www.editada.org

International Journal of Combinatorial Optimization Problems and Informatics, 14(3), Sep-Dec 2023, 61-70. ISSN: 2007-1558. https://doi.org/10.61467/2007.1558.2023.v14i3.408

Genetic Programming in Software Engineering: A Systematic Literature Review

Ángel Juan Sánchez-García[*], *Leslie Loaiza-Meseguer, Jorge Octavio Ocharán-Hernández and Juan Carlos Pérez-Arriaga*

[1]Facultad de Estadística e Informática, UV. Xalapa, Veracruz México.
leslielm63@gmail.com, {angesanchez, jocharan, juaperez}@uv.mx

Abstract. Software Engineering is involved in all phases of the Software Development Life Cycle, implying a systematic and disciplined development process. Currently, there are optimization challenges within the phases and activities of Software Engineering, where a Genetic Programming (GP) approach can yield better results due to its individual representation. This Systematic Literature Review aims to analyze the current state of GP application in Software Engineering by identifying the phases and activities of software development where Genetic Programming has been applied and summarizing the advantages of using this technique. The results indicate that GP has been utilized in every phase of software development, predominantly in the construction phase. Moreover, Program Synthesis emerges as a promising area, significantly impacting new fields such as Genetic Improvement.

Keywords: Genetic programming ·Software Engineering · Systematic Literatura Review · Optimization

Article Info
Received 11 Sep, 2023
Accepted 11 Dec, 2023

1 Introduction

Industry 4.0 seeks to improve processes and products by incorporating new technologies, cloud computing, the Internet of Things, and Artificial Intelligence. Software Engineering, on the other hand, seeks to develop computer systems through a systematic, disciplined, and orderly process to obtain a quality product and reduce the number of defects. This implies that Software Engineering is present in all the life cycle phases of a software project (Boehm, 1976).

Optimization is vital in this development process since it seeks to maximize the performance of software and minimize resource consumption. There are optimization problems within the phases and activities of Software Engineering that need to be solved since, during the construction of a project, several factors can be found that negatively influence its performance, production time, and reliability, among other aspects (Yanyan & Renzuo, 2008).

Genetic Programming (GP) (Koza, 1992) has been used in Software Engineering to represent code structures. It is reported that problems in the construction phase have been addressed with optimization algorithms (Robles-Aguilar, 2021), specifically with GP for code refactoring (Chen, Zhang, & Zhao, 2009). However, Genetic Programming is not limited to the coding phase, as it has impacted activities such as Software reliability (Qi, Mao, Lei, Dai, & Wang, 2013), code repair (Rathore & Kumar, 2015), and defect prediction (Ernst & Gorton, 2014), among others. Also, GP can produce more efficient programs than traditional programming methods since it focuses on code optimization. The nature of the representation of individuals in this approach allows it

to be used mainly in implementation or code construction. However, this research is based on the idea that it can be used in the other phases of software development.

Therefore, this Systematic Literature Review aims to explore applications of Genetic Programming in the different phases of software development to present the current state of this optimization approach that allows software engineers to save resources in software projects. This research will allow software engineers to know optimization mechanisms that could be adjusted to the needs of each problem in different Software development activities. On the other hand, it will allow Artificial Intelligence researchers to know other GP application areas where improvements can be implemented to problems in the software area.

This paper is organized as follows: Section 2 describes the background and related work. Section 3 details the method used to carry out this research work. Section 4 presents the results obtained. Finally, Section 5 draws conclusions and future work.

2 Related Work

Recent works have shown that Artificial Intelligence can bring benefits in each of the phases of software development, for example, in requirements analysis (Ernst, & Gorton, 2014), design (Wangoo, 2018), coding, and testing (Xie, 2013). It is reported that problems in the construction phase have been addressed with optimization algorithms (Robles-Aguilar et al., 2021), specifically with Genetic Programming for code refactoring (Chen, Zhang & Zhao, 2009). However, some literature reviews about GP were found in a manual search as related work described below.

In (Espejo, Ventura, & Herrera, 2009), the authors reviewed the application of GP to classification, a task the most researched in machine learning and data mining. The authors showed the different ways in which this evolutionary algorithm can help in the construction of accurate and reliable classifiers.

