Software Design oriented to Functional Programming: A Systematic Literature Review

Authors Name/s per 1st Affiliation (Author)
line 1 (of Affiliation): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: Email: name@xyz.com

Authors Name/s per 2nd Affiliation (Author)
line 1 (of Affiliation): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: Email: name@xyz.com

***Abstract*—The abstract goes here. DO NOT USE SPECIAL CHARACTERS, SYMBOLS, OR MATH IN YOUR TITLE OR ABSTRACT.**

***Keywords*-component; formatting; style; styling;**

I. Introduction

Software design is a key activity in the development cycle of a system, as its importance lies in being key to producing reliable, understandable and easy-to-maintain software [1]. In this sense, Functional Programming (FP) has become a popular approach in recent years

due to software development benefits [2][9], for example, adopting an FP style could make our programs more robust, more compact and easy to parallelize [4].

However, despite the benefits that FP brings to the software development cycle, currently most of the material related to Software design lacks clear and detailed specifications on how to implement a system created from Functional Programming. This does not happen for example in Object-Oriented Programming where there is a large amount of design resources and tools. In addition, the complexity and cost of Software design tasks using FP play an important role in its lack of adoption in industry [5].

With the aim of presenting a compilation of methodologies, practices and artifacts for functional programming-oriented software design as well as open challenges in this approach. This study carries out a systematic review of literature to portray the state-of-the-art in this field benefiting mainly software developers and consequently academic community and researchers who seek to carry out an analysis and application of design under FP context.

This paper is organized as follows. In section II, Related Work is presented. Section III describes the research method used to carry out the systematic literature review. In section VI, this systematic review is described. Section V presents the research results obtained from the synthesis of information carried out in the previous section. In section VI, the results are discussed. Finally, in section VII, the conclusions of the review are presented.

II. Related Work

Due to the challenge presented in collecting specific research papers related only to FP-oriented Software design, as a main criterion, those studies that gave us a general overview of FP throughout the Software development cycle were considered, regardless of the year in which they were published. These studies were mainly extracted from the digital libraries of IEEE Xplore and ACM Digital Library.

Now, as for the literature reviewed, the importance of functional programming in Software design is highlighted, especially in the implementation of parallel and distributed systems [6][8][9]. It is discussed how functional programming can improve software readability, maintainability, modularity and security [9]. In addition, specific characteristics of statically-typed functional programmers are explored and how this affects program construction [7]. On the other hand, the paper on "code smells" in Elixir suggests the need to identify and correct suboptimal code structures to improve software quality [10]. Overall, this systematic literature review offers a comprehensive view of the relevance of functional programming in software design, covering technical and psychological aspects of program construction.

It is also noted that the scarcity of specific works related only to FP-oriented Software design may indicate that the topic has so far been little explored in the literature and that there is a gap in terms of research and developments in this field. It highlights that it is a broad field of research and that it is constantly evolving with technologies related to Software development.

III. Research Method

The research method used to carry out this SLR was based on the guidelines provided by Kitchenham and Charters [citation], which consists of three main phases: (1) Planning, presented in this section, (2) Conducting, and (3) Reporting, described in sections IV and V, respectively.

A. Planning

In this phase, the study is planned and documented through a research protocol. Planning involves three impor-

Table I
SPECIFIC OBJECTIVES

No.	Specific Objective
1	Identify the methodologies and practices used in the design of PF-oriented software, as well as the artifacts generated in them, by collecting studies in order to understand the panorama of software design in the FP.
2	Categorize the practices and artifacts resulting from the application of PF-oriented software design through an analysis of the collected information to help developers optimize their design process, identifying areas for improvement and thus ensuring that systems are more efficient and scalable when working under that paradigm.
3	Identify the open challenges in software design when working under the context of the FP paradigm through the collected information for the advancement and evolution of the functional paradigm, as well as for the formation of strategies and tools to overcome these challenges.
4	Propose lines of research for future work according to the information gathered in the study through the analysis of it, in order to advance the knowledge and application of the functional paradigm in software design and thus encourage its adoption in industry and academia.

tant activities: specifying research questions, developing the protocol, and validating it. Through these activities, search concepts, search strings, definition of data sources and search strategies, specification of selection criteria (inclusion and exclusion criteria), data extraction strategy, and synthesis method are derived.

