higher quality solution (Sarkis-Onofre et al., 2021). To this end, the PRISMA Statement provides recommendations, through a four-phased flowchart, with the aim of selecting articles considered valuable for research, along with a 27-item checklist to guide researchers in its application (Sarkis-Onofre et al., 2021). This, together with the previously detailed outline of all research strategies employed, facilitates the understanding of the research process, providing transparency to a study, being of interest to both the researcher and potential readers (Sarkis-Onofre et al., 2021).

Furthermore, although its application is primarily intended for systematic reviews to perform meta-analyses in the biomedical industry, the PRISMA statement is widely accepted as a reliable mechanism for systematic literature reviews in most circumstances (Sarkis-Onofre et al., 2021).

Firstly, before carrying out the research itself, it is necessary to state the objectives that such queries intend to achieve and whose response provides a better understanding of the interaction between both topics before a solution is envisioned. Thus, the questions that must be asserted through an overlook of the previous subsections, serving as a basis for the research, are:

1. What is the current status of research?
2. What are the benefits and issues with the application of AI in this area?
3. What AI techniques are currently useful in Software Development?

The selection of appropriate keywords, present in Table 2, is also done through the analysis of the previous subsections of the literature review. Please note that all chosen keywords entered into search queries are in English, as only papers in English are to be accepted for this study. Furthermore, the purpose of the “additional keywords” column is to reduce returned articles in favor of the ones most compatible with the defined questions.

Table 2 – Keywords

| *Artificial Intelligence* | *Software Development Process* | *Additional Keywords* |
|---|---|---|
| Artificial Intelligence | Software Development | Benefits |
| Artificial Neural Networks | Software Development Model | Challenges |
| Machine Learning | Software Development Lifecycle | Techniques |
| | | Current Status |

To further improve the quality of the proposed solution, the retrieved articles are restricted, using additional criteria, as shown in Table 3. Thus, said criteria dictates the characteristics the returned articles must obligatorily possess, being excluded if an article is perceived to be in direct opposition to it, if they are duplicated, or if the titles, abstract or content fall outside of this paper's scope.

Table 3 – Article Inclusion Criteria

| *Criteria nº* | Criteria Explanation |
|---|---|
| *Criteria 1* | Articles must address AI utilization in Software development process. |
| *Criteria 2* | Articles must be published between 2020 and 2024. |
| *Criteria 3* | Articles must be written in English. |
| *Criteria 4* | Articles must be peer reviewed academic papers. |

The criteria presented above in Table 3 are considered the most critical for this work. Articles written in other languages other than English should be excluded for transparency purposes, as English is considered a global language, allowing readers an easier access to the knowledge collected in this systematic literature review. Accepted articles will also be required to be peer reviewed to ensure a high-level of credibility and quality of the information used. Furthermore, to further enhance the value of the information collected, only articles within the mentioned timeframe should be considered, as a fast evolution of knowledge within the scope of the topics studied can make some knowledge outdated or even obsolete. These criteria must be considered when filtering articles, as outlined in the PRISMA flowchart.

The search query is carried out in the reputable study databases shown in Table 5. The following string is entered into them: ("Software Development" OR "Software Development Lifecycles" OR "Software Development Models") AND ("Artificial Intelligence" OR "Machine Learning" OR "Artificial Neural Networks") AND ("Current status" OR "Benefits" OR "Challenges" OR "techniques").

Table 4 – Searched Databases

| *Article Database* | *Database's URL* |
|---|---|
| Scopus | https://www.scopus.com/home.uri |
| IEEE Xplore | https://ieeexplore.ieee.org/Xplore/home.jsp |
| Web of Science | https://www.webofknowledge.com/ |

Searches through the PRISMA Statement are performed according to the flowchart provided and using the 27 item checklists as guidelines. The aforementioned flowchart is divided into four cascading phases that progressively filter the studies found, as visually demonstrated in Figure 8.

The identification phase corresponds to the search, according to the previously specified sequence and criteria, returning 2042 studies in total. Then, in the screening phase, duplicated articles are identified and removed, resulting in 686 articles being discarded from the total, and in a new total of 1716. Still, within the same phase, the titles of the articles are overlooked to identify those perceived as not exploring the interaction between both topics and not following predefined criteria. Through a screen of the articles, more specifically of their titles, a total of 724 studies were identified that did not follow the criteria, giving a new total of 992 articles to be filtered through the eligibility phase.

The eligibility phase aims to exclude articles whose information does not offer value in answering previously formulated questions and in building the solution. To this end, the abstracts of these articles were first analyzed, deducting 816 articles, returning a new total of 176 studies. Then, the content of these articles was examined for the same purpose, leaving a total of 37 approved articles.

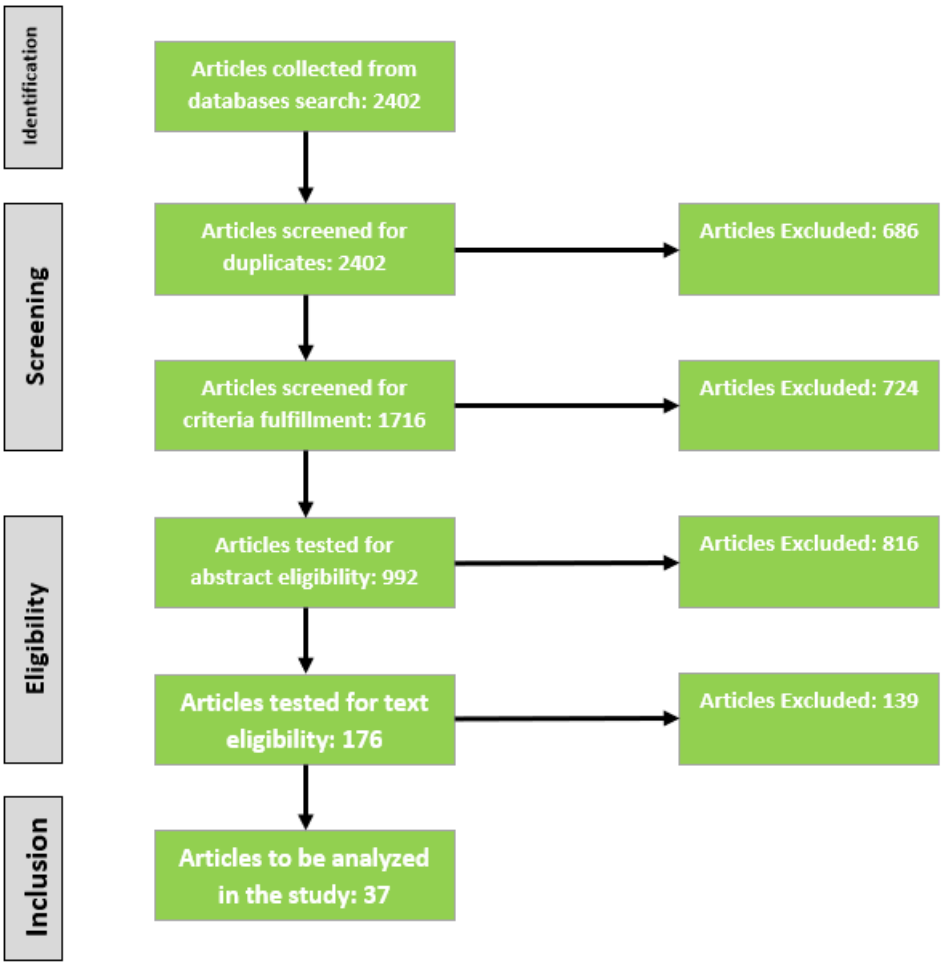


Figure 8 – PRISMA Statement Process

After applying the PRISMA methods described previously, 37 were considered to be relevant to the study, that are listed in Table 5. Therefore, these studies are further analyzed to answer the questions of the systematic literature reviews.

Table 5 – Articles collected through PRISMA

| Nº | Author | Title | Type |
|---|---|---|---|
| **[1]** | (Shafiq et al., 2021) | A Literature Review of Using Machine Learning in Software Development Life Cycle Stages | Journal Article |
| **[2]** | (Ramirez et al., 2019) | A Systematic Review of Interaction in Search-Based Software Engineering | Journal Article |
| **[3]** | (Dwivedi et al., 2021) | Artificial Intelligence (AI): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy | Journal Article |
| **[4]** | (Korzeniowski & Goczyła, 2019) | Artificial intelligence for software development — the present and the challenges for the future | Journal Article |
| **[5]** | (Shankar & Chaudhari, 2023) | Framework for the Automation of SDLC Phases using Artificial Intelligence and Machine Learning Techniques | Journal Article |
| **[6]** | (Navaei & Tabrizi, 2023) | Impact of Machine Learning on Software Development Life Cycle | Conference Proceedings |
| **[7]** | (Chuanjian et al., 2023) | Research on Software Development Based on Low-Code Technology | Conference Proceedings |
| **[8]** | (Sofian et al., 2022) | Systematic Mapping: Artificial Intelligence Techniques in Software Engineering | Journal Article |
| **[9]** | (Kumar et al., 2023) | The Current State of Software Engineering Employing Methods Derived from Artificial Intelligence and Outstanding Challenges | Conference Proceedings |
| **[10]** | (Moroz et al., 2022) | The Potential of Artificial Intelligence as a Method of Software Developer's Productivity Improvement | Conference Proceedings |
| **[11]** | (Sampada et al., 2020) | A Review on Advanced Techniques of Requirement Elicitation and Specification in Software Development Stages | Conference Proceedings |
| **[12]** | (Liu et al., 2022) | Artificial Intelligence in Software Requirements Engineering: State-of-the-Art | Journal Article |
| **[13]** | (Naseem et al., 2021) | Empirical Assessment of Machine Learning Techniques for Software Requirements Risk Prediction | Conference Proceedings |

| Nº | Author | Title | Type |
|---|---|---|---|
| **[14]** | (Sainani et al., 2020) | Extracting and Classifying Requirements from Software Engineering Contracts | Conference Proceedings |
| **[15]** | (Imtiaz et al., 2021) | Predicting Vulnerability for Requirements | Conference Proceedings |
| **[16]** | (Quba et al., 2021) | Software Requirements Classification using Machine Learning algorithm's | Conference Proceedings |
| **[17]** | (Brar & Nandal, 2022) | A Systematic Literature Review of Machine Learning Techniques for Software Effort Estimation Models | Conference Proceedings |
| **[18]** | (Iftikhar et al., 2020) | Artificial Intelligence Based Risk Management in Global Software Development: A Proposed Architecture to Reduce Risk by Using Time, Budget and Resources Constraints | Journal Article |
| **[19]** | (Al Asheeri & Hammad, 2019) | Machine Learning Models for Software Cost Estimation | Conference Proceedings |
| **[20]** | (Darandale & Mehta, 2022) | Risk Assessment and Management using Machine Learning Approaches | Conference Proceedings |
| **[21]** | (Aychew & Alemneh, 2022) | Selection of Architectural Patterns based on Tactics | Conference Proceedings |
| **[22]** | (Singh & Kumar, 2023) | Software Cost Estimation: A Literature Review and Current Trends | Conference Proceedings |
| **[23]** | (Bassi & Singh, 2023) | A Systematic Literature Review on Software Vulnerability Prediction Models | Journal Article |
| **[24]** | (Dehaerne et al., 2022) | Code Generation Using Machine Learning: A Systematic Review | Journal Article |
| **[25]** | (Dewangan et al., 2022) | Code Smell Detection Using Ensemble Machine Learning Algorithms | Journal Article |

| Nº | Author | Title | Type |
|---|---|---|---|
| **[26]** | (Menshawy et al., 2021) | Code Smells and Detection Techniques: A Survey | Conference Proceedings |
| **[27]** | (Malhotra et al., 2023) | Examining deep learning's capability to spot code smells: a systematic literature review | Journal Article |
| **[28]** | (Ricca et al., 2021) | AI-based Test Automation: A Grey Literature Analysis | Conference Proceedings |
| **[29]** | (Shaji & R, 2023) | Analysis and study of IDSs for cybersecurity vulnerability detection and prevention: A Comprehensive Review | Conference Proceedings |
| **[30]** | (López-Martín, 2022) | Machine learning techniques for software testing effort prediction | Journal Article |
| **[31]** | (Samant et al., 2023) | Optimizing Issue Tracking Systems using Deep Learning-based Issue Classification | Conference Proceedings |
| **[32]** | (S. Sharma & Chande, 2023) | Optimizing test case prioritization using machine learning algorithms | Journal Article |
| **[33]** | (Batool & Khan, 2022) | Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review | Journal Article |
| **[34]** | (Worku et al., 2023) | Test Case Generation from Quality Attribute Scenarios Using Machine Learning Approach | Conference Proceedings |
| **[35]** | (Hourani et al., 2019) | The Impact of Artificial Intelligence on Software Testing | Conference Proceedings |
| **[36]** | (Saini & Britto, 2021) | Using Machine Intelligence to Prioritise Code Review Requests | Conference Proceedings |
| **[37]** | (Bocu et al., 2023) | An Extended Survey Concerning the Significance of Artificial Intelligence and Machine Learning Techniques for Bug Triage and Management | Journal Article |

### 3.3.2. PRISMA Results

#### 3.3.2.1. SLRQ 1 - What is the current status of research?

After a selection of articles, necessary to perceive the current status of research, as well as answer the other questions of the systematic literature review, it can be observed that the topic of artificial intelligence in software development has been a subject that has captured substantial interest by researchers. This is supported by [1], which recognizes the existence of an exponential increase in such literature, due to the "potential of such technologies".