A review of the application of Genetic Programming in water resources engineering was found in Mehr et al. (2018), where the authors delve into its variants, such as multigene GP, linear GP, gene expression programming, and grammar-based GP. This review shows the benefits of Genetic programming in engineering and a wide range of applications in hydrological, hydraulic, and hydro climatological domains.

Regarding Software engineering, in Afzal, & Torkar (2011), the authors report a systematic literature review on Software Engineering predictive modeling. This review covers 23 primary studies between 1995 and 2008. The review results show that symbolic regression using genetic programming has been applied in Software quality classification, Software cost/effort/size estimation, and Software fault prediction/software reliability growth modeling.

This paper analyzes the predictive capability of search-based techniques for ascertaining four predominant software quality attributes, i.e., effort, defect proneness, maintainability, and change proneness. In a later review (Malhotra, Khanna & Raje, 2017), the authors systematically reviewed the application of search-based techniques for Software Engineering predictive modeling. They report that Genetic Programming was present in 39% of the studies that addressed effort estimation, but they found no studies where this technique is used in Maintainability Prediction or Change Prediction.

Recently, a comprehensive review of the studies to improve the explainability or interpretability of machine learning models through GP (Mei et al., 2022) was identified. In this review, two groups related to explainable artificial intelligence by GP were generated. The first considers the interpretability, aiming to directly evolve more interpretable models by GP, On the other hand, the second focuses on post-hoc interpretability, which uses GP to explain other black-box machine learning models, or explain the models evolved by GP by simpler models such as linear models. This review shows the strong potential of GP for improving the interpretability of machine learning models and balancing the complex tradeoff between model accuracy and interpretability (Mei et al., 2022).

Table 1 summarizes the objective of the search and the area of application. Additionally, Table 1 lists the number of studies selected and the coverage of years of each review. As can be seen in this section, reviews on

the application of GP in other areas were found. Only one review in Software Engineering was found, however it does not cover all phases of Software development. It can also be observed that in Software Engineering, GP has been little explored compared to other metaheuristics. In summary, no Systematic Literature Review on the application of GP in Software Engineering was identified. For this reason, the main objective of this research is to describe the current state of the use of GP in each of the phases of the software development life cycle, emphasizing the benefits for (although not limited to) software engineers, software developers, and testers.

Table 1. Related work summary.

Ref	Year	Coverage	Target Search	Applied area	Number of primary studies
Espejo, Ventura & Herrera	2009	1998 – 2008	Genetic programming	Machine learning	155
Mehr et al.	2018	1997 – 2018	Genetic programming	Water resources engineering	150
Afzal & Torkar	2011	1995 – 2008	Genetic programming	Software estimation	23
Malhotra, Khanna & Raje	2017	1992 – 2015	Search-based techniques	Software estimation	78
Mei et al.	2022	1995 – 2022	Genetic programming	Explainable artificial intelligence	217

3 Research Method

The method used to carry out this systematic literature review is based on the guidelines proposed by Kitchenham & Charters (2007), described below.

3.1 Resarch Questions

The research questions that guided this systematic review and their motivations are shown in Table 2.

Table 2. Research questions.

Research question	Motivation
RQ1.- In which phases of software development have genetic programming been used?	The purpose of this question is to know the phases of software development in which genetic programming has been used to identify promising areas of this technique or its variants
RQ2.- In which activities of the software development phases have genetic programming been used?	It is essential to identify the specific activities of each Software Engineering phase where genetic programming has been applied to promote improvements in software engineers
RQ3.- What are the advantages of using genetic programming?	It is important to know the benefits of applying genetic programming and why it is used in different Software Engineering activities

3.2 Search strategy and data sources

The terms used to search the primary studies are defined in Table 3.

Table 3. Keywords and synonyms identified.

Keyword	Related terms
Software engineering	-
Genetic programming	GP

Being an exploratory study, each phase of software development (such as requirements, design, coding, or testing) was not placed in the search string. In addition, the term "Software Engineering" was added, which implies a development process, instead of putting only the term "software" since studies referring to using software to test genetic programming in different areas could be included. The search string used is based on the search terms defined above and is made up as follows:

"software engineering" AND "genetic programming"

Table 4 shows the databases that were selected for the search of the primary studies and their website.

Table 4. Data sources.

