Implication of Artificial Intelligence in Software Development Life Cycle: A state of the art review

P.C. Harish Padmanaban[1], Dr. Yogesh Kumar Sharma[2]

PH.D. Scholar, CSE & IT Dept. Shri Jagdishprasad Jhabarmal Tibrewala University, Vidyanagarari, Jhunjhunu, Rajesthan-333001
PH.D. Coordinator, HOD CSE & IT Dept. Shri Jagdishprasad Jhabarmal Tibrewala University, Vidyanagarari, Jhunjhunu, Rajesthan-333001

***Abstract*—Artificial Intelligence (AI) is the more youthful field in software engineering prepared to acknowledge difficulties. Programming designing (SE) is the commanding mechanical field. Man-made brains strategies, for example, learning based frameworks, neural systems, fluffy rationale and information mining have been upheld by numerous specialists and engineers asthe approach to improve a large number of the product advancement exercises. Similarly as with numerous different controls, programming advancement quality improves with the experience, information of the engineers, past tasks and ability. Programming additionally develops as it works in changing and unpredictable conditions. Henceforth, there is noteworthy potential for utilizing AI for improving all periods of the product advancement life cycle. This paper gives a study on the utilization of AI for programming designing that covers the primary programming advancement stages and AI strategies, for example, common language preparing systems, neural systems, hereditary calculations, fluffy rationale, insect state enhancement, and arranging techniques. Thus, mechanizing SE is the most applicable test today. Man-made intelligence has the ability to enable SE in that manner. Here in this paper we present a best in class writing survey which uncovers the at various times work accomplished for robotizing Software Development Life Cycle (SDLC) utilizing AI.**

Keywords: **Artificial Intelligence, Computational Intelligence, SLDC, Review**

I. INTRODUCTION

There is a planned dinner at your farmhouse. When you will start planning for this? What is the approach that you will follow? Will you be enquiring about the number of guests that are coming to attend the grand party? And you will prepare a delicious menu (Let's say Architecture)? You will start preparing the food and once it is cooked, you will test it, isn't it? (Merely to check if there is a taste in your food). I hope that we have given you a basic idea of what we are trying to explain here, let's jump to our topic. A Software Development Life Cycle(SDLC) is a defined approach and series of steps that are followed for developing any software in order to meet or exceed the set expectation or customer requirements.

Types of Phases in SDLC

There are several phases in a lifecycle of software which is given below:

Requirement Phase:

This is the first and fundamental step in the Life Cycle of Software Development. It starts with gathering the requirements from customers or clients. In most of the organizations, this role is taken care by Business Analysts. A Business Analysts interacts with the customer/clients, set up daily meetings, documents the requirements in Business Requirement Specifications (or Simple Business Specification) and handover the final documented requirement to the development team. It is the responsibility of Business Analysts that every detail is captured and documented and also to make sure that everyone clearly understands the client requirements.

Analysis Phase:

Once the Requirement Gathering phase is completed, the next task is to analyze the requirements and get it approved from the customer/clients. This phase is mainly done by Project Managers, Business Analysts, and Consultants.

Design Phase:

Once the Analysis Phase is over, next comes the need to come up with the most accurate, robust, efficient and cost-effective architecture of the product that needs to be developed. Usually, more than one design is proposed in this phase and the best one is selected based on different parameters such as robustness, durability, timeline, cost-effectiveness, and many more! The different design architecture is generally documented in Design Document Specification or DDS. This phase consists of 2 design approaches:

- Low-Level Design: This task is performed by the Senior Developers where they specify the function of each module of the product architecture that has to be developed.
- High-Level Design: This task is performed by Architects/Senior Architects where they design

different possible architectures of the product that has to be developed.

Development Phase:

This phase is where the actual implementation of programming languages and different frameworks is being utilized for the development of the product. In this phase, all developers are involved. Developers are expected to follow certain predefined coding standards and guidelines; they are expected to complete the project modules within the defined deadline for the project. This phase is also the longest and one of the most critical phases in the Software Development Life Cycle. This phase is documented as a Source Code Document (SCD).