During the analysis of the articles retrieved during its process, it was also observed that the application of AI technology has been studied thoroughly for all SDLC phases [1, 6, 8]. However, in a quantitative analysis, it was concluded that the testing phase was the focal point of research in this area [1, 8], while the integration of AI technologies in the maintenance phase was the least studied.

In terms of technologies, ML has been a subject of widespread investigation in this area [2]. The potential of ML to learn from historical and current data drives its predictive capabilities, providing information necessary to accelerate the development process and assist in decision-making activities, as well as for other uses to be outlined in more detail, is evidenced in [3, 6]. In fact, the use of ML to create predictive models for various applications in the SDLC has been one of the most researched areas for this particular technology [1, 6].

Regarding the current state of research in each phase of the SLDC, and starting with the requirements gathering and analysis phase, AI is used in requirements elicitation [11, 12, 14], to facilitate the process of deriving further meaning from information collected from stakeholders, whether through direct interaction between the contractor or by analyzing customer feedback, through reviews or social media [5]. Requirements prioritization is also another area where AI is used to ranking requirements from the most critical to the project to the least important [12]. AI can also be used to classify requirements into functional requirements and non-functional requirements, ambiguous and unambiguous, and other categories considered useful for the given case [6, 11]. In [13] and [15] the authors argue that most flaws within a system can be traced to the earliest phases of the development process, underlining the critical nature of earlier risk detection from the collected requirements. For this, a risk prediction model is presented in their work.

For the Architecture and Design phase, the information can be used to facilitate estimation of the effort required for the project, using software effort prediction models. These types of models, such as the one proposed in [17], [19] and [30], allow an easier estimation of time required to complete the project and its potential costs. In [14], a model is also proposed with the intention of predicting architectural patterns for the project, to facilitate modeling, showing the analytic potential of these technologies to assist even design activities.

The use of generative technologies is also studied in the literature. For coding, it has the potential to generate code through the use of natural language as input, that is called NL-to-Code generation, as explained in [24]. In the same article, 2 other categories are also demonstrated: Code-to-Code (which is mostly associated with autocorrection, autocompletion of a code string, or translation of code into other programming languages), or Code-to-NL (that is mostly used for the generation of documentation through the built code).

Code Smells (i.e., “any sign that possibly can negatively affect the software process”, affecting its maintainability, understandability, implementation, and quality [26]) have been identified in literature. These solutions are intended for the identification and consequential resolution of these code smells through techniques such as refactoring [25, 26, 27], that is considered a code-to-code usage [26]. As previously mentioned, the integration of AI technologies in the testing phase has been the most studied [6], and therefore, achieves competitive results.

In [34] and [35], the use of generative technologies for generating test cases is demonstrated, becoming an important subject in the software development industry. According to [38], the automated generation of test oracles is also tested in some of the collected literature, however, it was limited to the visual correction of the presentation layer, with more functional test oracles and their generation not being considered in the selected literature. The prioritization and classification of test cases is also studied in [32] and [35]. In [31], it is demonstrated that the identification and classification of issues is also possible. Requests for code review and their prioritization are studied in [36].

Software fault prediction has been the focus of researchers in the testing phase, leading to more maintainable software and a more time-efficient solution to any issue that might arise within the built software [33]. In [30], predictive techniques are to estimate software testing effort.

There has been an exponential increase in software vulnerabilities since 2016, reported by the National Institute of Standards and Technology (NIST), incurring considerable costs for companies annually [23]. In [29], AI is used to build Intrusion Detection Systems (IDS) that examine and alert possible intrusive activities, allowing a quick application of necessary corrective measures and, consequently, a safer system.

It is also important to acknowledge that a framework that attempts to achieve the same goal as this study was developed in [5]. This article presents the possible algorithms used for each phase of the SDLC and strongly advocates automation in most of the development process through chatbots.
Confidence in the potential of these technologies to reshape the software development process is emphasized in [9], which asserts that by the year 2050, notable changes would be apparent, through the emergence of new jobs, and responsibilities. Developers are expected to adapt to new skills and expertise sets, required to use AI as a complementary tool, rather than a standalone replacement for their work. Artificial Intelligence in this area mainly intends to be an auxiliary tool, helping to avoid more monotonous and repetitive software development activities [12]. On a further note, and according to [12], Artificial Intelligence in its current state does not replace the need for human participants in the development process, as human-led creative work cannot be replaced by AI [6]. The authors in [9] even argue that no one will be able to replace the role of a developer, except themselves, as they fail to adapt to changes in required skills and necessary knowledge sets.

#### 3.3.2.2. SLRQ 2 – What are the benefits and issues with the application of AI in this area?

Before creating the solution, it is necessary to acquire a better understanding of the strengths and challenges of using these technologies in software engineering. Such an analysis of the perceived advantages and difficulties will be conducted from the perspective of the relevance cycle, starting with the benefits.

Software engineering remains the predominant sector within the current industrial landscape, meaning that leveraging artificial intelligence can bring significant benefits to this area, as has happened in many other industries [6]. The confidence in these advantages is so great that the same article calls the automation of software development activities the most important task of the present.

The need for software is gradually increasing, as is the demand for more complex offerings, also increasing time completion of projects and their costs [8]. The inclusion of such technologies generally results in lower costs and risks, optimized solutions and an accurate completion timeframe of software projects, balancing the consequences in the aforementioned increase in demand [2, 6, 9] and allowing companies to remain competitive in the market.

An effective use of AI tools multiplies the creative capacity of human developers by performing more repetitive and monotonous activities [2], as well as providing additional information and patterns that enhance this creative thinking [9]. This, in turn, allows those who participate in the software development process to perform more creative tasks, achieving better results and increasing their performance [2, 8, 10].

Machine learning offers significant utility across the various processes of the SDLC by leveraging information from previous projects [8]. The capabilities of this technology provide timely data-driven business decisions and more thorough data analysis, capturing patterns that can easily escape the human eye [1, 6]. This pattern recognition capability provides critical information for the various phases of the SDLC.

The predictive characteristics of these technologies allow developers to extract more meaning from data. This is essential for planning activities, which allow an estimation of potential risks, as well as a more accurate estimate of both the necessary budget for a project and the timeframe for its completion [13]. This is essential, as they may mean the success or failure of the project. It is also important to build a better relationship between the developer and the customer, as only 16.2% of projects are delivered in time and only 42% have full functionality, which can harm relationships due to lack of trust and broken promises [13]. Furthermore, AI capabilities leverage a predictive approach to possible vulnerabilities within a system, instead of a more reactive approach, allowing to be estimated and consequently corrected [8].

According to [35], test generation has become an important issue in the software development industry to ensure that requirements are met as requested by stakeholders, as well as providing optimal and secure solutions. Manual testing is, however, a time-consuming and repetitive activity, which exacerbates its high costs. AI helps ensure the success of this phase and contribute to a superior return of investment.

Another benefit arising from the inclusion of AI for automation purposes is through the use of NLP technologies, which facilitate user interaction with the system. Chatbots can even provide personalized solutions according to their requests/ needs [5]

The use of artificial intelligence in software engineering also yields some challenges, both inherent to AI and due to the fact that this is a recent field of research in this area [2].

To begin this exposition of current challenges, it is essential to underline the wide availability of ways to solve a software engineering problem using AI and, although there are algorithms more compatible with a given case, choosing the right AI model and algorithm can prove to be difficult [1].

Another obstacle to the use of artificial intelligence is its stochastic nature, characterized by the inherent randomness and probabilistic behavior of its outcomes. This results in systems returning different outputs when inserting the same inputs to the model execution [1, 4, 10]. This is problematic as to build trustworthy solutions through AI, their outputs need to be reproducible and rigorous [1].

The lack of sufficiently labeled and structured datasets is also described as a barrier in building AI models. In a general sense, this can be overcome as many researchers are starting to share datasets publicly [1], yet this is more difficult in the corporate environment, as companies prize the privacy of their data necessarily for their solutions not to be copied due to competition.

Sometimes sub-optimal solutions can be returned through poorly made queries or poorly constructed models. This means that there is a need for humans with sufficient knowhow to regulate and approve such outputs for use [9].

Successful AI solutions and models can bring many advantages to both companies and SDLC participants, but they can make collaborators overly dependent on their decision support capabilities. This problem is called automation bias, and reduces collaborators information seeking vigilance, which can be exacerbated if the AI system fails [2, 3].

In [3], it is stated that AI-based systems tend to function like a black box, lacking transparency. In other words, AI models often return solutions without an explicit justification, which can make outputs hard to interpret. This can lead to misinformation and a lack of transparency.

As generative technologies are built using source code, the generated solutions may use legacy code, which can affect maintainability and even increase the vulnerability of systems [10].

In some classification problems, the boundaries between different classes might be blurred, even for human actors, opening the possibility of misclassifications in these areas. This is more prevalent in cases where image or voice samples are used [4].

The transition to the use of AI solutions also presents several organizational and managerial challenges that must be taken into consideration. Resistance to change and high investment in training are expected in these cases [3].

Finally, problem solving by providing solutions generated from learnt data is within the capabilities of AI, as previously explained. However, AI reaches its limits when attempting to solve unique problems, design novel routines, and uncover new problem sets [9].

#### 3.3.2.3. SLRQ 3 – What AI techniques are currently useful in Software Development?

After analyzing the acquired articles in the literature review, it can be concluded that there is extensive research on artificial intelligence algorithms and their application in different software development tasks. Most of these algorithms fall into the category of machine learning, as explained in the "current status" SLRQ, which is becoming one of the most important technologies in this area [6]. The research on techniques focuses on the identification of algorithms compatible with a given activity, with even some performing comparative studies between these algorithms to identify the most efficient one.

In the task of collecting requirements and their subsequent categorization, classification algorithms are used: RF, NB, Log R., SVM, DT, KNN, AdaBoost [5, 11, 12]. These ML algorithms offer the ability to discern requirements from non-requirements by classifying them as such, and then further delineate the collected requirements into distinct classes, such as functional and non-functional requirements. Furthermore, NLP techniques should also be leveraged for their proficiency in textual analysis and to facilitate interactions between users and the technology [12], especially through building chatbots for elicitation purposes [5, 12]. Deep Learning, more specifically, ANN and CNN, has also shown promise for the categorization of requirements [1, 16], and for finding defects in requirements that need further clarification [12]. Some presented models combined classification algorithms and TF-IDF or Bag of Words to leverage NLP with successful outcomes [11, 16]. Among these approaches, respectively, RF and SVM presented the best results.

Other uses of classification algorithms within requirements elicitation are requirements prioritization [12], validation [12] and requirements change requests [11]. In [14], a model for collecting requirements from contracts is also studied with BiLSTM (a DL algorithm) for requirements elicitation and BERT, an NLP model, for classification into various categories. For requirement prioritization, KNN and J48 were both efficient, respectively, in [11] and [12].

Risk prediction through collected requirements is studied in [13], employing several ML algorithms: KNN, Average One Dependency Estimator (A1DE), Composite Hypercube on Iterated Random Projection (CHIRP), Decision Table/ Naïve Bayes Hybrid Classifier, Credal Decision Tree (CDT), CS-Forest, DT, J48, NB, and RF. In it, CDT, DT and DTNB provided the best results.

Vulnerability prediction through requirements is proposed in [15], where a voting-based ensemble of RF, KNN, DT, and Log R provided the highest efficiency. To efficiently perform this task, it is suggested that a hard ensemble followed by a soft ensemble of ML algorithms be employed in the prediction module [15]. In the same article, SVM, ANN and NB were also found to be effective for this task.