1) Search process: The search process includes activities such as describing research questions, defining the search strategy, academic sources to use as information sources, and specifying search concepts to construct the search string in the selected academic sources.

a) Research questions:: To define research questions, specific objectives need to be defined, so each research question will be closely related to an objective. I shows the defined specific objectives.

Having defined the specific objectives of the document in I, the research questions are shown in II along with their justification.

b) Search strategy:: The search for studies was carried out in four academic databases, shown in III. In addition, an automated search strategy was used to obtain the results from the selected search sources. This process involved applying the search string to all metadata through the search engine of each source. It is important to specify that for the ACM Conferences results, only 25% of the results were reviewed. Additionally, the Connected Papers tool was also used, which is an online platform that allows users to search and discover scientific articles in a wide variety of fields.

c) Search concepts: To define a search string, the most relevant concepts and their related terms were analyzed based on the research questions. The search concepts were:

- Software Design
- Functional Programming
- Methodology

Table II
RESEARCH QUESTIONS

Research Question	Main Motivation
RQ1. What are the practices followed when designing Software oriented towards the Functional Programming paradigm?	To identify open challenges and analyze design artifacts related to designing Software oriented towards the FP, it is important to first identify the methodologies and practices followed when working under this approach. Additionally, it is relevant to consider why this methodology is being used and how it aids in the Software Development process.
RQ2. What are the methodologies followed when designing Software oriented towards the Functional Programming paradigm?	
RQ3. What artifacts are produced when designing Software oriented towards the Functional Programming paradigm?	In addition to identifying the methodologies and practices followed in designing Software oriented towards the Functional Programming paradigm, it is important to identify artifacts produced during these processes, in order to categorize them under an analysis.
RQ4. What are the challenges that arise when designing Software oriented towards the Functional Programming paradigm?	Henderson [5] argues that the Functional Programming paradigm is little adopted by industry due to the complexity and cost that its design tasks can become. In addition to this, given the scattered nature of information regarding Software Design under this approach, it is important to identify the new challenges that development teams face when they need to design Software under the approach of this paradigm, as this can generate new lines of research for future work.

Table III
SELECTED SEARCH SOURCES

Data source	Location
ACM Digital Library	dl.acm.org
IEEE Xplore	xplore.ieee.org
SpringerLink	link.springer.com
ScienceDirect	sciencedirect.com

d) Search string: To define a search string that covered the maximum range of obtained results, terms derived from the search concepts were used. The resulting search string is defined as follows:

(*"Software Design"* OR *Design* OR *Diagram OR Pattern* OR *Methodology*) AND (*"Functional Programming"* OR *"Functional Paradigm"*)

B. Primary study selection

a) Selection criteria: The criteria will be used to determine which studies are included or excluded in the RSL. In Table IV, the inclusion criteria (IC) are shown, while in V, the exclusion criteria (EC) are shown.

b) Selection process: In the first stage, inclusion criteria IC1 and IC2 are applied to determine whether full access to the publication is available and whether it was published within a specific range of years. In the second stage, exclusion criterion EC1 and inclusion criterion IC3 are applied to determine whether the publication is written in English and

Table IV
INCLUSION CRITERIA

ID	Inclusion Criteria
IC1	Full access to the publication is available.
IC2	Publications published between the years 2012 to 2022.
IC3	The title, abstract, or content of the publication shows indications of being related to the topic of interest.

Table V
EXCLUSION CRITERIA

No.	Exclusion Criteria
EC1	Publications that are in a language other than English.
EC2	The publication is a derivative of another

Table VI
DATA EXTRACTION

Data Extraction	
Title	
Authors	
Year	
Source	
Reference or publication details	
Type of publication (reports, journal, etc.)	
Keywords	
Abstract	
Research question answered	
Notes	
Data Synthesis	

whether the title, abstract or content suggests that it is related to the topic of interest. In the third stage, inclusion criterion IC4 and exclusion criterion EC2 are applied to determine whether the publication answers at least one of the research questions posed and whether it is not a derivative of another publication. This way, the most relevant and appropriate primary studies are selected for the systematic literature review.

