JOURNAL OF CRITICAL REVIEWS

ISSN- 2394-5125 VOL 08, ISSUE 03, 2021

DESIGNING SOFTWARE WITH ARTIFICIAL INTELLIGENCE TO TEST IT AGAINST SRS

Sunny Arora[1], Vishal Kumar[2]

[1,2]Guru Kashi University, Talwandi Sabo

Abstract

SDLC is the interaction that shows how to make, plan, and keep up with a software project so that all the functional and client needs, goals, and destinations are met. AI is the newest field in computer science that is ready to meet people's needs. Software design (SE) is a huge field in the modern world. The most important thing to do now is to make SE easier to do. Computer-based intelligence can help SE in that way. In this paper, we show a cutting-edge writing audit that shows how AI can be used to automate the Software Development Life Cycle (SDLC) at different times. SDLC is a way of working together for a software project, inside of a software group. It shows how to make, keep up with, replace, and change or upgrade specific software in a specific way. It means to be the standard for all of the things that have to be done to make and keep up with software. It is the way people work together that shows how to make, plan, and keep up with the software. These strategies work on the nature of the software task and the way that software is made and changed.

KEYWORDS: DESIGNING SOFTWARE, ARTIFICIAL INTELLIGENCE, SRS

1. Introduction

As software frameworks get bigger and more complicated, there are a lot of different problems with them. Many different steps must be taken before a piece of software can be shared, including gathering requirements, planning the project, writing the code and putting it through tests. In each of these stages, there are different tasks or exercises. Because software is so big and complicated, these software design jobs have become more and more expensive and more likely to go wrong. As a result, there is a need to look into computationally smart ways to do different software design jobs. Using heuristic calculations that are planned and used to get good results in a reasonable amount of time is called computational insight. This is how people think about things. Calculations like these have been used in a lot of different fields, like medical science, bioinformatics, computer organisations (to direct and book), and estimating. So, too, analysts have used careful methods in other areas of software design, like software requirement prioritisation, software cost assessment, dependability assessment and so on. These include software deformity and viability forecasts as well as software weakness and size forecasts. There are a lot of smart ways to use computational strategies, like AI nears, meta-heuristic calculations, and improvement plans: Advancement calculations can be used to get an answer to a question where the goals or focuses to be reached are known. AI calculations are used when we have enough information so that information can be removed and models can be made. For example, models can be made to predict which types of software are likely to make mistakes. A meta-heuristic is a high-level, iterative interaction that helps and controls a basic heuristic to do a good job of exploring the question space. The hidden heuristic can be a search in your neighbourhood, or a low or high level method. Because meta-heuristics use the space in which they look, they can make close-to-ideal arrangements with high accuracy and limited resources in a reasonable amount of time. To write a book like this, scientists, academics, and experts will be able to write about how they think and research how to use astute methods in the field of software design.

1.1. Software Requirements Specifications

Software Requirements Specifications (SRS) is a document that lays out what a software framework should be able to do and how it should work. It also lists any rules that the framework should follow. By definition, a necessity is a goal that must be reached, while a detail shows how the goal will be reached. At the end of the day, a detailed record shows how to do specific tasks. An important part of the job of making sure that the system meets its needs is to be proactive in getting to know the people who will be working on it. There used to be a few people who didn't think that organisations should pay less to fix problems that were found right away in any Software Development Life Cycle (SDLC). The main point of this study is the requirements determination stage of the SDLC, because if mistakes are found at this point, then the cost of fixing them will be less. The Software Quality Assurance (SQA) review method is used to look into the nature of the SRS. This method is used to see if the necessary principles in the requirements-determination stage are being followed very closely. The main goal of a quality check of the SRS is to make sure that the software requirements, among other things, are complete, reliable, right, modifiable, positioned, detectable, unambiguous, and reasonable. CBR is a man-made brainpower process that thinks by remembering and reviewing things that have happened in the past. It is very similar to how people think when they face new problems. It also talks about how people come up with new solutions for problems they've recently encountered based on their past experiences and learn new skills at the same time. This method will speed up the SRS quality investigation process. Because many tasks that need to be done are very similar to tasks that have been done in the past and the answers for these similar tasks can be found in an information base or case base that has all of the past issues and arrangements. Retrieve, Reuse, Revise, and Keep are the four main steps in a CBR cycle.

1.2.Utilization of AI in Software Planning and Requirement Analysis:-

Prerequisite research is the most important and important step in the SDLC process. It is done by the senior members of the group with help from the client, the outreach group, market studies, and business people in the area. This information is then used to design the basic business plan and to look into how useful the product will be in the affordable, functional, and specialised areas. In the planning stage, you also have to think about things like how to meet quality standards and how to show that the project is risky.