Database	Website
IEEXplore	https://ieeexplore.ieee.org/Xplore/home.jsp
ScienceDirect	https://www.sciencedirect.com/
ACM	https://dl.acm.org/
SpringerLink	https://link.springer.com/

3.3 Selection of primary studies

The inclusion and exclusion criteria are described in Tables 5 and 6. These criteria are proposed for the selection of primary studies of this research.

Table 5. Inclusion criteria.

ID	Description
IC1	Studies with full access
IC2	Studies published between 2017 and 2023
IC3	Studies in the title or abstract allude to any of the phases or activities of software engineering
IC4	Studies in the abstract indicate answering at least one research question

Table 6. Exclusion criteria.

ID	Description
EC1	Studies in a language other than English
EC2	Studies that are book chapters, presentations, abstracts, or technical reports
EC3	Repeated or duplicated studies

3.4 Selection procedure

The primary study selection procedure consists of four stages. Figure 1 shows in detail the primary study selection criteria previously defined in section 3.3 that are applied in each stage.

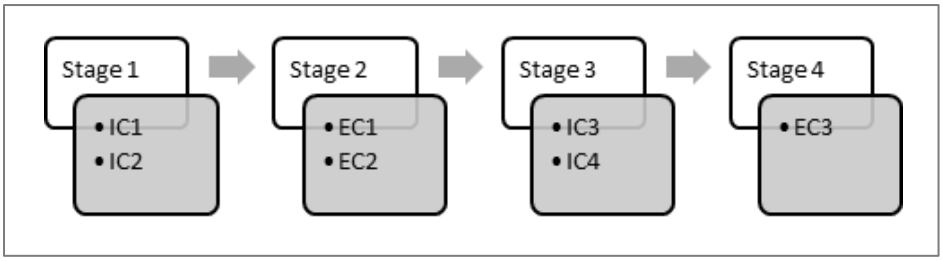

Figure 1. Primary study selection procedure.

Table 7 shows in detail the results of each database during the four stages. The list of references for the 44 primary studies selected can be found in "Appendix A: Primary Studies References." (2023). The template used to extract data from each primary study can be found in "Appendix B Data Extraction Template." (2023). The questions defined to evaluate the quality of primary studies can be found in "Appendix C Quality Assessment Questions." (2023).

Table 7. Application of inclusion and exclusion criteria by stage.

Stage	IEEEXplore	ACM	SpringerLink	ScienceDirect	Total
Initial search	1,934	1,545	139,851	560	**143,890**
Stage 1	627	301	78,180	58	**79,166**
Stage 2	13	196	374	46	**629**
Stage 3	13	18	14	4	**46**
Stage 4	**13**	**13**	**14**	**4**	**44**

4 Results

After applying the four selection stages, 44 primary studies were selected. Of these selected studies, 61% correspond to articles published in journals, while 39% are conference papers, as it is shown in Figure 2. Most of the studies belong to the source SpringerLink (31%), followed by ACM (30%), IEEE Xplore (30%), and ScienceDirect (9%).

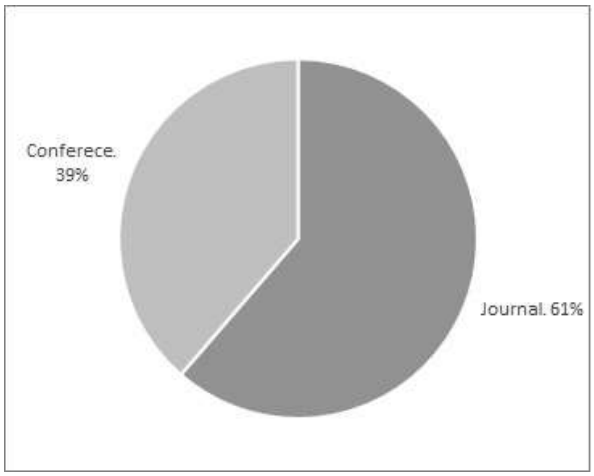

Figure 2. Selected primary studies by publication type.

The distribution of primary studies by year of publication was identified in Figure 3, where it can be seen that the majority of studies came from 2017 and 2021, and also, until March 2023, no study was selected with our selection criteria. In the following subsections, the most relevant information is extracted to answer each of the research questions.