In the context of planning and design in software engineering, AI can be used to suggest functions, modules and interfaces through the application of various ML algorithms, including NLP, DT, MNB, RNN, Log R., ANN, RF, and Genetic Algorithms [5]. One such instance is exemplified in [21], where a model for recommending architectural patterns, using tactics as a selection feature, is researched. The presented model used NLP for data preprocessing, and then a combination of either TF-IDF or word2vec was used as feature extraction techniques. These were applied together with DT, SVM or NB to build the model, and TF-IDF and SVM showed the best results.

For effort estimation in [17], the following ML algorithms are used: SVM, RF, DT, NB. Lin R., K-Star, Case-Based Reasoning, and ANN. In the same article, it is stated that ANN is the most used algorithm for this task in the literature, despite this, neither ANN nor other algorithms can be treated as the best, as each one has its strengths and weaknesses.

Cost estimation is explored in [19], through the application of numerous algorithms and their subsequent tests, through various metrics, with RF and REPTrees showing the best results. In [22] a similar study was carried out, comparing the following models and algorithms: COCOMO, NN, RF, GA Fuzzy Logic, SVM, ANN, DT. In the same study, it was found that although learning-oriented methods exhibit better results, no method can be considered superior.

For the assessment of risks, [20] introduces a model trained on recorded data to detect risk factors within a project. To this end, the following classifiers were compared: ANN, KNN, NB and DT, with NB yielding the best results for the test dataset employed.

The implementation and coding phase may benefit from NLP technologies to facilitate tasks. The use of chatbots is advised in [5] with the intention of providing tips, links or contacts to address problems when encountered, or even generate code. The suggested algorithms for this purpose are CNN, RNN, ANN, DT, NB, SVM, Log R., J48, AdaBoost, Extreme Gradient Boosting [5].

Generative technologies in this phase are explored in [24], which divided this process into three categories, as explained in SLRQ 1: the first is NL-to-code, where RNN and transformer models are predominantly utilized. For GUI code generation, CNN shows effectiveness in extracting high-level visual features from images, from which code can be built through algorithms like LSTM. To code for NL, numerous studies reviewed in [24] used LSTM encoder-decoders, CNN and Abstract Syntax Tree data structures. Lastly, for code-to-code, algorithms such as KNN, LSTM, RF, ANN were employed in various models, as shown in literature review of the article.

Regarding code smell detection, a range of ML algorithms were identified in [26] for their detection capabilities: RF, ANN, DT, SVM, DL. The results of [27] also highlighted the effectiveness of hybrid models such as CNN-LSTM, as well as the utility of RNN in this domain. Furthermore, [25] tested with ensemble and DL methods to detect code smells in Java. Gradient Boosting, XGBoost, AdaBoost and Max Voting were the best algorithms for the detection of data class, god class, feature envy, and long method code smells, with Max Voting proving to be the most efficient overall.

To predict vulnerabilities within built software, researchers have explored a wide variety of ML and DL approaches. Among them, the most popular algorithms were NB, RF, SVM, DT, Log R., as these presented the best performance [23].

In the testing phase, as previously mentioned, a wide variety of techniques were investigated. Test case generation, mainly through NLP, is one of the most popular types of research at this phase [28]. In [34], a model for test case generation is proposed through quality attribute scenarios (i.e. quality attributes, such as performance, modifiability, and security). In the proposed model, feature extraction is performed using either TF-IDF and word2vec, each combined with one of the following ML algorithms: SVM, Multinomial NB, DT, RF, AdaBoost and Gradient Boosting Machine. Results show that TF-IDF with DT outperforms all other models, with a combination of TF-IDF and SVM being a close second [34]. In [35], genetic algorithms for generating test cases are also suggested, as well as Hybrid

Genetic Algorithms for generating GUI test cases. Regarding test oracle generation, it is concluded in [28] that it is mainly used for GUI testing and is limited to visual correctness of the presentation layer.

Test case prioritization is stated to play a crucial role, especially when considering the increase in required number of test cases arising from the increasing complexity of systems. In [32], several algorithms are suggested for this purpose: ANN, DT, RF. Comparatively, ANN and DT show a tradeoff between efficiency and speed, with ANN being the most efficient and the slowest, and DT being the fastest and least efficient. RF shows a balance between efficiency and speed.

Software fault prediction is studied in [33] through a comparative examination of various machine learning, deep learning, and data mining algorithms. The algorithms that proved to be most efficient for machine learning were SVM and for data mining, SVM and DT. Regarding DL, CNN was shown to provide the best results, but multilayer perceptrons are also reliable and are more computationally efficient.

For vulnerability detection, DT, SVM and RF are suggested for classifying vulnerable and non-vulnerable code [5].

In [37], a survey on articles that study bug triaging is conducted. In this article it is stated that ML models are a natural solution to implement an automatic bug triage system, being frequently used and simple to model. ML algorithms used for this activity are SVM, KNN, J48, RF, ANN. DL characteristically demonstrates consistent computational performance, scalability and learning rates, however, it requires greater training time. The DL algorithms used for this task, according to the article, are CNN and RNN.

In [31], text classification for tracking and management problems and bugs is studied. To this end, it proposes the extraction of features from textual data through TF-IDF and then apply NN algorithms. Similar work mentioned in the article concluded that SVM was faster and had better performance. Decision trees are more efficient in hierarchical classification, and KNN should be applied when K is known. DL algorithms (CNN and RNN) return better accuracy in multiclass document classification. In [36], a model for prioritizing code review requests is built using Bayesian Networks.

Software maintenance can harness AI to provide greater system security by building intrusion detection systems. In [29], models used to examine raw network packet data and intrusive activities are shown in its literature review. These approaches employ ML algorithms (ANN, NB, SVM, clustering) and DL algorithms (LSTM and CNN) to assist in accomplishing this cybersecurity task. Although ML algorithms are the most popular, DL is advantageous for computer vision applications, but presents adaptability problems with high-dimensional nonlinear data. DL is also useful for validating users, packages and flows [29].

# 4. FRAMEWORK TO LEVERAGE ARTIFICIAL INTELLIGENCE IN SOFTWARE DEVELOPMENT

## 4.1. ASSUMPTIONS

Through the literature review and the subsequent analysis of the acquired information, it was possible to draw some essential conclusions for generating the framework.

In the previous chapter, a study detailing the different methodologies that guide software development was realized. In this chapter, it was elucidated that there is no single universal SDLC model favorable for all cases, each one having different characteristics that must be considered when building the framework. The literature review allowed the acknowledgement of many of these SDLC models together with their underlying characteristics, as shown in Table 6:

Table 6 – SDLC characteristics

| SDLC models | Characteristics |
|---|---|
| **Waterfall** | The waterfall model depends on thorough documentation for its success, as well as for its requirements to be testable. Therefore, it could benefit from *interpretable* algorithms.<br>As it is inefficient in changing in requirements, it should not include algorithms *sensitive to changes in data*. |
| **Evolutionary** | This model builds many iterations, each more complex than the other, therefore, requires the included algorithms to be *scalable* and *insensitive to changes* in data. |
| **Spiral** | This model divides the project into small segments, starting with a low number of requirements, adding complexity as the project moves through the various phases of the SDLC. Its strong focus on documentation, together with its management difficulties, require for *interpretable* algorithms.<br>The iterative nature of this model, starting from simple software and evolving to more complex software, requires that used algorithms are *scalable* and *flexible*. |
| **Agile** | The key features of this model provide an opportunity for heavy use of AI to maximize automation. It requires regular collaboration with stakeholders, accepting changes in requirements and planning mid-project. It rewards fast deliveries and short iterations over documentation. It is considered that the algorithms included in this phase must be *scalable* and *flexible*. |
| **RAD** | It prizes speed in deliveries, receiving feedback to refine it and accepting changes to requirements mid-project. It is efficient in small and medium-sized projects, lacking scalability. It should also include *computationally efficient* algorithms. |
| **Unified Process** | This model is recommended for large-scale and complex systems. It allows use-cases to evolve simultaneously with the architecture, providing better risk analysis. It divides work into increments and is also recommended for use with UML. The algorithms for this SDLC must be *flexible* and *scalable*. |

The literature review also underlined the fundamental position of the Waterfall model as the primordial SDLC, upon which other later models were built, following the same patterns in their approaches to software development. This justifies the choice of the waterfall model's phases as a general representative of a software development process for the sake of simplicity. They are: Requirements Gathering and Analysis, Design, Implementation and Coding, Testing, and System Operation and Maintenance.

Table 7 shows the phases of the SDLC, together with the activities that, according to the reviewed literature, were deemed compatible with the inclusion of artificial intelligence for automation purposes. Furthermore, the table lists the algorithms used for each activity in the researched literature and demonstrated satisfactory performance, even if they were not considered exactly the most efficient when testing.

Table 7 – SDLC phases and Compatible Algorithms

| SDLC phases | Activities | Algorithms |
|---|---|---|
| Requirements | Req. Elicitation/ Classification | KNN, Log R, NB, MNB, DT, RF, SVM, ANN, CNN, AdaBoost, NLP, BiLSTM, Fuzzy Similarity KNN, J48, BERT(NLP) |
| | Re. Prioritization | J48, KNN, NLP |
| | Risk Prediction | KNN, NB, DT, RF, J48, A1DE, CHIRP, CDT, CS-Forest, Decision Table\Nayve Bayes Hybrid Classifier |
| | Vulnerability Pred | KNN, Log R, DT, RF, NB, ANN, SVM |
| Design | Function/ module/ interface suggestion | ANN, Log R, DT, RF, RNN, MNB, GA, NLP |
| | Architectural Patterns | NB, DT, SVM, NLP |
| | Effort Estimation | ANN, NB, DT, RF, SVM, Lin R, K-Star, Case-Based Reasoning |
| | Cost Estimation | ANN, SVM, RF, GA Fuzzy Logic, COCOMO |
| | Risk Assessment | ANN, KNN, NB, DT |
| Implementation and Coding | NL-to-Code | CNN, RNN, NLP |
| | Code-to-Code | ANN, KNN, RF, LSTM |
| | Code-to-NL | LSTM, NLP |
| | Code Smell Detection | ANN, DT, RF, SVM, CNN, RNN, AdaBoost, Gradient Boosting, XGBoost, Bagging |
| | Vulnerability Pred | Log R, NB, DT, RF, SVM |
| Quality and Testing | Test Case Generation | MNB, DT, RF, SVM, AdaBoost, NLP |
| | Test Case Prioritization | ANN, DT, RF, SVM |
| | Soft fault Prediction | DT, SVM, CNN, MLP |
| | Vulnerability Detection | DT, RF, SVM, CNN, RNN |
| | Issue/ bug management | ANN, KNN, DT, SVM, CNN, RNN, NLP |
| Maintenance | Intrusion Detection | ANN, Log R, SVM, Clustering, CNN, RNN, LSTM |

Each algorithm has its strengths and weaknesses that may make it more compatible with some SDLC models than others. The algorithms that were tested for such characteristics and, therefore, considered for inclusion in the framework are the most common in the literature review, as shown in Table 8 below:

Table 8 – Advantages and Disadvantages of different algorithms

| ANN | SVM | KNN | NB |
|---|---|---|---|
| + Adaptability.<br>+ Handles complex non-linear data.<br>+ Robust with noisy data.<br>+ Parallel computation.<br>+ Handles high dimensional data.<br>- Black box nature.<br>- Computationally intensive.<br>- Complex model tuning.<br>- Prone to overfitting. | + Accuracy.<br>+ Efficiency in high dimensional space.<br>+ Outliers have low impact.<br>+ Can perform non-linear classification.<br>- High training time.<br>- Inefficient in cases with high quantities of training data.<br>- Not advised with high nº of training examples | + Robust with noisy data.<br>+ Simple.<br>+ Versatile.<br>+ Easy to implement.<br>+ High accuracy for high K value.<br>- Computationally intensive as K increases.<br>- Slow and inefficient with high data volume. | + High training speed.<br>+ High efficiency.<br>+ Insensitive to noisy features.<br>+ Requires a small amount of training data.<br>+ Scalable.<br>- Assumes independence of each feature.<br>- Sensitive to how input data is prepared. |
| **DT** | **RF** | **Log R.** | **AdaBoost** |
| + Interpretable.<br>+ Fast.<br>+ Scalable.<br>+ Computationally cheap.<br>+ Missing and error values have less impact.<br>- Prone to overfitting.<br>- Sensitive to changes in data. | + Good performance<br>+ User-friendly.<br>+ Versatile.<br>+ Overfitting resistance.<br>+ Can handle large amounts of data.<br>+ Outliers have little impact.<br>- Trees grow with increase in predictors.<br>- Time consuming. | + Easy to train.<br>+ Efficient when dealing with linear data.<br>+ Interpretable<br>- Poor performance with non-linear data.<br>- Poor performance with irrelevant and highly correlated features. | + Easy to code.<br>+ Low generalization error.<br>+ Computationally efficient.<br>+ Applicable to complex tasks.<br>+ Easily modified.<br>+ Flexible.<br>- Sensitive to outliers.<br>- Noisy when training.<br>- Needs large samples. |
| **CNN** | **RNN** | **LSTM** | **Lin R.** |
| + Easy to train.<br>+ Good performance.<br>+ Efficient in feature extraction.<br>+ Practical in its application.<br>+ Scalable<br>- Hardware dependent, as it requires a large memory to store all intermediate results.<br>- Is of no use if data is completely unstructured. | + Processes inputs of any length.<br>+ Remembers intermediate information.<br>+ Model size does not increase with increases in input data.<br>- Slow computation.<br>- Training is difficult.<br>- Exploding and vanishing gradients | + Captures and stores long term dependencies.<br>+ Can solve RNN's vanishing and exploding gradient problem.<br>+ Captures and remembers context between events.<br>- Training is time consuming.<br>- Computationally expensive.<br>- Limited scalability.<br>- Hard to parallelize.<br>- Requires high number of training data. | + Easy to train and implement.<br>+ Efficient when dealing with linear data.<br>+ Interpretable<br>+ Computationally efficient<br>+ Robust to outliers<br>- Poor performance with non-linear data.<br>- Susceptible to overfitting.<br>- Sensitivity to multicollinearity. |

## 4.2. Framework Proposal

The information collected in the literature review, conditioned on the previously mentioned assumptions, resulted in the generation of the artifact demonstrated in Table 9. This was done through an analysis of the characteristics of both SDLCs and algorithms, to find, among algorithms presented in Table 7 for each phase, the most compatible.