C. Data extraction and synthesis

a) Data extraction strategy: After selecting the primary studies according to the selection criteria, the information was extracted from the studies, including the necessary data to answer the research questions. It is worth mentioning that the extracted information was validated by more than one researcher to validate this process. The format used for extraction is found in VI.

b) Data synthesis strategy: To answer the research questions, the narrative synthesis method was followed according to how it was proposed by Kitchenham and Charters [2] in order to analyze the extracted information according to the strategy that was followed. The results of the narrative synthesis are presented in the section V.

D. Quality Assessment

In order to determine the quality of each selected publication and to determine to what extent the research results are valid and free from bias, a checklist was used. The process for evaluation was as follows:

1) Rate the studies using the checklist(s):
   - Simple scoring of yes (1) / partially (0.5) / no (0)
2) Validate the ratings
3) Use the results of the quality evaluation.

The checklist used is shown in VII, based on the proposal by Dybå & Dingsøyr[cite] but with some modifications to fit the project.

Now, once the studies were evaluated, it was possible to classify the publications according to the rating obtained in the Checklist. Table VIII shows the specified classification according to the obtained rating.

Table VII
QUALITY ASSESSMENT CHECKLIST

No.	Evaluation criteria
1	Is the document based on research (or is it simply a "lessons learned" report based on expert opinions)?
2	Is there a clear statement of the research objectives?
3	Is there an adequate description of the context in which the research was conducted?
4	Was the research design appropriate to address the research objectives?
5	Were the data collected in a manner that addressed the research topic?
6	Was the data analysis sufficiently rigorous?
7	Is there a clear statement of the findings?
8	Does the study have value for research or practice?

Table VIII
QUALITY ASSESSMENT CLASSIFICATION

Score	Classification
7-8 points	High quality
5-6 points	Medium quality
0-4 points	Low quality

E. Conducting

After conducting searches according to the predefined selection criteria in the selected search engines, 100 results were obtained, which were then subjected to the selection criteria. Figure 1 shows the process that was followed while applying the criteria, and the included studies with their year of publication are presented in the table.

The results obtained from the selection process for all stages and by source are shown in Figure 2.

Likewise, as mentioned in section 3, the Backward snowballing and Forward snowballing strategies were also carried out, as well as the use of the Connected Papers tool. Their results are shown in Figure 3.

IV. RESULTS

A. Year of publication

The low level of activity with respect to the growing interest in Functional Programming-oriented Software design was identified. Specifically, 2016 was designated as the year