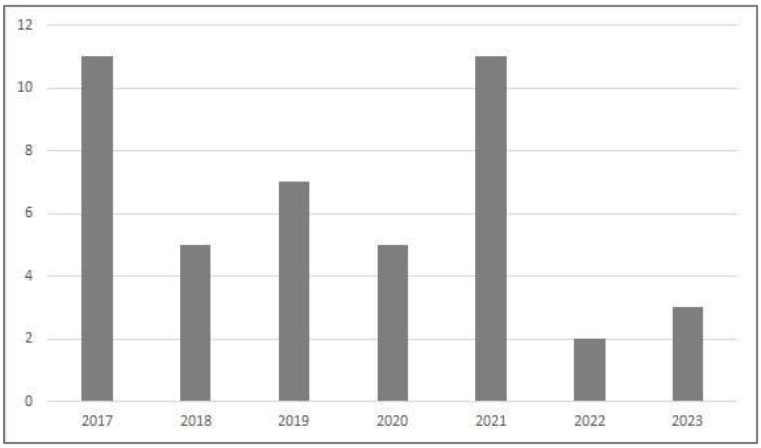

Figure 3. Selected primary studies by year.

4.1 RQ1. In which phases of software development has genetic programming been used?

As can be seen in Figure 4, the phase that has had the most applications of Genetic Programming is the construction phase, with 66% of the total selected studies. It was found that the tree structure used by Genetic Programming to represent its individuals (computer programs) is beneficial for the construction of software since it allows the creation of a new code from an existing code (Li et al., 2022); thus, removing branches from one tree to insert them into another (Langdon et al., 2017), which promotes the improvement of both functional and non-functional properties (Sohn & Yoo, 2021), automatic code generation (Miller, 2020), as well as code reuse and restructuring (Krauss, 2017).

Genetic Programming was found to be a tool that facilitates pattern identification (Huppe, Saied, & Sahraoui, 2017) . This property allows us to identify code smells (Kessentini & Ouni, 2017), locate faults (Kim, Mun, Yoo & Kim, 2019), and patch generation (Cao, Liu, Shi, Chu, & Deng, 2021); the latter enables automatic program repair (Yuan & Banzhaf, 2020). On the other hand, the Testing phase is mentioned in 19% of the

articles. Wei et al. (2018) again point out the ability of genetic programming for pattern identification, which, according to their study, allows the identification of the worst-case scenario in the execution of software, as well as the detection of vulnerabilities and performance errors. Other authors propose that the application of genetic programming allows the automation of black box testing (Drusinsky, 2017) and the evaluation of graphical interfaces (Ines, Makram, Mabrouka, & Mourad, 2017).

Regarding the Design phase, only 11% of the articles were found to mention it. It was found that genetic programming helps the automation of both prototype generation (Valencia-Ramírez et al., 2017) and the modeling of software product lines (Vescan, Pintea, Linsbauer, & Egyed, 2021). Finally, the Planning phase was found to be present in 2% of the articles and the Maintenance phase in 2%, indicating a lack of information on the areas of opportunity for genetic programming in these phases.

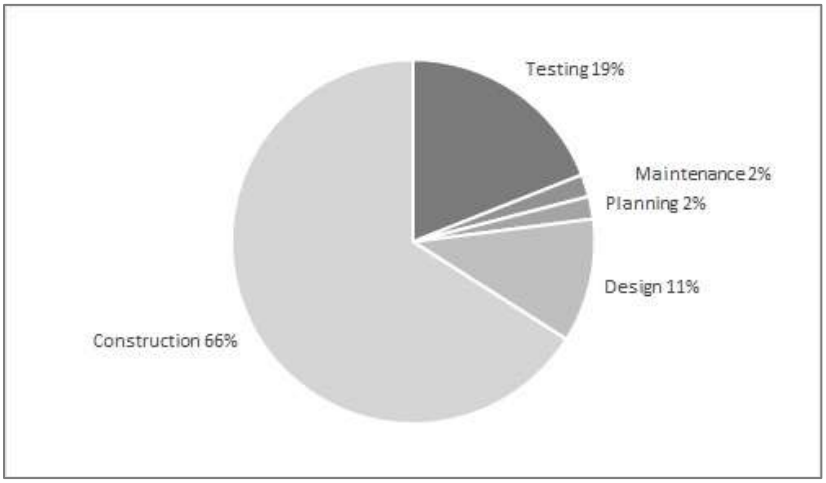

Figure 4. Selected primary studies by phase.