Testing Phase:

Once the Development phase is completed, the next step is to test the developed software. The developed software is sent to the testing team where they conduct different types of testing thoroughly on the software and look for defects. If any defect is found, the testing team records and document which is again sent back to the development team for error removal. This role is taken care of by Software Testers and Quality Analysts of the company. The testing team has to make sure that each component of the software is error free and it works as expected.

Deployment and Maintenance Phase:

After the testing phase is over, the first version of the software is deployed and delivered to the customer for their use. Once the customer starts using the developed software, there is the scope of bug fixing that was not detected during testing phase as when a large group of end users starts using the software, there could be some probability that few boundary cases might have been missed. There is also scope for upgrading the software with newer versions and latest security patches and technologies. And finally, there is also scope for enhancement of the software by adding more features into the existing software.

II. POPULAR SDLC MODELS:

There are many different SDLC Models that are designed for implementing in the software development process. The most important and popular ones are:

Waterfall Model:

In the waterfall model, the whole process of the Software Development is divided into phases where the output of one phase acts as the input to the next phase. The next phase begins only when the previous phase gets completed.

Iterative Model:

This model starts with a smaller set of requirements and it does not need the full context of product specification in order to start the SDLC process. This process is repetitive and on each iteration of the SDLC process, a newer version of the software is made. Each iteration may between 2-6 weeks. Each iteration develops a separate component in this approach. This model also requires a mode resource than the waterfall model.

Spiral Model:

This model is a combination of a Waterfall and Spiral model and it works in an iterative manner. Based on the risk involved in the project, this model guides the team to adopt elements of one or more SDLC models such as a waterfall or Iterative model. Here the lifecycle of Software is divided into smaller parts and new functionality can be added to the software even at the late stages of SDLC.

V-Model:

V model is basically an expansion to the waterfall model where the testing and the development phases are planned in a parallel. One side consists of the verification phase while the other one consists of the validation phase which is finally joined by coding. The next state starts only when the previous state gets completed.

The controls of man-made brainpower and programming building have grown independently. There isn't much trade of research results between them. Computer based intelligence inquire about procedures make it conceivable to see, reason and act. Research in programming designing is worried about supporting architects to grew better programming in less period. Rech and Altoff(2008) state "The controls of computerized brains and programming designing have numerous shared characteristics. Both arrangement with demonstrating certifiable articles from this present reality like business process, master learning, or procedure models."

Today a few research headings of the two controls come nearer together and are starting to assemble new research zones. Encompassing knowledge (AmI) another exploration region for disseminated, non-meddlesome, and smart programming framework both from the course of how to manufacture these framework just as how to planned the coordinated effort between frameworks.

In conclusion computational insight (CI) assumes a significant job in research about programming investigation or undertaking the executives just as information revelation in AI or databases [21].

Man-made brainpower methods, which mean to make programming frameworks that display some type of human insight, have been utilized to help or mechanize the exercises in programming designing. Programming investigations are been connected with incredible accomplishment to distinguish abandons in various types of programming reports

III. UTILIZATION OF AI IN PLANNING & PROJECT EFFORT ESTIMATION

Good undertaking arranging includes numerous viewpoints: staff should be doled out to errands such that assesses their experience and capacity, the conditions between assignments should be resolved, times of undertakings should be evaluated such that meets the venture consummation date and the task plan will unavoidably require update as it advances. Simulated intelligence has been proposed for most periods of arranging programming improvement ventures, including surveying plausibility, estimation of expense and asset necessities, chance appraisal and planning. This segment gives pointers to a portion of the proposed employments of learning based

frameworks, hereditary calculations, neural systems and case based thinking, in venture arranging and abridges their adequacy.

