HILL PUBLISHING

Future Trends in AI Research, 2024, 1(1), 27-33
https://www.hillpublisher.com/Journals/ftair/
ISSN Online: 3066-3083

# Research on Automated Software Development Processes Based on Artificial Intelligence Algorithms

**Yue Hou*, Jing Liu**

Meiga Polytechnic Institute of Hubei, Xiaogan 432017, Hubei, China.

**How to cite this paper:** Yue Hou, Jing Liu. (2024) Research on Automated Software Development Processes Based on Artificial Intelligence Algorithms. *Future Trends in AI Research*, 1(1), 27-33.
DOI: 10.26855/ftair.2024.12.007

**Received:** October 17, 2024
**Accepted:** November 7, 2024
**Published:** November 28, 2024

***Corresponding author:** Yue Hou, Meiga Polytechnic Institute of Hubei, Xiaogan 432017, Hubei, China.

## Abstract

Traditional software development methods have many problems such as high price, low efficiency, and difficult quality assurance. It is urgent to introduce artificial intelligence algorithms to make changes and promote its automation transformation. Based on this, this article starts from the perspective of the entire software development life cycle, including requirements analysis, design, coding, testing and maintenance, and explains how to use artificial intelligence algorithms to improve the automation process of software development, aiming to provide reference for the intelligent transformation of the software development industry. AI (Artificial Intelligence) software development focuses on creating applications that can perform tasks requiring human-like intelligence, such as learning, reasoning, and decision-making. AI development requires interdisciplinary knowledge in computer science, mathematics, and domain-specific expertise. The field continues to evolve rapidly, driving innovation across industries. AI software development involves designing, building, and deploying applications that mimic human intelligence—enabling machines to learn, reason, and make decisions. This field combines computer science, data science, and domain expertise to create systems that improve over time.

## Keywords

Artificial intelligence algorithm; software development; automated process; software life cycle

In the context of the development of the digital age, various types of software are the key to driving the coordinated and efficient operation of systems and equipment. However, the traditional software development process faces the problems of being time-consuming, labor-intensive, and costly, and the development cycle is lengthened because the stages of demand analysis, design, coding, testing, and maintenance rely heavily on the support of professionals. In response to this, drawing on and using the powerful data analysis, pattern recognition, and learning capabilities of artificial intelligence technology can help solve the problems existing in software development and help achieve automation in software development.

## 1. Requirements Analysis Phase

### 1.1 Foundations of Natural Language Processing Technology

Natural language processing (NLP) technology plays a key role in the process of using artificial intelligence algorithms to parse software requirements described in natural language [1]. NLP covers a series of complex technologies



**Note:** The article was originally published in *Advances in International Computer Science*, and this is an authorized translation of it.

and methods that aim to enable computers to understand, analyze, and generate human language. Its basic technologies include lexical analysis, syntactic analysis, and semantic analysis.

### 1.1.1 Lexical Analysis

Lexical analysis divides natural language text into lexical units and annotates information such as part of speech. Hidden Markov Model (HMM) is a commonly used algorithm model for lexical analysis.

HMM is a dual random process, the state transition is a Markov process, and the observation value generation depends on the current state. Assuming that the hidden state set is and $Q = \{q_1, q_2, \cdots, q_N\}$ the observation value set is $V = \{v_1, v_2, \cdots, v_M\}$, for T the observation sequence $O = \{o_1, o_2, \cdots, o_T\}$ and state sequence of length $I = \{i_1, i, \cdots, i_T\}$, the HMM probability calculation is based on the following formulas:

State transition probability: $a_{ij} = P(i_{t+1} = q_j | i_t = q_i)$, which means the probability of being in the state $q_i$ at time $t$ and transferring to the state $q_j$ at time $t + 1$.

Observation probability: $b_j(k) = P(o_t = v_k | i_t = q_j)$, which indicates the probability of generating an observation value $v_k$ when it is in the state $q_j$ at time $t$.

Initial state probability: $\pi_i = P(i_1 = q_i)$, which represents the probability of being in the state $q_i$ at the initial moment.

The Viterbi algorithm solves the most likely state sequence and completes the part-of-speech tagging. The application example of HMM in lexical analysis is shown in Table 1.