Table 9 – Framework to Leverage Artificial Intelligence in Software Development

| | SDLC Models | | | | | |
|---|---|---|---|---|---|---|
| | Waterfall | Evolutionary | Spiral | Agile | RAD | Unified Process |
| Requirements | DT, Log R, NB, NLP | DT, Log R, AdaBoost, CNN, NLP | DT, Log R, NLP | DT, Log R, AdaBoost, CNN, NLP | KNN, Log R, DT, NB, SVM, CNN, NLP | DT, Log R, AdaBoost, CNN, NLP |
| Design | Log R, Lin R, DT, NB, NLP (ANN, RF, RNN) | Lin R, Log R, DT, NLP (ANN, RF, RNN) | Lin R, Log R, DT, NLP (ANN, RNN) | Lin R, Log R, DT, NLP (ANN, RNN) | Lin R, Log R, DT, SVM, NB, KNN, NLP (ANN, RNN) | Lin R, Log R, DT, NLP (ANN, RNN) |
| Implementation & Coding | Log R, NB, DT, NLP (CNN, ANN, RNN, LSTM, KNN, RF) | CNN, Log R., DT, AdaBoost, NLP (ANN, RNN, LSTM, KNN, RF) | Log R, DT, NLP (CNN, ANN, RNN, LSTM, KNN, RF) | CNN, Log R., DT, AdaBoost, NLP (ANN, RNN, LSTM, KNN, RF) | CNN, Log R., NB, DT, SVM, KNN, NLP (ANN, RNN, LSTM, RF) | CNN, Log R., DT, AdaBoost, NLP (ANN, RNN, LSTM, KNN, RF) |
| Testing | DT, NLP | DT, CNN, AdaBoost, NLP | DT, NLP | DT, CNN, AdaBoost, NLP | KNN, DT, SVM, CNN, NLP | DT, CNN, AdaBoost, NLP |
| Maintenance | Log R. | Log R, CNN | Log R | Log R, CNN | Log R, SVM, CNN | Log R, CNN |

New Iteration

In some phases, algorithms not compatible with the respective SDLC are presented in parenthesis, meaning that these algorithms should only be used to build generative technologies present in activities within those phases, as they bypass some established SDLC characteristics used to restrict the algorithm's choice. This is because these algorithms are very efficient in building generative AI, however, as they are not considered the most compatible with the respective SDLCs, companies should use them at their own risk.

Additionally, it is also recognized the existence of more complex generative technologies, also known as Large Language Models (LLM), such as ChatGPT, many of which are available to be acquired by companies through the market and, consequently, available to carry out many activities of the software development process. It should be noted that although these LLMs are largely unaddressed, the algorithms present in this framework are used to create such solutions.

### 4.2.1. Preliminary Activities

The requirements phase, as previously established, is the initial phase of the project where the contracted company surveys the appropriate stakeholders about the desired characteristics and functionalities for the software to be developed. Before this phase begins, the organization needs to analyze the information provided by customers in the first contact, along with additional inquiries if necessary, to acquire enough information to make an informed decision about the SDLC to be used in the project. After that, the AI algorithms are chosen for all phases of the chosen SDLC, by consulting the proposed framework represented in Table 9. Then, a development plan is created for the AI solutions and the Integrated Development Environment (IDE), along with other appropriate complementary tools, are chosen. Finally, the development plan for each AI solution is followed, ending with their creation.

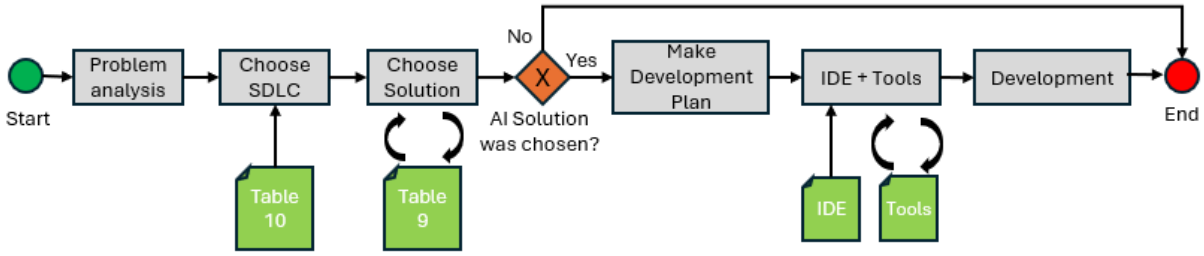


Figure 9 – Preliminary Development Activities Workflow

To facilitate the "Choose SDLC" activity, the conditions for when one of the SDLCs studied should be applied are explained in Table 10:

Table 10 – Conditions for SDLC Use

| SDLC | SDLC application |
|---|---|
| Waterfall Model | This model should be used when requirements are well defined and testable (Kramer, 2018), and is also more efficient in smaller projects (Ruparelia, 2010). |
| Evolutionary Model | It is appropriate for use in long projects where customer needs are clear and changes in requirement are anticipated, although not overly volatile. It should also be used when the customer prefers to use the core features of the software in advance instead of waiting for its final release (Adesina et al., 2020; *Evolutionary Model - Software Engineering*, 2019). |
| Spiral Model | The spiral model is indicated for projects of medium to high risk and long-term commitment, in which user needs need to be constantly reviewed (Alshamrani & Bahattab, 2015). |
| Agile | This SDLC philosophy is effective in large, complicated projects that exhibit requirement volatility. These models should also be applied in cases where the division of project tasks into increments is feasible and customers are available for regular contact (Adesina et al., 2020). |
| RAD | RAD is especially efficient in small and medium-sized projects with low complexity and a high degree of commitment from both developers and customers. A project using this model should also be relatively low complexity and with well-known and stable requirements (Berger & Beynon-Davies, 2009). |
| Unified Process | Its use is indicated for large and complex projects, where constant contact with users is possible, being requirements also volatile. This model should also be applied in cases where dividing project tasks into increments is feasible (Pressman, 2010). |

Additionally, note that the “Choose Solution” represented in Figure 9 is its own workflow, represented separately in Figure 10. This process represents the best approaches for AI solution sourcing: previously developed systems, AI models/ software provided by contracted companies, or developed internally from a foundational level. This process of investigating already built solutions reflects an organization’s desire to save time, effort, and capital, in opposition to building them internally.

The process works as follows: it starts with a simple search for any previously used AI software available to the company, that can also be applied to the current project with minimal adaptations. If such a system does not exist, or its use for the current project is, for any reason, undesirable, then the market should be explored for any available tool. After these processes, if both are negative, the organization assesses whether a satisfactory solution can be developed. If so, then the presented framework should be used to assist in the choice of the algorithms used to build AI models. Otherwise, the organization must proceed to use manual methods for the rest of this phase. But if any tool is found, hired or developed, it will have to be implemented to be ready for use.

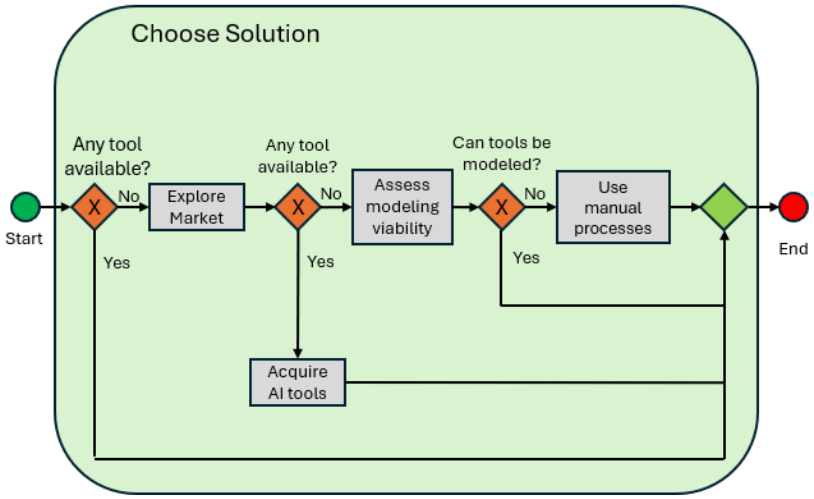


Figure 10 – Chosen Solution Activity Workflow

### 4.2.2. Requirements Gathering and Analysis Phase

After carrying out these preliminary activities, the requirements phase itself begins with the implementation (or not) of the AI tools acquired in the normal processes of this phase: the elicitation, classification, and validation of requirements. The normal functioning of these activities generates necessary documentation for the following phases.

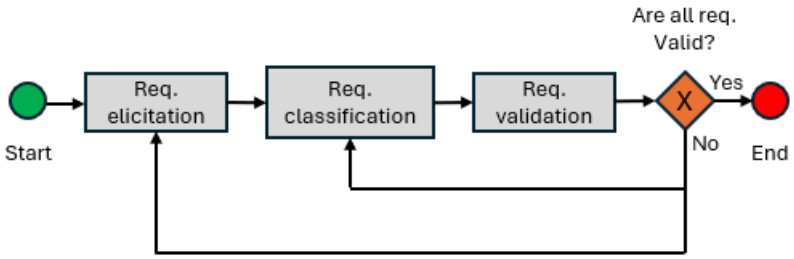


Figure 11 – Requirements Gathering and Analysis Phase Workflow

### 4.2.3. Design Phase

At this phase, the foundation for the rest of the project is planned, meaning that most of the AI solutions for each phase are also to be selected in this stage. This phase begins with a review of the documentation generated in the previous phase to better understand the context before a plan and architecture are drawn up. Then, the previously described AI tools are implemented to be used to estimate the risks, dispended effort, and potential cost of the project. Once this information is collected and reviewed, a plan for the remainder of the project is generated. Finally, a software architecture is made, which includes the user's interface.

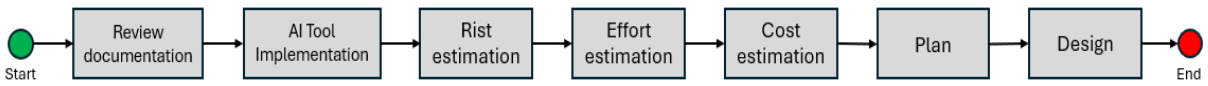


Figure 12 – Desing Phase Workflow

### 4.2.4. Implementation and Coding Phase

This phase represents a state where the models have already been acquired or trained, requiring only their implementation in the appropriate processes.

In the case of this phase, after the models have been implemented, software coding is carried out using the tools implemented to generate code snippets, complex code sequences, autocompletion, autocorrection, or any other method that proves compatible. Then, the code is reviewed for smells and possible cybersecurity vulnerabilities, which, if found, must be corrected.