Thus, different proposition that expect to use a KBS approach for venture the board, for example, the utilization of generation standards and cooperative systems (Boardman and Marshall, 1990), which appeared to be encouraging at the time have not been generally embraced. When thinking about whether to embrace a KBS approach, the expense of speaking to the learning appears to be high and except if this should be possible at a degree of deliberation that permits reuse, one can envision that it is ugly to programming engineers who are sharp and compelled to begin their activities immediately. Neural Networks Neural systems (NNs) have been broadly and effectively utilized for issues that require grouping given some prescient info highlights. They consequently appear to be perfect for circumstances in programming building where one needs to foresee results, for example, the dangers related with modules in programming support (Khoshgoftaar and Lanning, 1995), programming hazard examination (Neumann, 2002) and for anticipating flaws utilizing item situated measurements They recognized a sum of 39 hazard elements which they assembled into 5 chance classifications: venture intricacy, participation, cooperation, venture the executives, and programming building. These were diminished to 19 directly autonomous elements utilizing head segment investigation (PCA). The reasons for these progressions may change from the expanding comprehension of the client about the capacities of a PC framework to some unexpected hierarchical or natural weights. On the off chance that the progressions are not obliged, the first necessities set will wind up fragmented and conflicting with the new circumstance or in the most pessimistic scenario pointless (Meziane, 1994).

There are correspondence issues between the partners: During the necessities building stage, engineers need to converse with a wide scope of partners with various foundations, interests, and individual objectives (Zave, 1997). Correspondence with and seeing every one of these partners is an amazingly troublesome and testing task. Necessities are hard to oversee: One of the principle issues related with prerequisites is that of recognizability. Recognizability is the way toward following a necessity from its elicitation to usage and confirmation and approval. Connecting the various periods of prerequisites approval is regularly discarded. Other administration issues identified with programming the executives are: venture the executives, programming cost, advancement time, assets the board and dealing with the evolving condition. The primary commitment of AI in the prerequisites building stage are in the accompanying zones:

- Disambiguating characteristic language prerequisites by creating devices that endeavor to comprehends the regular language necessities and change them into less equivocal portrayals.
- Developing information based frameworks and ontologies to deal with the necessities and model issue areas.
- The utilization of computational insight to 283 Artificial Intelligence in Software Engineering take care of a portion of the issues related with prerequisites, for example, inadequacy and prioritization.

In the accompanying segments, we survey and talk about a portion of the frameworks created in these territories [22].

Restrictions

The principle constraints of our examination are the single-case use, little example size of specialists and the likelihood of assumption in information gathering and investigation from poll. The way that we utilized a solitary case all encompassing structure makes us progressively helpless to predisposition and dispenses with the likelihood of direct replication or the examination of differentiating circumstances. Accordingly, the general reactions about singlecase contemplates, for example, uniqueness and exceptional access to key witnesses, may likewise apply to our examination. Our objective was not to give measurable speculations about a populace based on information gathered from an example of that populace. Another constraint is that a piece of our assessment depends on semi-organized poll. The pragmatic assessment from industry is additionally needs behind for the conceivable proof of our structure.

IV. DOCUMENTING SOFTWARE REQUIREMENTS TO AI

And here comes the biggest difference between AI and traditional algorithmic programming…

On the one hand, in software development traditionally you documented your data input algorithm that would transform the input into expected results. An algorithm fed with specific input would always lead to a precise result with 100% certainty.

On the other hand, when developing AI you provide data and expected results. The outcome of software development is a neural network and its configuration models. Such a couple with a specific probability shall turn input data into expected results. That is the classical heuristic approach; providing a specific data input will result in specific data output, but within limitations of probability e.g. 80% or 90%.

From the above two approaches we can derive a different strategy to documenting intended use and software requirements. You should consider the specifics of an AI software component and firstly reflect on its intended use. Using an AI component for “advisory”, “data processing” services presents no difficulties. Additionally, you can address the “result probability challenge” with having the outcomes checked by humans or controlled by deterministic software component.

Whatever solution you design the software requirements should reflect this intended use. Never assume 100% accuracy of the results when specifying your software requirement. Instead, define the anticipated accuracy of your health software AI in terms of probability. Please note that humans quite seldom make diagnosis with 100% probability either so the regulator should not have any issues with that.