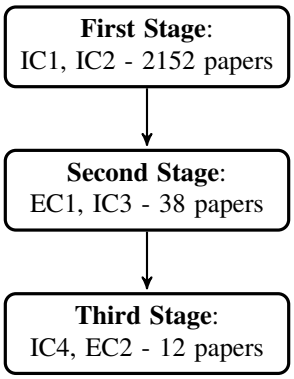

Figure 1. Total papers of the selection process by stage

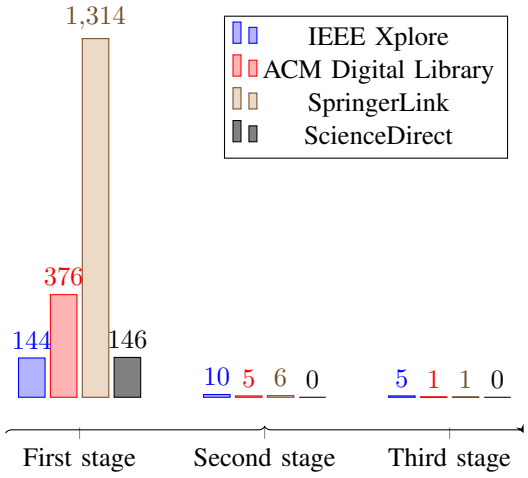

Figure 2. Results of the study selection process by source and by stage

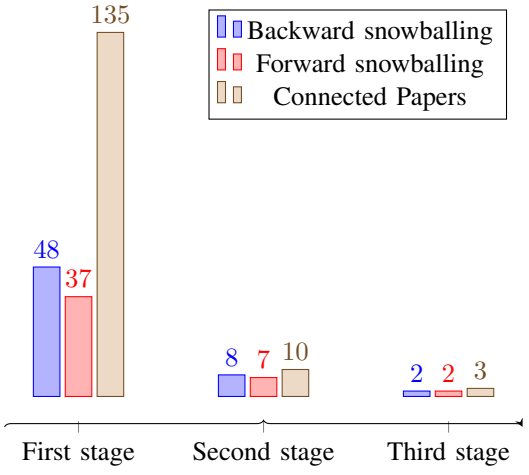

Figure 3. Results of additional search processes by stage

Table IX
STUDY SELECTION

ID	Author	Year	Source
SS1	Bailes, P., & Brough, L. [11]	2012	First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches
SS2	Sousa, T. B., & Ferreira, H. S. [12]	2012	Eighth International Conference on the Quality of Information and Communications Technology
SS3	Weck, T., & Tichy, M. [13]	2016	IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering
SS4	Pottier, F. [14]	2017	Proceedings of the ACM on Programming Languages
SS5	Poole, M. [15]	2019	3rd IEEE Blocks and Beyond Workshop
SS6	Motara, Y. M. [16]	2020	2nd International Multidisciplinary Information Technology and Engineering Conference
SS7	Motara, Y. M. [?]	2021	Trends in Functional Programming: 22nd International Symposium
SS8	Gibbons, J. [18]	2013	Central European Functional Programming School: 5th Summer School
SS9	Motara, Y. M. [19]	2020	Proceedings of the 35th Annual ACM Symposium on Applied Computing
SS10	Bailes, P. [20]	2013	Proceedings of IASTED International Conference Advances in Computer Science
SS11	Bailes, P., Brough, L., & Kemp, C. [21]	2014	International Conference on Software Engineering Advances
SS12	Piróg, M., & Wu, N. [22]	2016	Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming
SS13	Teatro, T. A., Eklund, J. M., & Milman, R. [23]	2018	IEEE Canadian Conference on Electrical & Computer Engineering
SS14	Ionescu, C., & Jansson, P. [24]	2016	Trends in Functional Programming in Education

with the most studies published on the subject. Figure 4 shows the trends in studies by year.

B. Distribution by classification

Of the 14 selected studies, the distribution of their classification was identified, of which 10 were identified as Journal type and 4 as Conference type. Graphically, they can be visualized in Figure 5.

C. Data sintesis

1) RQ1: What are the practices followed when designing Software oriented towards the Functional Programming paradigm?

A shortage of Software Design practices oriented towards Functional Programming has been observed in the literature, as they are not explicitly mentioned in comparison to the