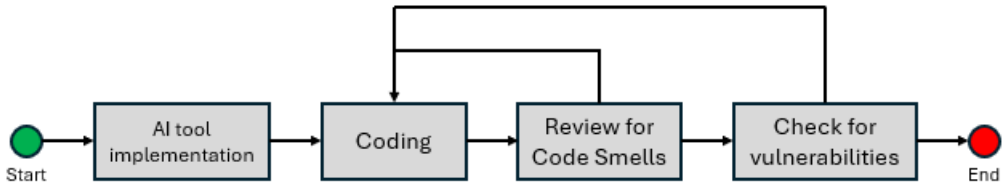


Figure 13 – Implementation and Coding Phase Workflow

### 4.2.5. Testing Phase

In the testing phase, the documentation generated in previous phases is reviewed to verify whether the committed goals were followed and accomplished. The information contained in the documentation is also used to plan the tests to be performed in this phase. These cases are generated in the "plan testing" activity, but if they are not considered sufficient, or their quality falls short, then the generation of test cases can be performed through the construction of AI models. After this process, the software is tested as intended and any fault detected is consequently corrected.

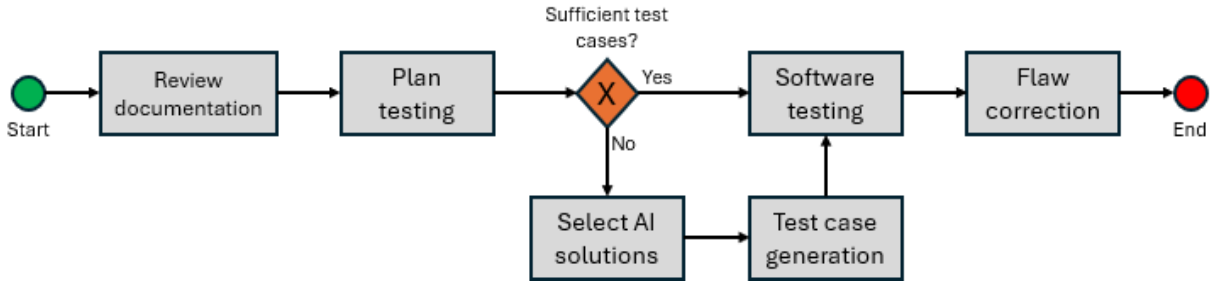


Figure 14 – Testing Phase Workflow

### 4.2.6. Maintenance Phase

Software maintenance was the phase where AI solutions were least explored, yet they can still provide a manner of utility in the post-launch state, as well as acquire information for developing new iterations. Before releasing the software into the environment, the chosen AI tools, if any, are implemented in the selected activities. These activities are carried out continuously and include bug detection and correction, intrusion detection, user support and user feedback by any means. These activities generate information that must be documented to be used in the continuous improvement process and in the generation of new iterations.

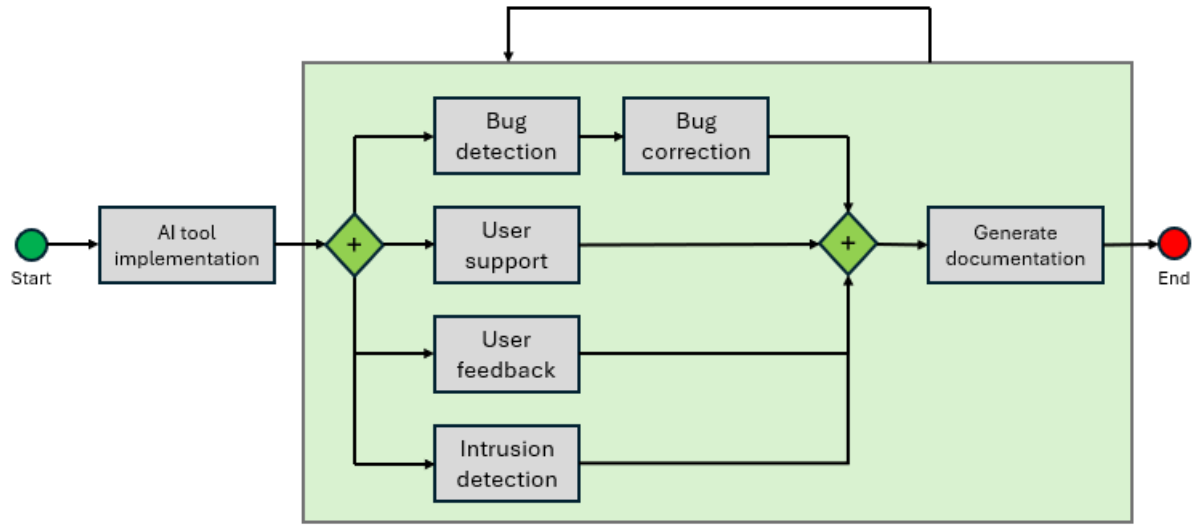


Figure 15 – Maintenance Phase Workflow

## 4.3. USE CASE

The purpose of this subsection is to provide a better understanding of how the framework would function in its application. As such, a fictional use case is presented below:

Odyssey Logistics is an internationally renowned logistics and supply chain management company specializing in product shipping and warehousing. To provide better transparency to its customers, it is currently considering creating real-time tracking software for the status and location of its customers' orders. Consequently, it contacted Gamma Solutions, a software development company known for its innovative approaches, with a team of skilled and experienced employees.

As a company that values innovation, Gamma Solutions decided to include artificial intelligence in the process of developing the requested software and, for this reason, decided to use the framework provided in this dissertation.

From the initial contact with the customer, it is expected that enough critical information will be acquired to perform the preliminary steps to start the project, which include the choice of the SDLC to be used. The insights gained from this initial interaction led to the following conclusions:

- The software to be developed is complex, as it needs to track numerous orders worldwide, in addition to being able to provide predictions for the delivery date and update them if necessary.
- It is a large-scale project.
- Many teams are available for incremental development if needed.
- The customer is available for regular contact with Gamma Solutions.

Based on this information, Gamma Solutions consulted Table 10 to facilitate the decision of the SDLC considered most appropriate for the project, which was decided to be the Unified Process. Following standard procedures, this model was adapted according to the specific circumstances of the project.

With the SDLC chosen, the company established the processes/ activities that include AI-based solutions, opting for the maximum automation possible through the software development process and its activities. Among the AI solutions that Gamma Solutions will integrate in this project, only three are highlighted in this use case: requirement elicitation, effort estimation, and code generation.

To ensure the development of accurate software in accordance with Odyssey Logistics' specifications, it is necessary to extract precise requirements from them, especially when considering the complexity of the system to be developed. Therefore, Gamma Solutions decided to use a requirement elicitation and classification system in its project. To this end, the framework present in Table 9 was consulted, which includes the recommended algorithms for building the AI solution. Among these, AdaBoost was considered the optimal choice, as both the project and the initial phase of the unified process are complex and AdaBoost thrives in complex tasks. AdaBoost also necessitates for high quantities of samples, which is not a problem for Gamma Solutions due to its ability to leverage information from an abundance of similar previous projects. As the AI solution must have textual inputs, therefore being an NLP, it was also decided that the AI solution be used together with TF-IDF, as the customer inputs will be inserted in text format. Since Gamma Solutions did not have a legacy system that could be used in this case, nor did they find a system on the market that could meet their needs, they decided to develop the AI solution in-house.

The inclusion of an effort estimation AI system was also considered essential for Gamma Solutions to optimize its resource allocation by assessing the time and labor required for the project. Therefore, after consulting the framework, they decided to build an AI solution based on Linear Regression. This decision is supported by Linear Regression's proficiency in dealing with large datasets effectively, ease of implementation and simplicity. Given the existence of a legacy system similar to the one required, developed for a prior project and using the chosen algorithm, it was decided to leverage the existing legacy system and adapt it according to the project's needs.

For the development of the software itself, Gamma Solutions decided to implement a code generation AI to assist developers in their normal activity. After examining the framework, an RNN-based solution was chosen as it can process inputs of any length and has an internal memory to process them. Gamma Solutions did not have a legacy system that could be adapted into this project; however, they found an NLP system available on the market that met their needs, and, therefore, proceeded to purchase it.

More generally, Gamma Solutions followed the same process for other activities to include suitable AI solutions in the chosen SDLC. At each point, they consulted the proposed framework, presented in Table 9, to decide on the appropriate AI algorithms to use in each activity.

After choosing the AI algorithms, and even still before the development process begins according to the unified process, Gamma Solutions proceeded to acquire the necessary AI tools. To achieve this goal, the company first considered the existence of any legacy AI systems used in a previous project and found some previously implemented tools. Some of these tools were considered appropriate to be included in the project, as they allowed the company to save time and costs. The root cause of this decision was that adapting it to the current project required fewer resources compared to acquiring similar solutions on the market or developing them again.

Subsequently, Gamma Solutions conducted a market exploration to identify potential AI tools to implement in the remaining processes. The solutions available on the market that were found to be satisfactory were acquired from the respective buyers to be implemented in the process. The remaining activities that did not have legacy systems that could be used, nor satisfactory solutions on the market were required for their in-house development. Finally, the Integrated Development Environment (IDE) tools were chosen for the development and necessary adaptations of the AI algorithms.

Finally, the development of AI-based solutions was carried out, thus concluding the preliminary phase. GAMMA Solutions implemented the AI models now available at the beginning of each phase and for each of the chosen activities. The rest of the development proceeds as intended in the unified process.

## 4.4. Evaluation & Discussion

For the purpose of evaluating the proposed framework, and as determined by Peffers' et al. (2006) guidelines for DSR, the constructed artifact is scrutinized by three experts in the field, whose background is briefly demonstrated in Table 11. Such scrutiny was accomplished through interviews where the framework is demonstrated, along with the logic behind its creation, as well as its application method.

Table 11 – Participants of the Framework's Evaluation

| Expert Nº | Background | Domain |
|---|---|---|
| Expert 1 | Guest Assistant Professor at Nova IMS, PHD in "Business-IT Alignment" | - Business-IT alignment<br>- BPM |
| Expert 2 | CEO, PHD in Information Systems and Computer Engineering | - Project Management<br>- Software Development |
| Expert 3 | Professor at Nova IMS, PHD in "Sciences and Information Technologies" | - Software Development |

Each interview was conducted with each expert individually on zoom by presenting the context that motivated the construction of the framework, the framework itself, and its application method. Each expert previously agreed that the meeting would be recorded to transcribe their answers to the following questions, presented in the appendix section:

> **Question 1** – What is your opinion on the usefulness of the proposed framework? Could you provide a further explanation?
>
> **Question 2** – Do you have any criticism concerning the framework? If so, could you offer an explanation as to why?
>
> **Question 3** – Could you offer any suggestions for potential refinements of the proposed framework?

These questions mark an opportunity to obtain feedback from three independent sources that can be critical in identifying unforeseen factors within the framework that may negatively influence a company with its application, as well as providing an opportunity for its improvement through the expertise of interviewees. Please note that all quotations included in this section that refer to the arguments presented by the experts in the respective interviews are present in the "Annexes" section of the dissertation.

After completing a brief presentation of the framework, and when asked about its usefulness of the proposed framework, the feedback expressed from the interviewees was unanimously positive, with all highlighting the importance of the work developed in this dissertation. Expert 2 confirmed the rationale behind conducting this study, namely, the potential for AI-embedded software development to offer numerous advantages with the potential to enhance the productivity of software developers, in turn, mitigating the current shortage of software developers. He reinforces this argument by stating that "(…) there is a growing deficit of software developers and one of the ways to combat this shortage is by giving them tools that enable already existing professionals to do more".

The iterative nature of the framework was also praised by Expert 1, stating that with adequate monitorization for each iteration there is the possibility of continuous improvement of the selected AI solutions. Furthermore, the interviewee highlighted the relevance of the framework for those performing project management functions.

Despite the positive responses obtained in the interviews, some criticisms of the proposed solution were highlighted by experts in the context of the second question. Expert 1 postulated the possibility of existing more than one algorithm chosen for a given AI solution, while the framework does not address the possible existence of compatibility issues from the complementary use of two (or more) AI algorithms in building a given AI solution. He supports this position by stating: "There is, eventually, some complexity if the algorithms have different construction bases, for example, built in different programming languages". Furthermore, he argues that interactions with the same algorithms result in more natural results, meaning it is preferable to construct AI solutions using just one algorithm to avoid undesirable complications and suboptimal outputs. This argument is considered valid and should be introduced as a consideration for others using the framework.

Both Experts 2 and 3 independently argued that the name chosen for the solution, i.e., for the framework, does not correspond to the nature of the artifact generated in this study. Expert 2 asserts that the artifact aligns more closely with the definition of a method, as the solution presents "a set of steps that must be followed to define which algorithms to use at a given moment in the development process". Expert 3 also shows similar concerns regarding the definition of the artifact, albeit for different reasons. Which means that, unlike Expert 2, who believes that the artifact should be redefined due to the presence of flows in its underlying recommendation, Expert 3 argues that due to the table format of the framework with outputs present in every cell, it lacks a clear progression, being this the reason why it should be considered a model. This is corroborated by his statements: "a framework would be the steps, a way of applying something" and that this is model because "it is something that crosses (i.e., intersects columns with rows)".