**Table 1. Application of HMM in lexical analysis**

| Application | Description | Advantage | Limitation |
|---|---|---|---|
| Part-of-speech tagging | Use HMM's state transition and observation probability to mark the word part of speech | High computational efficiency, capable of processing large-scale text | Weak ability to capture long-distance dependencies |

### 1.1.2 Syntactic Analysis

Syntactic analysis focuses on analyzing the grammatical structure of sentences, constructing syntactic trees, and clarifying the grammatical relationship between words. Graph-based dependency syntactic analysis algorithms are commonly used [2]. In graph-based dependency syntactic analysis, sentences are viewed as graphs, with nodes as words and edges as dependency relationships. Assuming there is na word in a sentence, the score function of the dependency relationship is defined as: $s(i, j, r) = w_r^T \cdot \emptyset(i, j)$, where is $w_r$ the weight vector of $\emptyset(i, j)$ the dependency relationship and r is the feature vector between words i and j. The syntactic tree is constructed by finding the edge connection method that maximizes the score function. Taking the requirement statement "the system needs to have efficient data retrieval function to meet the user's needs for rapid information query" as an example, the algorithm can determine that "system" is the subject of "has", "data retrieval function" is the object of "has", and other dependencies. The relevant characteristics are shown in Table 2.

**Table 2. Syntax tree construction feature table**

| Application | Description | Advantage | Limitation |
|---|---|---|---|
| Syntax tree construction | Graph-based algorithm to find the optimal dependency relationship and build a syntactic tree | Able to handle complex sentences effectively | High requirements for data annotation quality |

### 1.1.3 Semantic Analysis

Semantic analysis aims to explore the deep meaning of text and eliminate semantic ambiguity. Semantic role labeling (SRL) algorithm plays an important role in semantic analysis [3]. Semantic role labeling is usually based on the predicate-argument structure. Assume that the predicate is and Pits argument set is $A = \{a_1, a_2, \cdots, a_n\}$. The semantic role of each argument, such as agent, patient, etc., is determined by the classification model. For example, in the

sentence "users use software to edit files", "user" is the agent of "use", "software" is the tool, and "file editing" is the purpose. The semantic role labeling model training is based on the maximum entropy model, and the formula is:

$$P(y|x) = \frac{1}{Z(x)} exp\left(\sum_{i=1}^{n} \lambda_i f_i(x,y)\right)$$

where $y$ is the semantic role label, $x$ is the input text feature, $Z(x)$ is the normalization factor, \(\lambda_i\) is the feature weight, and $f_i(x,y)$ is the feature function. The characteristics are shown in Table 3.

**Table 3. Semantic understanding characteristics table**

| Application | Description | Advantage | Limitation |
|---|---|---|---|
| Semantic Understanding | Label the semantic role of predicates to assist semantic understanding | Improve the accuracy of semantic understanding | Reliance on semantic knowledge base |

## 1.2 Deep Application of Deep Learning Models

### 1.2.1 Recurrent Neural Network (RNN)

RNN can effectively model demand text with sequence characteristics. The basic unit calculation formula is:

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

where $h_t$ is $t$ the hidden state at time t, $x_t$ is $t$ the input at time t, $W_{hh}$ is the weight matrix from hidden layer to hidden layer, $W_{xh}$ is the weight matrix from input layer to hidden layer, $b_h$ is the bias term, and $\sigma$ is the activation function. When analyzing a large amount of user feedback and requirement documents, RNN processes each word in the order of the text, captures the dependencies between words, and then identifies high-frequency requirements.

For example, in the feedback from many e-commerce software users, expressions such as “simplify the shopping process” and “improve payment convenience” frequently appear. RNN can accurately capture these high-frequency word combinations and identify them as high-frequency demands [4].

### 1.2.2 Long Short-Term Memory Network (LSTM)

LSTM solves the problem of gradient vanishing or gradient exploding when RNN processes long sequences, and is better at handling long-term dependencies in long text requirements. The LSTM unit contains an input gate $i_t$, a forget gate $f_t$, an output gate $o_t$, and a memory unit $c_t$. The calculation formula is as follows:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

$$\widetilde{c_t} = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

$$c_t = f_t \Theta c_{t-1} + i_t \Theta \widetilde{c_t}$$

$$h_t = o_t \Theta tanh(c_t)$$

where $\Theta$ represents element-by-element multiplication. By learning from a large number of past requirement documents, LSTM can discover potential requirements. For example, when analyzing a requirement document for an office software, although "multi-language support" is not directly mentioned, by deeply mining the relevant functional descriptions and user usage scenarios, LSTM can infer that in the context of globalization, the software has potential demand for multi-language support [5]. The performance comparison of RNN and LSTM in processing software requirement text is shown in Table 4.