Some other software requirements worth adding at this moment are, for example: protecting your neural network models with cryptographic controls, identification of your AI version used for data processing, always sending the results together with the AI version from the AI software unit, monitoring, proper initialization of AI etc.

V. SELECT PROPER TECHNOLOGY STACK FOR AI

The first thing to do is to correctly select the best Technology Stack for developing and hosting your AI software unit.

Development Language – you should decide what language for developing AI's network topology will be used. For example, on the one hand, if you decide to select one of scripting languages your development may be faster but protecting your AI configuration and models within a production environment will be trickier. On the other hand, having the final AI network and configuration working on a component compiled to a binary executable will make your solution firstly much safer and secondly more efficient as well.

Training Environment – this is the second decision to consider. Where are you going to train your neural networks? You have a wide range of options here starting from cloud provided GPU units e.g. Amazon EC2 Elastic GPU to dedicated hardware for that purpose e.g. Razer CORE v2. In Pro4People, we usually use our own hardware graphical cards for neural network design, training and optimizing. As a result, we can bring the initial software development costs down. Additionally, we can achieve higher security in the software development phase by limiting that to our inside office perimeter.

Already within that phase you should have brought all the cryptographic controls into the game to protect your AI models. Additionally, you can think about patenting your proprietary network configurations / models.

Production Environment – finally, you have to decide on what infrastructure your AI will be operating. We usually see here that the AI is deployed to cloud infrastructure like Amazon EC2 Elastic GPUs since it offers your solution horizontal scalability options. As such services usually come at a quite hefty price, the proper architecture (pay-as-you-go approach) will help you to keep costs at bay [28].

AI Design Training and Optimizing

The most important part of AI software development is designing, training, and optimizing. Within that phase you will have to select or design a neural network which is the most suitable for the problem you are going to tackle.

Network Topology – What kind of network will work the best for the problem you want to tackle to offer new value proposition? Will you go for any of the well-known classes like e.g. Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) or Long Short-term Memory Networks (LSTM)? Does your problem require developing more sophisticated topologies (e.g. ZF Net, VGG, or ResNet), or even your own proprietary neural network topology? If you have no idea how to start that, do not worry -check this out: https://pro4people.com/ai-powered-solutions/

Data Curation – This is even more important as your network configuration. The goal of this step is to prepare data that will be used to train your neural network. You should have both the input data and the expected results prepared. The more reliable your data set is, the higher the chances of training and optimizing the AI model that will fit your needs. Please, protect your data as well. Having your data set and the expected outcomes a competitor can come up with their own AI configuration. You will also need that data later for Verification and Validation of your medical device. It will be also used in clinical trials.

Training / Optimizing – At this stage you have your chosen network topology and data ready for training. Now it is time to train your network, optimize it, and keep repeating this process over and over again until you fall within the probability expectations specified in your software requirements. It takes time, computing power and knowledge to train your network, polishing learning data, modifying neural network topology, and re-evaluating learning processes. In order to cut costs in this phase we usually do the training on physical HW in our company or in a cloud, including powerful machines supported by GPUs. Please remember that not all topologies have a support in GPU optimization and it also gives you an opportunity to optimize computation beyond the utilized AI framework.

Integration – The integration is a very important stage of your development. You have to integrate your AI software unit with the rest of the system architecture. The interfaces should be precisely specified, as quite likely your AI will be released quite often. Thanks to carefully specified interfaces your AI can be both forward and backward compatible from a product life cycle perspective. Equally important are the questions of scalability. Operating a EC2 Elastic GPU comes with a hefty price tag. If you design and integrate your solution with horizontal scaling, you can save costs and take advantage of a pay-as-you-go approach [31].

Software Unit Identification

Well, we all know the identification focus when developing software for medical devices, don't we? AI does not differ here at all. Consider closing the AI component as a separate Software Unit in your system architecture. This way you can benefit from its quite likely more frequent releases of new, better trained neural network in the future. Such a component should always identify its version and the models applied. It should be enough to identify the configuration used in making decision / processing health data in your system. Please, remember to store this version together with the processing result so you will always be able to say which version of your health software turned data into information / result.