Before addressing these concerns, it is important to define what a framework is: a framework is "a system of rules, ideas, or beliefs that is used to plan or decide something" (*Framework*, 2024). As the displayed artifact provides a structured approach for selecting the most suitable algorithms for a given phase within one of the studied SDLCs, it is aligned with the previous definition, and it can be concluded that the artifact can be considered a framework.

Additionally, as Expert 3's primary concern is the tabular format of the artifact, it is pertinent to reflect on other artifacts represented in a similar way that are still considered frameworks, such as Zachman's framework. This framework is represented in table form, featuring its outputs within each cell, without a discernable flow as outlined by the expert.

Furthermore, within this expert's definition of a framework, it can be argued that the proposed solutions would still fit within it, since the framework does adhere to a phased software development process represented in a downwards flow that depicts the process of software development and its respective phases, while outlining the algorithms considered most appropriate for each one.

Referring to suggestions considered by experts to enhance the framework, Expert 2 suggests a more precise recommendation of which algorithms should be used, “why one algorithm is better at a given moment than at another”. Although a simple generalized model that recommends the best algorithms to be used at a given phase of the SDLC has noticeable advantages, being recognized as of high importance by all experts, the lack of recommendation for the specific algorithm to be used could further strengthen the value of the study. For this reason, Expert 1, who also recognized the significance of such specific recommendations for the same reason, proposed implementing the framework in the environment for further testing in future work. The acquired feedback from the application of the framework across diverse real-world scenarios would provide not only an opportunity to verify its actual effectiveness, but also acquire information that would further discern the best algorithm for a given activity within a given industry.

Expert 3 argued the significance of the creation of a fictional use-case with concrete examples pertaining to particular activities within the SDLC phases, so that it helps professionals relate and perceive the usefulness of the framework according to their specific function within the software development process, as well as understanding how it is used in this process.

Expert 1 also suggests that a Project Manager should be interviewed, as their feedback could provide valuable insights to strengthen the framework. This was done in the interview with Expert 2, who is the CEO of a young company and has extensive experience in the activity in question.

Finally, Expert 3 suggested that the insertion of the references used build the framework in the cells of the framework, where the algorithms are present. He argues that the artifact, as it is a synthesis of the knowledge acquired in the literature review, should have that fact demonstrated. Otherwise, a third party could see it as something that was simply conceived by its author, without an apparent academic basis. Therefore, references to articles where the algorithms were tested by researchers will be inserted as requested.

### 4.5. Revised Framework

The interviews with the experts resulted in a set of suggestions that were taken into account to improve the solution. As advocated by Expert 3, references pertaining to the algorithms found in the literature for each phase were included in the respective rows, together with references to the SDLC in the appropriate columns, as the characteristics were extracted from them. Also note that some AI solutions can be built together with more than one of the presented algorithms, which can be problematic as compatibility and complexity issues can arise in their integration. For this reason, it is preferable for a given AI solution to be built using just one AI algorithm.

Table 12 – Revised Proposed Framework

| | SDLC Models | | | | | | |
|---|---|---|---|---|---|---|---|
| | Waterfall | Evolutionary | Spiral | Agile | RAD | Unified Process | |
| Requirements | DT, Log R, NB, NLP | DT, Log R, AdaBoost, CNN, NLP | DT, Log R, NLP | DT, Log R, AdaBoost, CNN, NLP | KNN, Log R, DT, NB, SVM, CNN, NLP | DT, Log R, AdaBoost, CNN, NLP | [15], [13], [16], [14], [11] |
| Design | Log R, Lin R, DT, NB, NLP (ANN, RF, RNN) | Lin R, Log R, DT, NLP (ANN, RF, RNN) | Lin R, Log R, DT, NLP (ANN, RNN) | Lin R, Log R, DT, NLP (ANN, RNN) | Lin R, Log R, DT, SVM, NB, KNN, NLP (ANN, RNN) | Lin R, Log R, DT, NLP (ANN, RNN) | [19], [17], [21], [20], [22], |
| Implementation & Coding | Log R, NB, DT, NLP (CNN, ANN, RNN, LSTM, KNN, RF) | CNN, Log R., DT, AdaBoost, NLP (ANN, RNN, LSTM, KNN, RF) | Log R, DT, NLP (CNN, ANN, RNN, LSTM, KNN, RF) | CNN, Log R., DT, AdaBoost, NLP (ANN, RNN, LSTM, KNN, RF) | CNN, Log R., NB, DT, SVM, KNN, NLP (ANN, RNN, LSTM, RF) | CNN, Log R., DT, AdaBoost, NLP (ANN, RNN, LSTM, KNN, RF) | [23], [24], [25], [26], [27] |
| Testing | DT, NLP | DT, CNN, AdaBoost, NLP | DT, NLP | DT, CNN, AdaBoost, NLP | KNN, DT, SVM, CNN, NLP | DT, CNN, AdaBoost, NLP | [33], [28], [36], [31], [32], [34], [37] |
| Maintenance | Log R. | Log R, CNN | Log R | Log R, CNN | Log R, SVM, CNN | Log R, CNN | [29] |
| | (Adenowo & Adenowo, 2020; Kramer, 2018; Ruparelia, 2010) | (Adesina et al., 2020; Chandra, 2015; *Evolutionary Model - Software Engineering*, 2019; May & Zimmer, 1996) | (Alshamrani & Bahattab, 2015; Boehm, 1986) | (Adesina et al., 2020; Fowler & Highsmith, 2001a; Gheorghe et al., 2020b; S. Sharma et al., 2012) | (Adesina et al., 2020; Berger & Beynon-Davies, 2009; Beynon-Davies et al., 1999b; Daud et al., 2010) | (Jacobson et al., 1999; Pressman, 2010) | |

New Iteration

# 5. CONCLUSION

As the existing literature was thoroughly reviewed, enough information was acquired to confirm the existing trend relating to an increased interest in the integration of artificial intelligence in software development. However, the scarcity of written articles that advise on good practices for inclusion of artificial intelligence in software development was also observed.

A surprising discovery during the systematic literature review was a framework that underlined the potential algorithms to be included in the different phases of the software development process (Shankar & Chaudhari, 2023). Yet, this framework did not consider the different constraints that surge with the use of the different SDLC, confirming the existence of the research gap initially.

## 5.1. SYNTHESIS OF THE DEVELOPED WORK

The framework proposed in this paper, as initially established, was developed with the intention of facilitating the inclusion of artificial intelligence in the various activities of the software development process. The research gap that justifies its creation, along with the proper justification that supports the need to address it, is uncovered in the first chapter of this dissertation.

The final objective of the research to cover the previously defined research gap was to build a framework, that is, an artifact. Thus, Design Science Research was employed, through a combination of both interpretations of Peffers et al. (2006) and Hevner et al. (2007), as it was perceived to provide a crucial basis to guide the development of the framework. The foundational knowledge used in the generation of the framework was acquired through the literature review, where both topics, artificial intelligence and software development, were researched individually. Subsequently, a systematic literature review was conducted using the PRISMA Statement to uncover existing knowledge centered on the use of AI in software development, with a primary, but not exclusive, focus on the algorithms used in the various activities of the software development process. With the acquisition of sufficient information, the artifact's design was conducted along with the development of several workflows that demonstrate its use within the normal software development process. Finally, for evaluation purposes, interviews were carried out independently with three experts, in order to identify the strengths, weaknesses, and opportunities for improvement of the proposed solution. This feedback was used to enhance the framework through a revision of the framework, which is also present in section 4.

The research questions defined in the inception of this work were answered throughout its execution. The first question, "What are the impacts of applying AI to the software development process?", was answered in the second question of the systematic literature review, where the advantages and disadvantages of the inclusion of AI in the software development process were listed. For the second research question, "How can we help software development companies take advantage of AI?", was answered through the construction of the framework, where it was decided that the best method to help companies include AI in their processes was by recommending algorithms compatible with the different phases of the chosen SDLC. Lastly, the research question, "What AI technologies are appropriate for each phase of the software development life cycle?", was answered in the third question of the systematic literature review, where the algorithms researched for different tasks were

listed. This can be summarized in Table 7, present in the “assumptions” subchapter of the fourth chapter. Furthermore, this research question was also answered in the proposed framework, as it represents the application of these technologies not only to the different phases of the software development, but also to the different SDLCs.

## 5.2. LIMITATIONS

Throughout the course of this study and the design of the artifact, some limitations were identified. The first one that should be considered is that the framework only covers a limited set of AI algorithms, omitting many others that were not as prevalent in the reviewed literature, and that could also be used to build AI solutions that could be advantageous for the software development process. The same phenomenon also occurs with SDLC models, since only some of the existing SDLC were considered in this work, ignoring many other available models, and limiting the framework’s applicability to only the included SDLC.

Another limitation noted in this work is that the framework recommends a set of algorithms that can be used in a given phase of the chosen SDLC. However, it does not specify the specific algorithm to be used in a given case for a specific activity, as noted by many of the experts interviewed.

The evaluation of the framework was conducted simply from a theoretical standpoint, and was not implemented in the environment for testing purposes, as required by Peffers’ et al. (2006) DSR guidelines. This results in difficulty in gaining a realistic understanding of the framework’s strengths and weaknesses, opportunities for improvement, and any unforeseen factors. This is especially important as the ultimate goal of this work is to provide assistance to companies and, therefore, such a field test would be desired.

Finally, the framework was evaluated by three experts who provided suggestions to improve it. This resulted in the review of some points of the generated solution, which was carried out in the final subsection of the previous chapter. However, the framework was not evaluated a second time, and therefore does not take into account the emergence of issues that may have arisen as a consequence of the present changes.

## 5.3. Recommendations for Future Research

With the completion of this work, opportunities for further research are revealed and presented in this subsection. The first recommendation arises from the framework's lack of testing in a real-world scenario. Its application in the environment would be advised, as it would allow a realistic view of the strengths, weaknesses, and opportunities for improvement that could require its further reformulation with the purpose of providing the best outcomes possible for those who apply it. Therefore, such application in diverse sectors would be desirable. This would allow a better understanding of the best procedures in different sectors and the best possible algorithms for each activity in different use cases.

A limitation defined in the previous subchapter is that the framework only covers a limited set of AI algorithms and SDLC. Therefore, it would be useful for similar work to be conducted with the aim of deepening the research, both on the algorithms and on the models that were not included.

As artificial intelligence is a current topic of academic and corporate interest, it is expected to evolve rapidly in the software development industry. In the context of this study, it is advisable that future researchers pay attention to these new AI developments, as new techniques may emerge that could lead to the updating of this framework or the development of new ones.

## BIBLIOGRAPHICAL REFERENCES


Abdel-Jaber, H., Devassy, D., Al Salam, A., Hidaytallah, L., & EL-Amir, M. (2022). A Review of Deep Learning Algorithms and Their Applications in Healthcare. *Algorithms*, *15*(2), 71. https://doi.org/10.3390/a15020071

Acemoglu, D., & Restrepo, P. (2018a). Artificial Intelligence, Automation, and Work. In *The Economics of Artificial Intelligence: An Agenda* (pp. 197–236). University of Chicago Press. https://www.nber.org/books-and-chapters/economics-artificial-intelligence-agenda/artificial-intelligence-automation-and-work

Acemoglu, D., & Restrepo, P. (2018b). Low-Skill and High-Skill Automation. *Journal of Human Capital*, *12*(2), 204–232. https://doi.org/10.1086/697242

Acharya, B., & Sahu, P. K. (n.d.). Software Development Life Cycle Models: A Review Paper. https://www.academia.edu/45633866/SOFTWARE_DEVELOPMENT_LIFE_CYCLE_MODELS_A_REVIEW_PAPER

Adenowo, A., & Adenowo, B. (2020). Software Engineering Methodologies: A Review of the Waterfall Model and Object- Oriented Approach. *International Journal of Scientific and Engineering Research*, *4*, 427–434.