**Table 4. Comparison of the performance of RNN and LSTM in processing software requirement text**

| Model | Long sequence processing capability | Gradient Problem | High-frequency demand recognition capability | Ability to explore potential needs |
|---|---|---|---|---|
| RNN | Weaker | Gradients are prone to vanishing or exploding | Better | Weaker |
| LSTM | Powerful | Effectively solve the gradient problem | Good | Powerful |

## 2. Design Phase

In the software design stage, artificial intelligence plays a vital auxiliary role, especially in software architecture design and module division. Genetic algorithm, as a classic optimization algorithm, is widely used in searching for the optimal software architecture solution. Genetic algorithm is based on Darwin's theory of evolution and simulates natural selection and genetic mechanisms. Its basic process includes initializing the population, where each individual in the population represents a software architecture solution; calculating the fitness of each individual, and the fitness function is designed according to the functional requirements and performance constraints of the software, for example, it can be defined as:

$$\text{Fitness} = w_1 \times \text{Functionality} + w_2 \times \text{Performance} + w_3 \times \text{Maintainability}$$

Among them, $\text{Functionality}$ represents the degree of satisfaction of software functional requirements, $\text{Performance}$ represents performance indicators such as response time and throughput, $\text{Maintainability}$ represents the maintainability of the software, and $w_1$, $w_2$ and $w_3$ are weight coefficients, which are adjusted according to the focus of the project.

Through genetic operations such as selection, crossover and mutation, the population is continuously updated iteratively, gradually approaching the optimal solution. For example, the selection operation can adopt the roulette selection method, and the probability of an individual being selected is proportional to its fitness. The crossover operation simulates the gene recombination in biological genetics, exchanging some genes of two parent individuals to generate new offspring individuals. The mutation operation randomly changes certain genes of an individual with a certain probability to maintain the diversity of the population.

At the same time, artificial intelligence can use neural network models in deep learning, such as multi-layer perceptrons (MLP), to learn existing excellent software design patterns. Through a large amount of labeled data training, the model can learn the relationship between different design patterns and software functions and performance. When facing a new software project, input the project's functional requirements, performance constraints and other information, and the MLP model can output reasonable design suggestions, including software architecture types, module division strategies, etc. The application comparison of genetic algorithms and MLP in the software design stage is shown in Table 5.

**Table 5. Algorithm model comparison table**

| Algorithm Model | Application | Advantage | Limitation |
|---|---|---|---|
| Genetic Algorithms | Search for the optimal software architecture solution based on the fitness function | Strong global search capability and can handle complex constraints | The amount of calculation is large and it is easy to fall into the local optimum |
| MLP | Learn excellent design patterns and provide design suggestions | Quick learning ability and quick to give advice | High reliance on training data and poor interpretability |

## 3. Coding Phase

Code generation technology is based on machine learning models and has greatly improved coding efficiency and accuracy. For example, language models based on the Transformer architecture, such as the GPT series, have performed well in the field of code generation. The Transformer architecture uses a multi-head attention mechanism that

can focus on different positions of the input sequence in parallel, thereby better capturing the semantic information in the text [6].

For code generation tasks, the model is trained based on a large code corpus. During the training process, the model learns the grammatical structure, semantic relations, and patterns of different functional codes of the code. When given a software design and requirement description as input, the model encodes the input through a self-attention mechanism, and then generates the corresponding code based on the learned code pattern. For example, for a simple requirement description "implement a function that calculates the sum of two integers", the model generates the following Python code framework:

```
def add_numbers(a, b):
result = a + b
return result
```

The characteristics of the language model based on the Transformer architecture in code generation are shown in Table 6.

**Table 6. Code features table**

| Feature | Description | Advantage | Limitation |
|---|---|---|---|
| Semantic understanding ability | Understanding demand semantics through self-attention mechanism | Able to generate code according to complex requirements | The generated code may contain logical errors and requires manual verification |
| Coding style adaptability | Learn multiple coding styles | Can generate code that conforms to a specific style | The effectiveness of code generation for specific domains is limited by training data |

## 4. Testing Phase

Artificial intelligence plays an important role in software testing in many aspects. In terms of test case prioritization, classification algorithms based on machine learning, such as support vector machines (SVM), can be used. SVM separates data of different categories by finding an optimal hyperplane. In test case prioritization, various features of test cases, such as execution time, number of lines of code covered, historical defect discovery rate, etc., are used as input features to classify the priorities into high, medium, and low categories. The decision function of SVM can be expressed as:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b\right)$$

Among them, $\alpha_i$ is the Lagrange multiplier, $y_i$ is the category label of the sample, $K(x_i, x)$ is the kernel function, band is the bias term.