4.2 RQ2. In which activities of the software development phases have genetic programming been used?

Twenty-nine papers that mention activities of software development phases involving the application of genetic programming were found. In the Planning phase, by applying genetic programming in the restructuring of plans and initializing the population of individuals with existing plans, we can reuse their information to generate new plans (Kinneer, Coker, Wang, Garlan, & Goues, 2018).

In the Design phase, it was found that 67% of the items correspond to prototype generation. 33% of the articles refer to the modeling of Software Product Lines (SPL). This activity is achieved by automating the definition of basic elements and the way to combine them, to be subsequently composed and tested with real users and thus find the optimized compositions (Salem, 2017). The automatic generation of the generic models used by this activity is possible by utilizing an initial population of these and calculating the set of valid characteristics for each one to apply GP to them subsequently (Vescan, Pintea, Linsbauer, & Egyed, 2021).

In the Construction phase, we identified that 32% of the items correspond to the activity of automatic program repair. This activity aims to generate error repairs without human intervention, without the need for special instrumentation or annotations in the source code (Sobania & Rothlauf, 2021). This application searches and generates modifications from an abstract syntax tree that can patch a bug in the underlying program and creates new program variants by mutation and crossover (Kinneer, Coker, Wang, Garlan, & Goues, 2018). A 32% of the articles mention the automatic coding of programs; for this activity, it is important to mention that one of the most relevant applications of genetic programming is the technique called program synthesis, which can automatize the coding of programs by automatically generating source code in a programming language, that maintains the constraints of a predefined specification (Sobania & Rothlauf, 2021).

It was determined that 31% of the articles corresponding to the identification of bugs in the software, which is achieved through the identification of recurrences (Yuan & Banzhaf, 2020) and sequence-by-sequence learning (Li, 2022). These properties are helpful in code smell identification (Kessentini & Ouni, 2017) and fault localization (Sohn, & Yoo, 2021). Finally, 5% of the articles mention that Genetic Programming supports code restructuring by identifying patterns within the code to optimize its fragments and reuse them to create new code (Krauss, 2017).

In the testing phase, it was found that 67% of the articles talk about black box testing, in which the application of GP is helpful in discovering local variables, actions performed on output variables, counting loops, and while loops due to the ability of GP to discover a functional relationship between data features and to group them into categories (Drusinsky, 2017). Also, the application of Genetic Programming in black box testing is proper when identifying worst-case execution and vulnerabilities in programs by identifying patterns in data inputs (Wei, Chen, Feng, Ferles & Dillig, 2018). 33% of the articles allude to interface evaluation, where genetic programming allows the automatic generation of rules to evaluate their quality, providing previously defined quality metrics, context criteria, and a list of possible types of problems, taking advantage of the principle of Genetic Programming where individuals adapt to their environment through mutation and crossover (Ines, Makram, Mabrouka, & Mourad, 2017).

4.3 RQ3. What are the advantages of using genetic programming?

Based on the answers to questions RQ1 and RQ2, it was found that there are numerous advantages thanks to the mutation and crossover operators, the principles of biological evolution on which genetic programming is based, and its ability to identify patterns in the different phases and activities of software development.

Genetic programming in the Planning phase serves to restructure plans, reducing the costs of operating in complex environments of change and uncertainty by adapting autonomously to change in the pursuit of its quality objectives (Kinneer, Coker, Wang, Garlan, & Goues, 2018). In the Design phase, prototyping (Salem, 2017) and modeling of software product lines (Vescan, Pintea, Linsbauer, & Egyed, 2021) can be automated using genetic programming, significantly improving the performance of these activities (Valencia-Ramírez et al., 2017). In the Construction phase, the application of this technique helps in the automation of different activities such as program repair (Sobania, & Rothlauf, 2021), bug identification (Yuan, & Banzhaf, 2020), and code restructuring (Krauss, 2017). Obtaining the improvement of functional and non-functional properties, such as code size, execution time, or memory consumption (Sohn, & Yoo, 2021). Furthermore, genetic programming is a technique that has excellent flexibility since it offers the possibility of handling a large number of individuals and of reworking the solutions obtained by relaunching a new evolution from one or more previously obtained solutions so that, with its application, the activity of evaluating graphical interfaces can be automated and thus optimize the process involved (Sohn, & Yoo, 2021). All this together helps a software engineer to do his job efficiently since it eliminates the manual part of his work and increases the quality of his results.