Adesina, Odule, Alatishe, & Morafa. (2020). A Survey of Software Engineering Models, Comparisons and Scenario of Projects. https://scholar.googleusercontent.com/scholar?q=cache:s2FyxVXuB7IJ:scholar.google.com/+(Adesina+et+al.,+2020)+sdlc&hl=pt-BR&as_sdt=0,5

Al Asheeri, M. M., & Hammad, M. (2019). Machine Learning Models for Software Cost Estimation. *2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies*, 1–6. https://doi.org/10.1109/3ICT.2019.8910327

Alshamrani, A., & Bahattab, A. (2015). A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model. *International Journal of Computer Science Issues (IJCSI)*. https://www.academia.edu/10793943/A_Comparison_Between_Three_SDLC_Models_Waterfall_Model_Spiral_Model_and_Incremental_Iterative_Model

*Artificial Intelligence Act*. (2023). https://www.europarl.europa.eu/doceo/document/TA-9-2023-0236_EN.pdf

Auria, L., & Moro, R. A. (2008). Support Vector Machines (SVM) as a Technique for Solvency Analysis. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.1424949

Aychew, M., & Alemneh, E. (2022). Selection of Architectural Patterns based on Tactics. *2022 International Conference on Information and Communication Technology for Development for Africa (ICT4DA)*, 13–18. https://doi.org/10.1109/ICT4DA56482.2022.9971369

Bassi, D., & Singh, H. (2023). A Systematic Literature Review on Software Vulnerability Prediction Models. *IEEE Access*, *11*, 110289–110311. https://doi.org/10.1109/ACCESS.2023.3312613

Batool, I., & Khan, T. A. (2022). Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review. *Computers and Electrical Engineering*, *100*, 107886–107886. https://doi.org/10.1016/j.compeleceng.2022.107886

Berger, H., & Beynon-Davies, P. (2009). The utility of rapid application development in large-scale, complex projects. *Information Systems Journal*, *19*(6), 549–570. https://doi.org/10.1111/j.1365-2575.2009.00329.x

Beynon-Davies, P., Carne, C., Mackay, H., & Tudhope, D. (1999). Rapid application development (RAD): An empirical review. *European Journal of Information Systems*, *8*. https://doi.org/10.1057/palgrave.ejis.3000325

Boateng, E. Y., Otoo, J., & Abaye, D. A. (2020). Basic Tenets of Classification Algorithms K-Nearest-Neighbor, Support Vector Machine, Random Forest and Neural Network: A Review. *Journal of Data Analysis and Information Processing*, *08*(04), 341–357. https://doi.org/10.4236/jdaip.2020.84020

Bocu, R., Baicoianu, A., & Kerestely, A. (2023). An Extended Survey Concerning the Significance of Artificial Intelligence and Machine Learning Techniques for Bug Triage and Management. *IEEE Access*, *11*, 123924–123937. https://doi.org/10.1109/ACCESS.2023.3329732

Boehm, B. (1986). A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, *11*(4), 14–24. https://doi.org/10.1145/12944.12948

Brar, P., & Nandal, D. (2022). A Systematic Literature Review of Machine Learning Techniques for Software Effort Estimation Models. In H. K. Mittal, V. Jain, & Z. Polkowski (Eds.), *2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT)* (pp. 494–499). IEEE. https://doi.org/10.1109/CCiCT56684.2022.00093

Brocke, J. vom, Hevner, A., & Maedche, A. (2020). Introduction to Design Science Research (pp. 1–13). https://doi.org/10.1007/978-3-030-46781-4_1

Burns, E., Lawton, G., & Rosencrance, L. (2022). *What is Logistic Regression? - Definition from SearchBusinessAnalytics*. Business Analytics. https://www.techtarget.com/searchbusinessanalytics/definition/logistic-regression

Chandra, V. (2015). Comparison between Various Software Development Methodologies. *International Journal of Computer Applications*, *131*(9), 7–10. https://doi.org/10.5120/ijca2015907294

Chuanjian, C., Shuze, G., & Hua, W. (2023). Research on Software Development Based on Low-Code Technology. *2023 2nd International Conference on Artificial Intelligence and Autonomous Robot Systems (AIARS)*, 210–213. https://doi.org/10.1109/AIARS59518.2023.00049

Darandale, S., & Mehta, R. (2022). Risk Assessment and Management using Machine Learning Approaches. *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, 663–667. https://doi.org/10.1109/ICAAIC53929.2022.9792870

Daud, N. M. N., Bakar, N. A. A. A., & Rusli, H. M. (2010). Implementing rapid application development (RAD) methodology in developing practical training application system. *2010 International Symposium on Information Technology*, 1664–1667. https://doi.org/10.1109/ITSIM.2010.5561634

Deep Learning | Introduction to Long Short Term Memory. (2019, January 16). *GeeksforGeeks*. https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/

Dehaerne, E., Dey, B., Halder, S., De Gendt, S., & Meert, W. (2022). Code Generation Using Machine Learning: A Systematic Review. *IEEE Access*, *10*, 82434–82455. https://doi.org/10.1109/ACCESS.2022.3196347

Dewangan, S., Rao, R. S., Mishra, A., & Gupta, M. (2022). Code Smell Detection Using Ensemble Machine Learning Algorithms. *Applied Sciences*, *12*(20), Article 20. https://doi.org/10.3390/app122010321

Dwivedi, Y. K., Hughes, L., Ismagilova, E., Aarts, G., Coombs, C., Crick, T., Duan, Y., Dwivedi, R., Edwards, J., Eirug, A., Galanos, V., Ilavarasan, P. V., Janssen, M., Jones, P., Kar, A. K., Kizgin, H., Kronemann, B., Lal, B., Lucini, B., … Williams, M. D. (2021). Artificial Intelligence (AI): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research,

practice and policy. *International Journal of Information Management*, *57*, 101994–101994. https://doi.org/10.1016/j.ijinfomgt.2019.08.002

*Evolutionary Model—Software Engineering*. (2019, April 30). GeeksforGeeks. https://www.geeksforgeeks.org/software-engineering-evolutionary-model/

Fowler, M., & Highsmith, J. (2001). The Agile Manifesto. https://andrey.hristov.com/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf

*Framework*. (2024, June 5). https://dictionary.cambridge.org/dictionary/english/framework

Gheorghe, A.-M., Gheorghe, I., & Iatan, I. (2020). Agile Software Development. *Informatica Economica*, *24*, 90–100. https://doi.org/10.24818/issn14531305/24.2.2020.08

Hevner, A. R. (2007). A Three Cycle View of Design Science Research. https://www.researchgate.net/publication/254804390_A_Three_Cycle_View_of_Design_Science_Research

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. https://www.researchgate.net/publication/201168946_Design_Science_in_Information_Systems_Research

Hourani, H., Hammad, A., & Lafi, M. (2019). The Impact of Artificial Intelligence on Software Testing. *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, 565–570. https://doi.org/10.1109/JEEIT.2019.8717439

*IBM Global AI Adoption Index 2022 | IBM*. (2022). https://www.ibm.com/watson/resources/ai-adoption

Iftikhar, A., Musa, S., Alam, M., & Su'ud, M. M. (2020). Artificial Intelligence Based Risk Management in Global Software Development: A Proposed Architecture to Reduce Risk by Using Time, Budget and Resources Constraints. *Journal of Computational and Theoretical Nanoscience*, *17*(2), Article 2. https://doi.org/10.1166/jctn.2020.8735

Imtiaz, S., Amin, M. R., Do, A. Q., Iannucci, S., & Bhowmik, T. (2021). Predicting Vulnerability for Requirements. *2021 IEEE 22nd International Conference on Information Reuse and Integration for Data Science (IRI)*, 160–167. https://doi.org/10.1109/IRI51335.2021.00028

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). The Unified Process. https://people.eecs.ku.edu/~saiedian/810/Readings/rup-unified-process.pdf

Jagli, D., & Temkar, R. (2013). The Unified Approach for Organizational Network Vulnerability Assessment. *International Journal of Software Engineering & Applications*, *4*(5), 37–48. https://doi.org/10.5121/ijsea.2013.4503

Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets*, *31*(3), 685–695. https://doi.org/10.1007/s12525-021-00475-2

Kalcheva, N., Todorova, M., & Marinova, G. (2020). Naive Bayes Classifier, Decision Tree and Adaboost Ensemble Algorithm—Advantages and Disadvantages. *6th International Scientific Conference ERAZ - Knowledge Based Sustainable Development*, 153–157. https://doi.org/10.31410/ERAZ.2020.153

Korzeniowski, Ł., & Goczyła, K. (2019). Artificial intelligence for software development—The present and the challenges for the future. *Bulletin of the Military University of Technology*, *68*(1), Article 1. https://doi.org/10.5604/01.3001.0013.1464

Kramer, M. (2018). Best Practices in Systems Development Lifecycle: An Analyses Based on the Waterfall Model. *9*, 77–84.

Kumar, R., Naveen, V., Kumar Illa, P., Pachar, S., & Patil, P. (2023). The Current State of Software Engineering Employing Methods Derived from Artificial Intelligence and Outstanding Challenges. In L. Kumre & V. Chaurasia (Eds.), *2023 1st International Conference on Innovations in High Speed Communication and Signal Processing (IHCSP)* (pp. 105–108). IEEE. https://doi.org/10.1109/IHCSP56702.2023.10127112

Lasi, H., Fettke, P., Kemper, H.-G., Feld, T., & Hoffmann, M. (2014). Industry 4.0. *Business & Information Systems Engineering*, *6*(4), 239–242. https://doi.org/10.1007/s12599-014-0334-4

*Linear Regression in Machine learning—GeeksforGeeks*. (n.d.). Retrieved June 3, 2024, from https://www.geeksforgeeks.org/ml-linear-regression/

Liu, K., Reddivari, S., & Reddivari, K. (2022). Artificial Intelligence in Software Requirements Engineering: State-of-the-Art. *2022 IEEE 23rd International Conference on Information Reuse and Integration for Data Science (IRI)*, 106–111. https://doi.org/10.1109/IRI54793.2022.00034

López-Martín, C. (2022). Machine learning techniques for software testing effort prediction. *Software Quality Journal*, *30*(1), Article 1. https://doi.org/10.1007/s11219-020-09545-8

M. Mijwil, M. (2021). Artificial Neural Networks Advantages and Disadvantages. *Mesopotamian Journal of Big Data*, *2021*, 29–31. https://doi.org/10.58496/MJBD/2021/006

Malhotra, R., Jain, B., & Kessentini, M. (2023). Examining deep learning's capability to spot code smells: A systematic literature review. *Cluster Computing*, *26*(6), Article 6. https://doi.org/10.1007/s10586-023-04144-1

May, E., & Zimmer, B. (1996). The Evolutionary Development Model for Software. *Hewlett-Packard Journal*. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5304a6d70439f180af1e349d518cb1d20b99e4a8

McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (2006). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955. *AI Magazine*, *27*(4), Article 4. https://doi.org/10.1609/aimag.v27i4.1904

Menshawy, R. S., Yousef, A. H., & Salem, A. (2021). Code Smells and Detection Techniques: A Survey. *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, 78–83. https://doi.org/10.1109/MIUCC52538.2021.9447669

Moroz, E. A., Grizkevich, V. O., & Novozhilov, I. M. (2022). The Potential of Artificial Intelligence as a Method of Software Developer's Productivity Improvement. In S. Shaposhnikov & Prof. P. Str. 5 saint Petersburg Electrotechnical University "LETI" Saint Petersburg (Eds.), *2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)* (pp. 386–390). IEEE. https://doi.org/10.1109/ElConRus54750.2022.9755659

Naseem, R., Shaukat, Z., Irfan, M., Shah, M. A., Ahmad, A., Muhammad, F., Glowacz, A., Dunai, L., Antonino-Daviu, J., & Sulaiman, A. (2021). Empirical Assessment of Machine Learning Techniques for Software Requirements Risk Prediction. *Electronics*, *10*(2), Article 2. https://doi.org/10.3390/electronics10020168

Navaei, M., & Tabrizi, N. (2023). Impact of Machine Learning on Software Development Life Cycle. In H. Kaindl, H. Kaindl, H. Kaindl, M. Mannion, L. Maciaszek, & L. Maciaszek (Eds.), *Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering* (Vols. 2023-April, pp. 718–726). SCITEPRESS - Science and Technology Publications. https://doi.org/10.5220/0011997200003464

Pannu, A. (2015). Artificial Intelligence and its Application in Different Areas. *4*(10). https://www.ijeit.com/Vol%204/Issue%2010/IJEIT1412201504_15.pdf

Peffers, K., Tuunanen, T., Gengler, C., & Rossi, M. (2006). The design science research process: A model for producing and presenting information systems research. *Proceedings Design Research Information Systems and Technology DESRIST'06*, *24*. https://www.researchgate.net/publication/238077290_The_design_science_research_process_a_model_for_producing_and_presenting_information_systems_research

Petersen, K., Wohlin, C., & Baca, D. (2009). The Waterfall Model in Large-Scale Development. In F. Bomarius, M. Oivo, P. Jaring, & P. Abrahamsson (Eds.), *Product-Focused Software Process Improvement* (Vol. 32, pp. 386–400). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-02152-7_29