- **Defining the Structure**
  - **Reason/Introduction**
    The reason area should sum up the whole SRS archive, so it should be there as well You can think of it as a summation of business records for the leader. It gives the task a sense of what to expect. Definitions, frameworks, and references are often important parts of this section. These help to build up important parts of the project.
  - **Generally speaking Description**

The general picture gives a general idea of the rules and different subsections. To show the specific requirements better, go to the "Specific Requirements." The general picture's job is to think about things that might change the requirements.

The parts of the general picture are the item's point of view, how the item works, what the customer can and can't do, assumptions, and conditions. These all have to do with figuring out what will happen and how to deal with it instead of how to meet the requirements. Plan restrictions, for example, include everything from making sure software is compatible to making sure the equipment you need is available.

- **Specific Requirements**

People write down all of the things that need to be done to improve in this section. This part helps people figure out how to make the item so that it fits their needs.

Among other things, you'll find out about things like how the project will work together and how the project will interact with other things. The "specific requirements" section is where you'll find this information. It is important for the whole programme to work for each of these subsections to talk about a lot of things that are important for that.

- **Making an Exceptional SRS**

Now you know how to make a good SRS archive. A quick search will show you a lot of different ways you can use this new knowledge if you aren't 100% sure about your new skills yet. SRS is what you want to improve, so hit it right away. It's important to remember that the goal of this document is to make programme development go as smoothly as possible, not to have the best SRS ever. It is one of the most important things we talked about that your SRS should be able to change with the needs of the project. There is a lot of information here, so it might seem like a lot to read at once. That's because it is. This article gives a clear picture of a very complicated process. The best way to move forward with your SRS research is to think of it as a way to explain all of your projects to your partners in simple chunks of information. Accept it piece by piece as you move through each part of the archive. You'll be grateful to yourself for taking the time to learn more about your next improvement project. As with everything else, practise will make your SRS more stable. However, these rules, traits, and design ideas are a good start.

2. ARTIFICIAL INTELLIGENCE IN SOFTWARE ENGINEERING

In software design, rules for how to make and make software are shown in a way that makes sense. Software design is the most important part of managing the development of software. The process of making software better is a long one that goes through a lot of different stages and needs executable code. In the past, people have been writing code for how software works. There isn't a machine that can do this better yet. AI is the process of making smart machines that can do things that people do. Artificial intelligence can be used to help with a lot of software development projects. Because of this, there is a lot of room for working on all stages of the SDLC (Software development life cycle).

- **SRS**

Software Requirement Specification (SRS) Format, as the name implies, is a complete description and specification of the needs of software that must be met for the development of a software framework to be successful. These requirements can be as useful as non-requirements that depend on some kind of need. Clients and a worker for hire work together because it's important to fully understand the needs of clients..

- **Role of AI Techniques in Software Development Activities**

The Way We Make Things

In this case, AI strategies will focus on the tasks of requirements investigation, engineering configuration, coding, and testing, among other things. This is what Hany H Ammar et al said in 2010.

1.) Software requirements analysis.

First, requirements are written down in normal language in a set of archives. Then, the requirements are written down in a computer programme. These archives usually deal with "unsettled views on a group of people." They will, in most cases, be "fragmentary, conflicting, disconnected, not focused on, and often exaggerated, above and beyond what is necessary." The main activities of this stage are gathering requirements, organising events, and transforming them into a less vague representation, which is what this stage is all about. (Young,2003)

There are a lot of problems that come up at this point.

- Requirements are unclear.
- Requirements are fragmented, dubious and unclear.
- They also don't work together.
- It's hard to keep track of the requirements because they change all the time.
- There are communication problems between the partners.

In the next area, some of the tech is shown.

2.) Processing Natural Language Requirements NLR

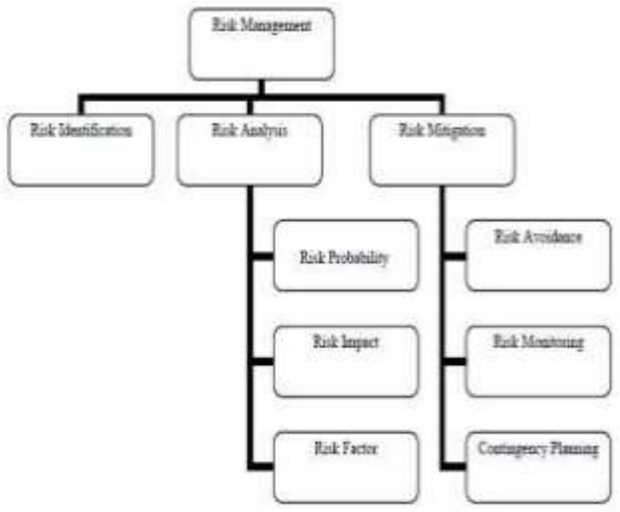
A structure for translating NL (English) specifications into formal specifications (TELL) was shown, but the structure was not used. It set the groundwork for future systems. When you translate NL sentences, you need to keep in mind the express properties of a receptive framework. Then, you need to explain an activity-based transient reason. Another framework, the FORSEN framework, was made. Its main goal was to translate NL requirements into the formal specifications language VDM, which is used for formal specifications. This framework allowed us to see if there were any ambiguities in the NL requirements. Saki and other people did this study in 1989.