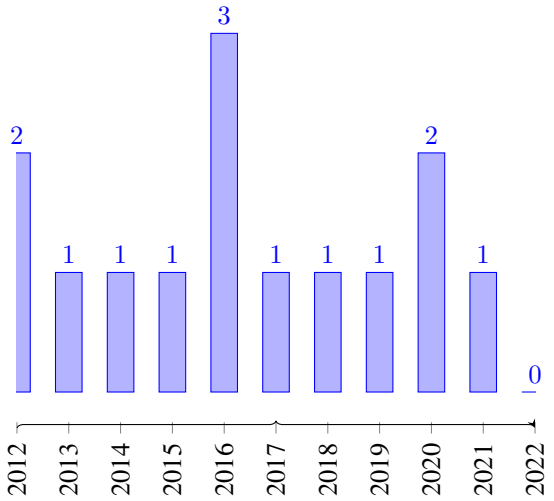

Figure 4. Results of additional search processes by stage

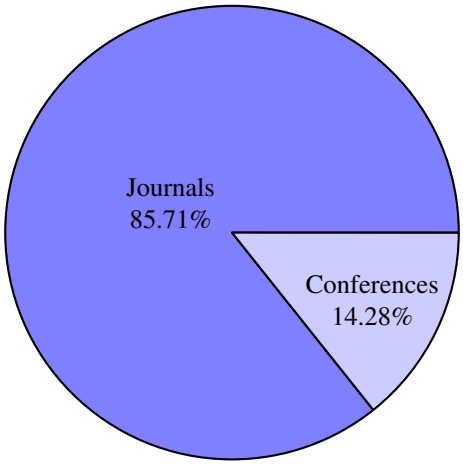

Figure 5. Distribution of studies by classification

amount present in other paradigms, such as Object-Oriented. However, by conducting a deeper analysis of the information, the following practices could be found:

Modeling high-level functional abstractions: these are concepts and techniques used in Functional Programming to simplify and abstract the logic of a program, i.e., focusing on efficiently solving general and generic problems. These abstractions are based on the use of functions and are designed to be intuitive and easy for the programmer to use. These high-level functional abstractions are useful for simplifying and abstracting complex logic in a program, which facilitates its long-term design and maintenance [16]. Some examples of high-level functional abstractions include:

- Monads: they are a data structure that encapsulates a value and provides a series of operations to work with that value in a safe and controlled way.
- Functors: they are a class of data type that can be "applied" to other data types to create a new data type.

Designing with mathematical foundations: the Functional Programming paradigm is closely related to mathematics, as it is based on the use of mathematical functions to describe computation. For this reason, it is common to use mathematical languages as Software Design tools in the context of Functional Programming. This allows for precise and concise modeling of the behavior of a functional system, which facilitates its design and understanding by the programmer [17]. Additionally, functional thinking tends to use mathematics as an intuitive and general tool for modeling systems, which can be very useful in the context of software design.

Identification of functions and data: in Object-Oriented Programming, the common process is to extract nouns and turn them into classes, followed by analyzing verbs to find the methods of a class. On the other hand, in Functional Programming, the process is reversed. Verb extraction is followed by noun extraction to discover the data with which a process operates [17]. The function or process is central to the design of functional systems, and the fundamental element to extract is the verb. This helps to build safe, predictable, and easy-to-compose and reuse systems.

Use of Recursive Patterns: Recursive Patterns allow avoiding details and complexities of general recursion, i.e., they simplify the definition of a large number of functions that would otherwise require an explicit recursion definition [20], offering developers greater comfort and reliability with general recursion, which greatly benefits the Software Design process.

2) RQ2: What are the methodologies followed when designing Software oriented towards the Functional Programming paradigm?

There has been found that literature reports very little information about Software Design methodologies focused on Functional Programming, and like RQ1, they are not explicitly found. However, in study SS8, the following is found:

Domain-Specific Languages (DSLs) can be a valuable methodology in Functional Software Design. DSLs are programming languages designed to solve specific problems in a particular domain and have simplified syntax and semantics designed to be easily understood by domain experts. DSLs are often used in Software Design to improve development efficiency and productivity, as they allow developers to express clearly and concisely the requirements and constraints of the specific problem they are trying to solve. Functional programming languages are particularly suited to host DSLs. Language features such as algebraic data types, higher-order functions, and lazy evaluation contribute [18]. With a clear definition, the following analysis can be reached:

A Software Design methodology based on the use of DSLs can be developed, where a specific process is followed for designing and implementing a DSL to solve a specific

problem in a particular domain.

Likewise, covering the topic of DSLs, there are some Software Design methodologies that include the creation of DSLs as one of their stages, such as Domain-Driven Design (DDD) [17]. Therefore, these methodologies could be considered valid when listing Software Design methodologies focused on Functional Programming. In fact, although it was outside the scope of this research work, books such as Domain Modeling Made Functional The Pragmatic Bookshelf [REFERENCIA] and Functional and Reactive Domain Modelling [REFERENCIA], which deal with these topics in more detail, were found.