5 Conclusions and Future Work

A systematic Literature Review was carried out where the selection process of primary studies was divided into four phases where; after applying the previously defined selection criteria, 41 primary studies were obtained as a result.

Although Genetic Programming is an Artificial Intelligence technique that uses concepts of biological evolution to create and optimize programs, the studies analyzed showed that it had been used in the different stages of software development, although the implementation phase predominates. The analysis of the studies also allowed us to identify the advantages of its application in both functional and non-functional properties, in addition to the usefulness for a software engineer to use this technique as an automation tool in the different processes that exist at the time of software development. As a result, the objectives of the research work were achieved.

It was found that Genetic Programming has a great relationship with well-established areas, for example, Program synthesis, which substantially impacts new fields such as Genetic Improvement. This field of science uses Genetic Programming to correct bugs in software and improve functional and non-functional software requirements (Krauss, 2017). Therefore, in future work, we will seek to identify the applications and advantages of these areas.

References

"Appendix A: Primary Studies References." (2023). Retrieved November 13, 2023, from https://drive.google.com/file/d/1XNJYEli1rDjU3SWQeYYBRYJy2aT67XOP/view?usp=drive_link

"Appendix B Data Extraction Template." (2023). Retrieved November 13, 2023, from https://docs.google.com/document/d/1JQ4x7up_ZlkXBol26z500ff-wK4DlH14IWh274JBJp8/edit?usp=sharing

"Appendix C Quality Assessment Questions." (2023). Retrieved November 13, 2023, from https://drive.google.com/file/d/1oXzxAj8LgK0tIfyG6kamhQHlWOOHfLUT/view?usp=drive_link

Afzal, W., & Torkar, R. (2011). On the Application of Genetic Programming for Software Engineering Predictive Modeling: A Systematic Review. *Expert Systems with Applications*, 38(9), 11984-11997.

Boehm (1976). Software Engineering. *IEEE Transactions on Computers*, 25(12), 1226-124.

Cao, H., Liu, F., Shi, J., Chu, Y., & Deng, M. (2021). Automated Repair of Java Programs with Random Search via Code Similarity. In 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C) (pp. 470 - 477). IEEE.

Chen, H., Zhang, Y., & Zhao, J. (2009). Improved Genetic Programming Model for Software Reliability. In 2009 International Asia Symposium on Intelligent Interaction and Affective Computing (pp. 164–167). IEEE.

Drusinsky, D. (2017). Reverse Engineering Concurrent UML State Machines Using Black Box Testing and Genetic Programming. *Innovations in Systems and Software Engineering*, 13, 117-128.

Ernst, N. A., & Gorton, I. (2014). Using AI to Model Quality Attribute Tradeoffs. In 2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE) (pp. 51–52). IEEE.

Espejo, P. G., Ventura, S., & Herrera, F. (2009). A Survey on the Application of Genetic Programming to Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(2), 121-144.

Huppe, S., Saied, M. A., & Sahraoui, H. (2017). Mining Complex Temporal API Usage Patterns: An Evolutionary Approach. In 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C) (pp. 274–276). IEEE.

Ines, G., Makram, S., Mabrouka, C., & Mourad, A. (2017). Evaluation of Mobile Interfaces as an Optimization Problem. *Procedia Computer Science*, 112, 235-248.

Kessentini, M., & Ouni, A. (2017). Detecting Android Smells Using Multi-Objective Genetic Programming. In 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft) (pp. 122–132). IEEE.

Kim, Y., Mun, S., Yoo, S., & Kim, M. (2019). Precise Learn-to-Rank Fault Localization Using Dynamic and Static Features of Target Programs. *ACM Transactions on Software Engineering and Methodology*, 28, 1–34.

Kinneer, C., Coker, Z., Wang, J., Garlan, D., & Goues, C. L. (2018). Managing Uncertainty in Self-Adaptive Systems with Plan Reuse and Stochastic Search. In Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems (pp. 40-50).

Kitchenham, B., & Charters, S. (2007). Guidelines for Performing Systematic Literature Reviews in Software Engineering.