Verification & Validation (V&V)

When considering AI software unit verification and validation approach it does not differ from testing any other software unit. The typical configuration of test cases could be:

1. AI software Unit Test Level
2. Integration Test level

3. User Acceptance Tests – in the scope specific to software requirements specific to AI

The verification part of testing shall be covered within your R&D environment as a part of Software Development Life Cycle. The validation, as it shall be executed in production environment, usually can be postponed to version deployment and its validation. It is a good practice, to separate AI Software Unit Test Level as a separate automatic testing component. Then, running this test level can be done on any environment even in the continuous manner. Please, note that quite likely your AI may be a subject / part of your clinical studies in case they are required for your medical device software.

VI. CONCLUSION AND FUTURE WORK

This investigation gives an essential in general incorporating model that represents the key idea of coordination between AI procedures and deft programming advancement ventures. It must be referenced that the model displayed in this paper is as yet improved in on-going examination contemplates is as yet subject to assist refinement. In this postulation an incorporated structure an Agile practice with AI strategy is proposed. The real zone of commitment in this system is to extend and upgrade the spry programming improvement life cycle. This system is increasingly doable for the undertakings in which necessities and its answer are reused all through the improvement cycle. It has the ability to manage pretty much every sort and size of task i.e., little, medium and enormous size ventures. Fundamental commitments are:

- This will support the engineers and partners to have clear vision of situations and perspectives on client necessities.
- This will stick the designers and clients all through the improvement cycle and this will build the certainty of clients.
- Stakeholders and uniquely clients can get clear pictures of what sort of item these prerequisites will shape, so they can change at any stage.
- This will concentrate more on individuals and correspondence against procedure and documentation.

Future research will concentrate on progressively explicit to cross breed models to get top to bottom comprehension and give total structure. Besides, a study might be directed to inspire basic data about the manners by which industry tailors programming practices and strategies, and the similarity and viability of half breed programming approachs. Moreover, AI systems alongside the intercommunity between, traditional SE and dexterous strategies can be significant research course.

VII. REFERENCES

[1]. Zave P(1997) Classification of Research Efforts in Requirements Engineering. ACM Computing Surveys 29: 315-321.

[2]. Pressman RS (2001) Software Engineering. McGraw Hill, USA.

[3]. Cho J(2008) Issues and Challenges of Agile Software Development with Scrum. Issues in Information Systems 9: 188-195.

[4]. Cho J (2009) A Hybrid Software Development Method For Large-Scale Projects: Rational Unified Process With Scrum. Issues in Information Systems 10: 340-348.

[5]. Rachel H (2012) Report from the first international workshop on realizing artificial intelligence synergies in software engineering. ACM SIGSOFT Software Engineering Notes 35: 34-35.

[6]. Beck K, Beedle M,van Bennekum A, Cockburn A, Cunningham W,et al. (2001) Manifesto forAgile Software Development. United States of America.

[7]. Cockburn A (2002) Agile software development. Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA.

[8]. Pressman RS (2010) Software Engineering. McGraw Hill, USA.

[9]. Jiang L, Eberlein A (2008)Towards a framework for understanding the relationships between classical software engineering and agile methodologies. International Conference on Software Engineering, Leipzig, Germany.

[10]. Nuseibeh B, Easterbrook S (2000) Requirements Engineering: A Roadmap. ICSE '00 Proceedings of the Conference on the Future of Software Engineering, Limerick, NY, USA.

[11]. Prince J (2011) Interaction between Software Engineering and Artificial Intelligence- A Review. International Journal on Computer Science and Engineering 3: 3774-3779.

[12]. Al-masum SM, Morshed AM, Mitsuru I (2010) Object Oriented Hybrid Software Engineering Process (SEP) model for Small Scale Software Development Firms.

[13]. Abrahamsson P, Warsta J, Siponen MT (2003) New Directions on Agile Methods: A Comparative Analysis. Proceedings of the 25th International Conference on Software Engineering, Portland, OR, USA.