3) RQ3: What artifacts are produced when designing Software oriented towards the Functional Programming paradigm? During the research, various approaches were identified for designing software oriented towards functional programming, as well as some of their advantages and disadvantages, highlighting that many of them are not effective or inadequate due to the foundations on which they are based. For example, the use of UML in functional programming may be limited due to its focus on graphical representation of the software structure, which is a challenge when designing software under this approach, rather than on functional logic. This can hinder the understanding and implementation of functional designs. It is important to seek new forms of functional software design that align with the principles of this paradigm and allow for greater efficiency and precision in development.

Table 13 shows the artifacts reported in the literature along with the study where they were found. Among the artifacts found in the literature, recursive patterns (APF-09) stand out. Their main advantage is that they simplify and clarify the way of programming, compared to the use of general recursion, that is, they offer programmers opportunities for greater comfort in the way they perform recursion [20] [21]. These patterns allow us to avoid the details and complexities of general recursion.

TABLA

It is important to mention that although these recursive patterns are not exclusive to functional programming, they have a strong relationship with it. In functional programming, functions tend to be pure, that is, they always return the same result when provided with the same arguments. This facilitates problem solving through recursion. Additionally, in functional programming, mutable variables are avoided, which makes recursion a very useful technique for solving problems in an elegant and concise manner. In summary, recursion is a widely used technique in functional programming due to the properties of pure functions and immutability.

4) RQ4: What are the challenges that arise when designing Software oriented towards the Functional Programming paradigm?

Within the main challenges reported in the literature on Functional Programming-oriented Software Design, the following points stand out:

Table X
CHALLENGES IN FUNCTIONAL SOFTWARE DESIGN

ID	Challenge	Paper
R-01	Lack of a standard for designing Software focused on Functional Programming	SS6, SS7, SS9
R-02	Lack of focus on understanding programs written in functional languages	SS3
R-03	Difficulty of the paradigm	SS5
R-04	Ineffective or inadequate notations and diagrams	SS6, SS7

Lack of a standard for designing Software focused on Functional Programming: Currently, there is no standardized way to graphically model functional programs as is done with UML for object-oriented programs. This makes it difficult for developers of functional systems, as they must present them hidden behind object-oriented interfaces or give up the benefits of modeling [16].

Lack of focus on understanding programs written in functional languages: Although there are many approaches to help understand programs written in object-oriented languages, there is little research aimed at understanding programs written in functional languages. And while there are already approaches to visualize functional programs, they only visualize data structures, not complete programs [13]. For example, if a reverse engineering team does not have the appropriate artifacts (such as design) to understand and analyze code written in functional languages, it may be difficult to modify, extend, or reuse it.

Difficulty of the paradigm: It is important to mention that the learning curve of the functional paradigm can be challenging compared to other paradigms [15]. Therefore, in addition to the little research aimed at understanding programs written in functional languages, designing Software using this paradigm can present additional challenges.

Ineffective or inadequate notations and diagrams: There are many notations and different types of diagrams, such as UML or BPMN, that have been adapted to design Software oriented towards Functional Programming [16] [17]. However, it is important to note that these are not effective for designing in a functional way, as they were not originally grounded in concepts related to Functional Programming (such as higher-order functions, for example), making a functional system difficult or impossible to model using these artifacts.

Table X reports the four main challenges reported in the literature along with the reference of the study where it is mentioned.

In general, it can be observed that the most notable open challenges in the literature are R-01 and R-04, which have a direct traceability according to the artifacts found in RQ3, since a large number of findings were not reported, and some of these artifacts contain disadvantages in their use.

The found challenges can be classified as follows:

- Conceptual Challenges: they relate to the understanding and application of the principles of functional programming, such as immutability, composition, and abstraction.
  - R-01, R-03
- Design Challenges: these challenges refer to the lack of a standard for designing software focused on functional programming, as well as the lack of adequate tools and techniques to effectively represent the design of software written in functional languages.
  - R-01, R-04
- Understanding Challenges: these challenges relate to the difficulty that developers may have in understanding and maintaining code written in functional languages due to its complexity and abstraction.
  - R-01

V. Discussion

VI. Conclusion

The conclusion goes here. this is more of the conclusion

Acknowledgment

The authors would like to thank... more thanks here

References

[1] S. S. Yau and J. J-P. Tsai, *A survey of software design techniques*, IEEE Transactions on Software Engineering, no. 6, pp. 713–721, 1986.