Some of the examples of frameworks that have tried to make OO models from NL requirements are shown below: It was written by Fentechi in 1994.

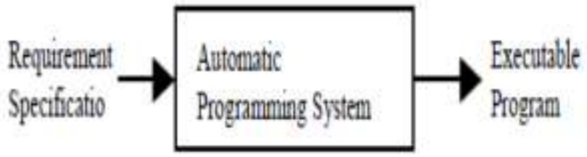
An all-encompassing system for making OO models from NL requirements with phonetics instruments was shown. The Large-scale Object-based Linguistic Interactor Translator Analyzer (LOLITA) NLP framework was used to help with the NL-OOPS, which means that to make OO specifications from NL requirements, a method was developed that used a lot of phonetic examples to connect the phonetic world and the reasonable world. This method was called "phonetic examples." Another framework was made called the Class-Model Builder (CM-Builder), which is an NL-based CASE tool that makes class graphs in UML from NL requirements documents. (Mezaine,1994)

3.) RISK MANAGEMENT

The Risk Management measure is a way to figure out risks before they happen and figure out how to avoid or lessen the effects of those risks if they do happen. In the beginning of the software development life cycle, the executives face a lot of risk. However, the real work of managing risks happens during the development of the product. It shows how the executives measure the risk they face in the figure shown in the text. Officially, the three stages of the executives' interaction with the risk are:



AI-based systems are free from risk management techniques because of automated programming methods, which make information structures more flexible and more able to be changed quickly. This is the age of computer projects that are written by a computer, and they are often written in languages that are easier for people to understand than traditional programming languages.



Making the specification smaller and easier to write is the goal. It will also be clearer and less likely to make mistakes than programming languages.

a. Software architecture design

Probably, the main problem for the software engineer is to make sure that the requirements model is used to make good designs. In this part, a picture of how AI will be used in the future to help with software engineering configuration is shown. The first step in designing software is to figure out the order of subsystems and parts with specific responsibilities from the data provided by the requirements and testing models. AI methods use quality properties to figure out how well a work of integrity is done over the space of possible models. Seclusion, complexity, changeability, understandability (or clearness), and reusability are some of the most well-known quality properties of engineering configuration that are used to help design. Seclusion is usually linked to the idea of coupling and union, which is when engineers try to make a well-thought-out plan by using subsystems and parts that aren't connected very well but are very strong. When Robyn Lutz was working on AI-based software design, she used Genetic Algorithms (GAs) to look through the space of possible future breakdowns of a system. A health project that uses data hypothetical measurement to track the information and control coupling between parts. The quality property used in the wellness work is related to how detailed and specific the design is that was made. After that, the main focus was on Product Line Architectures (PLAs), which are designed to make it easier to reuse and change the design of a reference design that can be used to start a family of models.

Constraint writing computer programmes is another AI process that is used when making software. People have used constraints programming to plan the PTIDEJ framework, for example. This is an example of how constraint programming can be used.

PTIDEJ is a computer programme that helps you find microarchitectures that look like configuration designs in object-oriented source code. Miniature engineering is a way to describe a group of classes in a protested programme. Most of the time, PTIDEJ wants to be able to explain what it says in its replies. This is very interesting because coding and software design is usually thought of as a kind of art, and completely automated systems aren't usually valued by people who want to buy them (or developers).

Search-Based Software Engineering (SBSE) is a new research field that focuses on looking at parts of Software Designing as problems that can be solved with metaheuristic search calculations made by AI. SBSE is the process of rethinking software design tasks as "streamlining issues." Hereditary calculations can be used to help people grow and find jobs. To figure out how old a computer's programme code is, hereditary calculations are used. They do this by making a group of possible answers for a problem. Those who live in the world are computer programmes.

b) Testing

During the turn of events, software testing is still a costly job, and one of the main problems is that it can't be automated. AI procedures can play an important role in this way. One of these methods is called constraint addressing methods. In the field of transformation testing, a lot of attention has been paid to the use of constraint-solving methods in the computerization of software testing (Constraint-based testing). In the example of ATGen, for example, a software test information generator that relies on representative execution and constraint rationale programming for ADA programmes is called ATGen. There are many ways AI strategies can help keep the testing process going. One of the first studies to propose using information-based frameworks for testing, this study shows a Prolog-based master framework that takes a COBOL programme as information and then tries to figure out what important conditions are. It then tries to make test information based on the conditions.