Koza, J. R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection (1st ed.). MIT Press.

Krauss, O. (2017). Genetic Improvement in Code Interpreters and Compilers. In Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (pp. 7–9). ACM.

Langdon, W. B., Lam, B. Y. H., Modat, M., Petke, J., & Harman, M. (2017). Genetic Improvement of GPU Software. *Genetic Programming and Evolvable Machines*, 18, 5-44.

Li, D., Wong, W. E., Jian, M., Geng, Y., & Chau, M. (2022). Improving Search-Based Automatic Program Repair with Neural Machine Translation. *IEEE Access*, 10, 51167-51175.

Malhotra, R., Khanna, M., & Raje, R. R. (2017). On the Application of Search-Based Techniques for Software Engineering Predictive Modeling: A Systematic Review and Future Directions. *Swarm and Evolutionary Computation*, 32, 85-109.

Mehr, A. D., Nourani, V., Kahya, E., Hrnjica, B., Sattar, A. M., & Yaseen, Z. M. (2018). Genetic Programming in Water Resources Engineering: A State-of-the-Art Review. *Journal of Hydrology*, 566, 643-667.

Mei, Y., Chen, Q., Lensen, A., Xue, B., & Zhang, M. (2022). Explainable Artificial Intelligence by Genetic Programming: A Survey. *IEEE Transactions on Evolutionary Computation*, 27(3), 621-641, https://doi.org/10.1109/TEVC.2022.3225509

Miller, J. F. (2020). Cartesian Genetic Programming: Its Status and Future. *Genetic Programming and Evolvable Machines*, 21, 129-168.

Qi, Y., Mao, X., Lei, Y., Dai, Z., & Wang, C. (2013). Does Genetic Programming Work Well on Automated Program Repair? In 2013 International Conference on Computational and Information Sciences (pp. 1875-1878). IEEE.

Rathore, S. S., & Kumar, S. (2015). Comparative Analysis of Neural Network and Genetic Programming for the Number of Software Faults Prediction. In 2015 National Conference on Recent Advances in Electronics & Computer Engineering (RAECE) (pp. 328–332). IEEE.

Robles-Aguilar, A., Ocharan-Hernandez, J. O., Sanchez-Garcia, A. J., & Limon, X. (2021). Software Design and Artificial Intelligence: A Systematic Mapping Study. In 2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT) (pp. 132–141). IEEE.

Salem, P. (2017). User Interface Optimization Using Genetic Programming with an Application to Landing Pages. Proceedings of the ACM on Human-Computer Interaction, 1, 1-17.

Sobania, D., & Rothlauf, F. (2021). A Generalizability Measure for Program Synthesis with Genetic Programming. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 822-829).

Sohn, J., & Yoo, S. (2021). Empirical Evaluation of Fault Localisation Using Code and Change Metrics. *IEEE Transactions on Software Engineering*, 47, 1605-1625.

Valencia-Ramírez, J. M., Graff, M., Escalante, H. J., & Cerda-Jacobo, J. (2017). An Iterative Genetic Programming Approach to Prototype Generation. *Genetic Programming and Evolvable Machines*, 18, 123-147.

Vescan, A., Pintea, A., Linsbauer, L., & Egyed, A. (2021). Genetic Programming for Feature Model Synthesis: A Replication Study. *Empirical Software Engineering*, 26, 1-29.

Wangoo, D. P. (2018). Artificial Intelligence Techniques in Software Engineering for Automated Software Reuse and Design. In 2018 4th International Conference on Computing Communication and Automation (ICCCA) (pp. 1–4). IEEE.

Wei, J., Chen, J., Feng, Y., Ferles, K., & Dillig, I. (2018). Singularity: Pattern Fuzzing for Worst Case Complexity. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 213 - 223). ACM.

Xie, T. (2013). The Synergy of Human and Artificial Intelligence in Software Engineering. In 2013 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE) (pp. 4–6). IEEE.

Yanyan, Z., & Renzuo, X. (2008). The Basic Research of Human Factor Analysis Based on Knowledge in Software Engineering. In 2008 International Conference on Computer Science and Software Engineering (pp. 1302–1305). IEEE.

Yuan, Y., & Banzhaf, W. (2020). Toward Better Evolutionary Program Repair. *ACM Transactions on Software Engineering and Methodology*, 29, 1-53.