In terms of automatic test case generation, mutation testing algorithms can be combined with machine learning. Mutation testing generates multiple variants by mutating the original code, and then judges the effectiveness of the test case based on the survival of the variants. The machine learning model can predict the code area where defects may occur based on the historical defect data and changes of the software, so as to generate test cases in a targeted manner. The comparison of the application effect of artificial intelligence in software testing is shown in Table 7.

**Table 7. Application effect table**

| Application | Algorithm Model | Advantage | Limitation |
|---|---|---|---|
| Test case prioritization | SVM | Can effectively prioritize test cases | High requirements for feature engineering and long model training time |
| Automatically generate test cases | Mutation testing combined with machine learning | Improve test case targeting and coverage | Mutation operations are computationally intensive and may generate redundant test cases |

## 5. Maintenance Phase

Artificial intelligence provides strong support for maintenance personnel in the software maintenance stage. In terms of code understanding, the abstract syntax tree (AST) combined with the graph neural network (GNN) technology can be used. AST parses the code into a tree structure, where nodes represent code elements and edges represent the relationship between elements. GNN can process AST and learn the structure and function information of the code.

For example, the graph convolutional network (GCN) in GNN can extract the semantic features of the code by performing convolution operations on the features of AST nodes, and provide code annotations and modification suggestions for maintenance personnel. In terms of predicting software maintenance workload and potential problems, time series analysis algorithms such as the ARIMA model can be used. The ARIMA model predicts future maintenance workload by analyzing historical maintenance data. Assuming that the maintenance workload time series is $y_t$, the expression of the ARIMA (p, d, q) model is:

$$\left(1-\sum_{i=1}^{p}\varphi_i B^i\right)(1-B)^d y_t = \mu + \left(1+\sum_{j=1}^{q}\theta_j B^j\right)\epsilon_t$$

Among them, $B$ is the backward shift operator, $\varphi_i$ is the autoregressive coefficient, $\theta_j$ is the moving average coefficient, $\mu$ is the constant term, and $\epsilon_t$ is the white noise sequence.

The application characteristics of artificial intelligence in the software maintenance stage are shown in Table 8.

**Table 8. Application characteristics of artificial intelligence in the software maintenance stage**

| Application | Algorithm Model | Advantage | Limitation |
|---|---|---|---|
| Code understanding | AST combined with GNN | Accurately analyze code structure and functionality | The ability to understand complex codes needs to be improved, and the computing resources are consumed a lot |
| Maintenance workload and problem prediction | ARIMA Model | Make effective forecasts based on historical data | High requirements for data stability and difficulty in handling sudden changes |

## 6. Conclusion

In the entire life cycle of software development, artificial intelligence algorithms play a powerful enabling role. However, artificial intelligence currently faces challenges such as data quality and model interpretability in the software development automation process. However, with the continuous development and innovation of technology, it is expected to achieve more efficient and intelligent software development and push the software industry to new heights.

## References

[1] Hu Y. Exploration of the application of artificial intelligence in computer application software development. Inf Rec Mater. 2025;26(2):81-3.

[2] Zhang Q. Research on the development of computer application technology in the era of artificial intelligence. Digit Commun World. 2024;(12):37-9.

[3] Tian L. Research on software development automation process based on artificial intelligence algorithm. Home Appl Repair. 2024;(1):49-51.

[4] Zhou W, Zhong W, Zhang L. Research on improving the testing efficiency of artificial intelligence equipment software. J Guilin Univ Aerosp Technol. 2022;27(4):463-7.

[5] Shi Q, Xue Z. Research on modular embedded software development based on artificial intelligence. Digit Commun World. 2019;(12):111.

[6] Wang Y. Discussion on the current situation and trend of archives informatization construction driven by the new generation of

artificial intelligence: based on the investigation of medical institutions. Arch Manag. 2024;(2):88-90.

[7] Ma Y, Guo F, Lu M. Application of cross-platform visual design in PySide artificial intelligence software development. Heilongjiang Sci. 2025;(6).

[8] Zhang J. Application of artificial intelligence in computer software development. In: Proceedings of the "2024 Vocational Education Week-Master Craftsmen of the Steel Industry Enter Campus" activity; 2024.