Pressman, R. S. (2010). Software engineering: A practitioner's approach (7th ed). McGraw-Hill Higher Education. https://www.mlsu.ac.in/econtents/16_EBOOK-7th_ed_software_engineering_a_practitioners_approach_by_roger_s._pressman_.pdf

Quba, G. Y., Al Qaisi, H., Althunibat, A., & AlZu'bi, S. (2021). Software Requirements Classification using Machine Learning algorithm's. *2021 International Conference on Information Technology (ICIT)*, 685–690. https://doi.org/10.1109/ICIT52682.2021.9491688

Ramirez, A., Romero, J. R., & Simons, C. L. (2019). A Systematic Review of Interaction in Search-Based Software Engineering. *IEEE Transactions on Software Engineering*, *45*(8), Article 8. https://doi.org/10.1109/TSE.2018.2803055

Ricca, F., Marchetto, A., & Stocco, A. (2021). AI-based Test Automation: A Grey Literature Analysis. *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 263–270. https://doi.org/10.1109/ICSTW52544.2021.00051

Royce, D. W. W. (n.d.). Managing the Development of Large Software Systems. https://blog.jbrains.ca/assets/articles/royce1970.pdf

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, *35*(3), 8–13. https://doi.org/10.1145/1764810.1764814

Sainani, A., Anish, P. R., Joshi, V., & Ghaisas, S. (2020). Extracting and Classifying Requirements from Software Engineering Contracts. *2020 IEEE 28th International Requirements Engineering Conference*, 147–157. https://doi.org/10.1109/RE48521.2020.00026

Saini, N., & Britto, R. (2021). Using Machine Intelligence to Prioritise Code Review Requests. *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 11–20. https://doi.org/10.1109/ICSE-SEIP52600.2021.00010

Samant, N., Limaye, H., Bapat, A., Shinde, S., & Nerurkar, A. (2023). Optimizing Issue Tracking Systems using Deep Learning-based Issue Classification. *2023 2nd International Conference on Paradigm Shifts in Communications Embedded Systems, Machine Learning and Signal Processing (PCEMS)*, 1–6. https://doi.org/10.1109/PCEMS58491.2023.10136114

Sampada, G. C., Sake, T. I., & Chhabra, M. (2020). A Review on Advanced Techniques of Requirement Elicitation and Specification in Software Development Stages. *2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, 215–220. https://doi.org/10.1109/PDGC50313.2020.9315741

Sarkis-Onofre, R., Catalá-López, F., Aromataris, E., & Lockwood, C. (2021). How to properly use the PRISMA Statement. *Systematic Reviews*, *10*(1), 117, s13643-021-01671-z. https://doi.org/10.1186/s13643-021-01671-z

*SDLC Waterfall Model: The 6 phases you need to know about*. (n.d.). Retrieved June 5, 2024, from https://www.rezaid.co.uk/post/sdlc-waterfall-model-the-6-phases-you-need-to-know-about

Shafiq, S., Mashkoor, A., Mayr-Dorn, C., & Egyed, A. (2021). A Literature Review of Using Machine Learning in Software Development Life Cycle Stages. *IEEE Access*, *9*, 140896–140920. https://doi.org/10.1109/ACCESS.2021.3119746

Shaji, L., & R, S. P. (2023). Analysis and study of IDSs for cybersecurity vulnerability detection and prevention: A Comprehensive Review. *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 1–7. https://doi.org/10.1109/ICCCNT56998.2023.10306653

Shankar, S. P., & Chaudhari, S. S. (2023). Framework for the Automation of SDLC Phases using Artificial Intelligence and Machine Learning Techniques. *International Journal on Recent and Innovation Trends in Computing and Communication*, *11*(6 s), Article 6 s. https://doi.org/10.17762/ijritcc.v11i6s.6944

Sharma, N., Sharma, R., & Jindal, N. (2021). Machine Learning and Deep Learning Applications-A Vision. *Global Transitions Proceedings*, *2*(1), 24–28. https://doi.org/10.1016/j.gltp.2021.01.004

Sharma, S., & Chande, S. V. (2023). Optimizing test case prioritization using machine learning algorithms. *Journal of Autonomous Intelligence*, *6*(2), Article 2. https://doi.org/10.32629/jai.v6i2.661

Sharma, S., Sarkar, D., & Gupta, D. (2012). Agile Processes and Methodologies: A Conceptual Study. *4*(05). https://www.researchgate.net/publication/267706023_Agile_Processes_and_Methodologies_A_Conceptual_Study

Singh, S., & Kumar, K. (2023). Software Cost Estimation: A Literature Review and Current Trends. *2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC)*, 469–474. https://doi.org/10.1109/ICSCCC58608.2023.10176495

Smith, A. (2023, February 9). *The Software Developer Shortage and What We Can Do About It*. HatchWorks. https://hatchworks.com/blog/software-development/the-software-developer-shortage-and-what-we-can-do-about-it/

Sofian, H., Yunus, N. A. M., & Ahmad, R. (2022). Systematic Mapping: Artificial Intelligence Techniques in Software Engineering. *IEEE Access*, *10*, 51021–51040. https://doi.org/10.1109/ACCESS.2022.3174115

*Software Developer Shortage in the US*. (2022). Grid Dynamics. https://www.griddynamics.com/blog/software-developer-shortage-us

Wang, P. (2019). On Defining Artificial Intelligence. *Journal of Artificial General Intelligence*, *10*(2), 1–37. https://doi.org/10.2478/jagi-2019-0002

*What Is Strong AI? | IBM*. (2021, October 13). https://www.ibm.com/topics/strong-ai

Worku, A., Alemneh, E., Assefa, Y., Getaneh, T., & Mekonnen, S. (2023). Test Case Generation from Quality Attribute Scenarios Using Machine Learning Approach. *2023 International Conference on Information and Communication Technology for Development for Africa (ICT4DA)*, 31–36. https://doi.org/10.1109/ICT4DA59526.2023.10302184

Yakubu, R. (2022, May 2). *CODING | The Power of AI in Business—Building The Future*. https://www.buildingthefuture.pt/en/explore/2022/coding-the-power-of-ai-in-business/

Yarlagadda, R. T. (2017). AI Automation and it's Future in the UnitedStates. *SSRN Electronic Journal*, *5*, 382.

Zelkowitz, M. V. (1978). *Perspectives on Software Engineering*.
https://www.academia.edu/49026042/Perspectives_in_Software_Engineering

# APPENDIXES

## APPENDIX A

Appendix A presents a transcription of the interview with Expert 1 for each of the predefined questions for the purpose of the framework's evaluation. The content in this appendix represents the expert's opinions after a brief presentation of the framework, where he offers feedback on its usefulness, some detected criticisms, and some recommendations for the framework's improvement.

**Question 1**

*"This matter of having a framework supported by an iterative flow, ends up being important because, by applying any method and having the proper monitorization for each iteration, I can continuously improve the chosen AI algorithms the chosen AI which I will utilize for a given situation. Therefore, I think it is a framework which will support the activity of a project manager (…). Thus, I believe it is important for these kinds of frameworks to come to the fore. It is important that there are researchers working in this direction and doing so through an empirical study or eventually with their application in a real-life situation in order to later assess if eventually the framework that is built here really achieves the defined objectives and will provide the benefits to a project within an organization."*

**Question 2**

*"The criticism that I believe I must refer is the fact that we can consider, or not, at the same time more than an algorithm in each model, within a given phase. It is there, eventually, some complexity if the algorithms have different construction foundations, for example, constructed in different programming languages. In this case the syntaxes might not be properly compatible (…). Anyway, I think that interactions with the same algorithms, if possible, will return more natural results, if that work can be used here. When we use different algorithms, it depends a lot on the way they are constructed and on what is intended. It would be interesting if we had here another element which could help you think in this question (…)."*

**Question 3**

*"(…) So, the framework is more of a proposition, a recommendation, therefore it does not have your recommendation after utilizing some. Maybe you can perform the test in a unitary way, you can suggest this or that algorithm, for example in the case of a problem existing in a company to design a stock management software. So that would be interesting. And maybe, because this project can evolve, do an implementation in different companies of the same sector so you can make a recommendation for that particular field, or in different sectors. And it can have a more generalized recommendation in the application of AI in the organizations. You can bring these questions to light for further work as I believe you have an important framework here (…).*

*At the implementation level I do not know if you have something defined in terms of monitorization after the framework is implemented. A set of ideas or outputs that would support recommendations that would go in accordance with the defined objectives initially in your main document. And that would close, from the beginning to the end, a proposition of a framework for the implementation of AI in the organizations.*

I would say that it is in your interest to interview a project manager because they would have a more holistic vision, not being simply concerned with the development of the code but with the development of the product as a whole (…), so it provides a vision that would have more inputs and recommendations (…)."

## APPENDIX B

Appendix B contains a transcription of the interview with Expert 2, covering each of the three questions for the purpose of the framework's evaluation. Here, the expert's feedback pertaining to each question is presented separately, regarding the framework's usefulness, perceived criticisms and opportunities for its improvement.

**Question 1**

*"About the utility of the proposed framework, it looks to me that it is relevant, as it is important to nowadays improve the productivity of the workers because, as stated, there is an increasing deficit of software developers and one of the ways to battle this scarcity is through giving them tools that permit the already existing professionals do more. This seems to be a good justification for what is being done here which is, trying to find the AI algorithms that can help the professionals in all phases of the development process, from the initial contact with the customer to the maintenance phase. For these reasons I see the utility of this framework (…). The problem that you are interested in exists in practice and, as the algorithms themselves have been evolving, this type of analysis, about which algorithms we can use and in which moments of the development process they can be used, needs to be done."*

**Question 2**

*"One point that does not seem right to me is the definition of the proposition itself. The proposition displayed can be seen as a method, because you propose a set of steps that should be followed to define which algorithms to use in a given moment during the development process. And beyond these guidelines of which algorithms we can use, you also give us the steps of how we can also apply the guidelines, so I suggest that part of the definition of the proposition be reviewed. (…) My proposal is for you to review that part of the definition of the proposition (…) as you are also proposing a utilization process of the framework, for me, the name that seems most appropriate for this proposition is a method."*

**Question 3**

*"You should review that part of the proposition as, for me, the name that seems most appropriate for this proposition is a method. Giving the originality of your work through defining the algorithms that are best for each moment of the development, I would also propose for you to try to add some examples which justify that choice: why an algorithm is better in a given moment than another. This could improve the proposition. This could be an excellent practical contribution. Such as saying, for example, in the moment we are performing test generation we should use LLM, the most known from OpenAI, to generate tests are excellent. In other words, give this kind of feedback about the commercial tools that could be used could be interesting and have a great practical applicability. Another example that could be easily applied is to construct a support chatbot in any application to help in maintenance, to translate NL to translate natural language and direct users to determined solutions or even respond to*

*questions through the transformation of text. And there are other examples that could add more quality to the work."*

## APPENDIX C

Appendix C, such as the previous ones, includes a transcription of Expert's 3 interview, conducted to provide answers to the previously defined questions for the purpose of the framework's evaluation.

**Question 1**

*"To answer the first question, yes, I consider it useful. I think it is a solution that can provide clear utility and can benefit the users that try to apply it."*

**Question 2**

*"That part of the table perhaps is more of a model, a framework would be the workflow where steps are written. This is more a model because this is something that crosses. (…) It is only a matter of the name, it could be called a method, a model or something else, but I would say that the table is a model because it crosses columns with rows. (…) A framework would be the steps, a way of applying something. If you did an application method for each area that would be a framework, but only from that table I wouldn't call it a framework, but the workflows could be called a framework because they have steps. (…) This table came from the literature review. It is a synthesis of it. When something of the sort is normally made the references are normally requested because otherwise people would see this and think that it is something that was made up. The solution is something that was studied and it should show that by including references."*

**Question 3**

*"So, if I understood correctly, this is a mixture of a model applied to a framework, which is the application method. A framework is considered more a set of recipes for others to follow. Independently of the chosen name, I have some considerations for you: I would include the references and give numbers to the articles, it's something that is done quickly and you would enrich the table. It is enough to put only one or two references in each square of the table. If there are many references for each square you could put the most cited or something of the sort.*

*About the framework, I would choose two or three, maybe three, phases and would try to demonstrate an application example. For example, to detect code smells I would give an example by applying a given AI, an AI example for cost estimation, and include the benefits specifically it could bring without even if the case is fictitious. Because what people will draw from this is related to their experiences in their professional lives. Some work in cost estimation (…), and others in requirement engineering, which takes time. Because of this I would give a use case with well explained concrete examples, even if fictitious, it would help others in understanding it better."*