[14]. Poppendieck T, Poppendieck M (2003) Lean Software Development: An Agile Toolkit.Addison-Wesley, Boston, Massachusetts, United States.

[15]. Jun L, Qiuzhen W, Lin G(2010) Application of Agile Requirement Engineering in Modest-sized Information Systems Development. Second WRI World Congress on Software Engineering, Hubei, Wuhan, China.

[16]. Paetsch F, Eberlein A, Maurer F (2003) Requirements Engineering and Agile Software Development. Twelfth IEEE International workshop, IEEE Computer Society Washington, DC, USA.

[17]. Zaigham M, Qureshi MRJ (2012)Novel Hybrid Model: Integrating Scrum and XP. International

Journal of Information Technology and Computer Science 6: 39-44.

[18]. Marchesi M, Mannaro K, Uras S, Locci M(2007) Distributed Scrum in Research Project Management, Agile Processes in Software Engineering and Extreme Programming, 8th International Conference, Italy.

[19]. Dingsøyr T, Hanssen G, Dyba T, Anker G, Nygaard J (2006) Developing software with scrum in a small cross-organizational project. European Conference on Software Process Improvement,13th European Conference, Finland.

[20]. Meziane F, Vadera S (2012) Artificial Intelligence in Software Engineering: Current Developments and Future Prospects. IGI Global Disseminator of Knowledge.

[21]. Dr. Yogesh Kumar Sharma and Dr. Surender (2013), "A Comparative Performance Study of Bluetooth and Zigbee Protocols", "Research Reformer – International Referred Online Research Journal", ISSN-2319-6904, Issue No. XI, Pp. 3-17

[22]. Dr. Yogesh Kumar Sharma (2018), "Framework for Privacy Preserving Classification in Data Mining", "Journal of Emerging Technologies and Innovative Research", ISSN: 2349-5162, Vol. 5, Issue 9, Pp. 178-183.

[23]. Rech J, Althoff KD (2004) Artificial intelligence (AI) and software Engineering (SE): Status and future trends. KI 18: 5-11.

[24]. Jeff S, Anton V, Jack B, Nikolai N (2015) Distributed Scrum: Agile Project Management with Outsourced Development Teams. 40th Annual Hawaii International Conference, Waikoloa, HI, USA.

[25]. Shi Y, Wang H, Tang J (2011) A case-based reasoning system for fault management in CDMA network. In Control and Decision Conference, China.

[26]. Ammar H, Abdelmoez W, Hamdi MS (2012) Software Engineering Using Artificial Intelligence Techniques: Current State and Open Problems. Institute of Communication, Culture, Information and Technology, University of Toronto Mississauga.

[27]. Batool A, Hafeez YM, Hamid B (2013) Comparative Study of Traditional Requirement Engineering and Agile Requirement Engineering. 15th International Conference on Advanced Communications Technology, SungnamKyunggi-Do, Korea.

[28]. Seth B., Dalal S., Kumar R. (2019) Hybrid Homomorphic Encryption Scheme for Secure Cloud Data Storage. In: Kumar R., Wiil U. (eds) Recent Advances in Computational Intelligence. Studies in Computational Intelligence, vol 823. Springer, Cham

[29]. Vlaanderen K, Jansen S, Brinkkemper S, Jaspers E(2011) The agile requirements refinery: Applying SCRUM principles management. Information Technology 53: 58-70.

[30].

[31]. Dac-Nhuong Le, Bijeta Seth and Surjeet Dalal, A Hybrid Approach of Secret Sharing with Fragmentation and Encryption in Cloud Environment for Securing Outsourced Medical Database: A Revolutionary Approach, Journal of Cyber Security and Mobility Vol: 7 Issue: 4 2018, Page: 379-408 doi: https://doi.org/10.13052/jcsm2245-1439.742

[32]. Hayata T, Han J (2009) A hybrid model for IT project with Scrum.IEEE International Conference onService Operations, Logistics, and Informatics (SOLI).