Fuzzy reasoning is another AI method that is used in software testing to deal with the uncertainty that comes with this stage of software development. In 2007 (Nand),

4 Conclusion

A software item can be made with the help of software designing, but by following the software designing standards, it takes a lot of time to make changes to it. AI can be used in software development to improve the nature of the item. With the help of a computerised device or a computerised programming tool, we can cut out the risk evaluation stage, saving us time and making a powerful product. As a result of artificial methods in Software Engineering, we can cut down on the time it takes to make software. Coding can be turned into Genetic Code, which is the main thing to think about when you're working with software to improve it

.

5. References

[1] Abbott, R. J. (1983). Program design by informal English descriptions. CACM, 26(11), 882–894.

[2] Bering, C. A., & Crawford, M. W. (1988). Using an expert system to test a logistics information system. In Proceedings of the IEEE National Aerospace and Electronics Conference (pp. 1363- 1368), Dayton, OH. Washington DC: IEEE Computer Society

[3] Coulin, C., Zowghi, D., & Sahraoui, A. (2010). MUSTER: A Situational Tool for Requirements Elicitation. In F. Meziane, & S. Vadera (Eds.), Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects (pp. 146-165).

[4] Falbo, R. A., Guizzardi, G., Natali, A. C., Bertollo, G., Ruy, F. F., & Mian, P. G. (2002), Towards semantic software engineering environments. Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering, (pp. 477-478).

[5] Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M., & Moreschini, P. (1994). Assisting requirement formalization by means of natural language translation. Formal Methods in System Design, 4(3), 243–263.

[6] Gupta, M., Bastani, F., Khan, L., & Yen, I.-L. (2004). Automated test data generation using MEA-graph planning. In Proceedings of the Sixteenth IEEE Conference on Tools with Artificial Intelligence (pp. 174-182). Washington, DC: IEEE Computer Society.

[7] Harmain, H. M., & Gaizauskas, R. (2003). CM-Builder: A natural language-based CASE tool for object-oriented analysis. Automated Software Engineering Journal, 10(2), 157–181.

[8] Hewett, Micheal, and Rattikorn Hewett (1994). 1994 IEEE 10th Conference on Artificial Intelligence for Applications.

[9] Howe, A. E., von Mayrhauser, A., & Mraz, R. T. (1995). Test sequences as plans: an experiment in using an AI planner to generate system tests. In Proceedings of the Tenth Conference on Knowledge-Based Software Engineering (pp. 184-191).

[10] Hull, E., Jackson, K., & Dick, J. (2005). Requirements Engineering. Berlin: Springer.

[11] Juristo, N., Moreno, A. M., & López, M. (2000). How to use linguistics instruments for Object-Oriented Analysis. IEEE Software, (May/June): 80–89..

[12] Kof, L. (2010). From Textual Scenarios to Message Sequence Charts. In F. Meziane, & S. Vadera (Eds.), Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects (pp. 83-105).

[13] Lubars, M. D., & Harandi, M. T. (1987). Knowledge- based software design using design schemas. In Proceedings of the 9th international Conference on Software Engineering, (pp. 253-262).

[14] Lutz, R. "Evolving good hierarchical decompositions of complex systems," Journal of Systems Architecture 47 (2001), 613–634.

[15] Lutz, R. "A Survey of Product-Line Verification and Validation Techniques," JPL-NASA Technical Report, 2007 http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/41221/1/07- 2165.pdf

[16] Jing (Janet) J. Liu, Samik Basu and Robyn R. Lutz: Generating Variation Point Obligations for Compositional Model Checking of Software Product Lines. Journal of Automated Software Engineering, p. 29, vol. 18, 2011

[17] Memon, A. M., Pollack, M. E., & Soffa, M. L. (1999). Using a Goal Driven Approach to Generate Test Cases for GUIs. In Proceedings of the Twenty-first International Conference on Software Engineering (pp. 257-266).

[18] Meziane, F. (1994). From English to Formal Specifications. PhD Thesis, University of Salford, UK.

[19] Meziane, F. and Vadera, S., (2010). Artificial Intelligence in Software Engineering Current Developments and Future Prospects, In "Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects", IGI Global

[20] Mich, L. (1996). NL-OOPS: from natural language to object, oriented requirements using the natural language processing system LOLITA. Natural Language Engineering, 2(2), 161–187