[2] D. Abajirov, *Does functional programming improve software quality?: an empirical analysis of open source projects on GitHub*, B.S. thesis, 2021

[3] A. Khanfor and Y. Yang, *An overview of practical impacts of functional programming*, 24th Asia-Pacific Software Engineering Conference Workshops (APSECW), IEEE, 2017, pp. 50–54.

[4] K. Hinsen, *The promises of functional programming*, Computing in Science Engineering, vol. 11, no. 4, pp. 86–90, 2009.

[5] P. Henderson, *Functional programming, formal specification, and rapid prototyping* IEEE Transactions on Software Engineering, no. 2, pp. 241–250, 1986.

[6] J. G. H. Pineda and Á. Puertas-González, *La programación funcional y las arquitecturas multicore: estado del arte*, Ingenio Magno, vol. 6, pp. 124-136, 2015.

[7] J. Lubin, *How Statically-Typed Functional Programmers Author Code*, in *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1-6, 2021.

[8] S. R. Vegdahl, *A survey of proposed architectures for the execution of functional languages*, IEEE transactions on computers, vol. 33, no. 12, pp. 1050-1071, 1984.

[9] A. Khanfor and Y. Yang, *An overview of practical impacts of functional programming*, in *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*, pp. 50-54, 2017.

[10] L. F. d. M. Vegi and M. T. Valente, *Code smells in Elixir: early results from a grey literature review*, in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pp. 580-584, 2022.

[11] P. Bailes and L. Brough, "Making sense of recursion patterns," in *2012 First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)*, pp. 16–22, IEEE, 2012.

[12] T. B. Sousa and H. S. Ferreira, "Object-functional patterns: Re-thinking development in a post-functional world," in *2012 Eighth International Conference on the Quality of Information and Communications Technology*, pp. 348–352, IEEE, 2012.

[13] T. Weck and M. Tichy, "Visualizing data-flows in functional programs," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, pp. 293–303, IEEE, 2016.

[14] F. Pottier, "Visitors unchained," *Proceedings of the ACM on Programming Languages*, vol. 1, no. ICFP, pp. 1–28, 2017.

[15] M. Poole, "A block design for introductory functional programming in Haskell," in *3rd IEEE Blocks and Beyond Workshop, B and B 2019*, pp. 31–35, IEEE, 2019.

[16] Y. M. Motara, "String diagrams for modelling functional programming," in *2020 2nd International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, pp. 1–7, IEEE, 2020.

[17] Y. M. Motara, "High-Level Modelling for Typed Functional Programming," in *Trends in Functional Programming: 22nd International Symposium, TFP 2021, Virtual Event, February 17–19, 2021, Revised Selected Papers 22*, pp. 69–94, Springer, 2021.

[18] J. Gibbons, "Functional programming for domain-specific languages," in *Central European Functional Programming School: 5th Summer School, CEFP 2013, Cluj-Napoca, Romania, July 8-20, 2013, Revised Selected Papers 5*, pp. 1–28, Springer, 2015.

[19] Y. M. Motara, "A structural modeling notation for the typed functional paradigm," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pp. 1515–1522, ACM, 2020.

[20] P. Bailes, "Recursion patterns and their impact on programming language design," in *Proceedings of IASTED International Conference Advances in Computer Science (ACS 2013)*, pp. 450–459, 2013.

[21] P. Bailes, L. Brough, and C. Kemp, "Fundamentals, Prospects and Challenges for Totally Functional Programming Style."

[22] Pir'og, Maciej and Wu, Nicolas. *String diagrams for free monads (functional pearl)*. In Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, pages 490–501, 2016.

[23] Teatro, Timothy AV and Eklund, J Mikael and Milman, Ruth. *Maybe and Either Monads in Plain C++ 17*. In 2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE), pages 1–4, 2018.

[24] Ionescu, Cezar and Jansson, Patrik. *Domain-specific languages of mathematics: Presenting mathematical analysis using functional programming*. arXiv preprint arXiv:1611.09475, 2016.