2017 24th Asia-Pacific Software Engineering Conference Workshops

An Overview of Practical Impacts of Functional Programming

Abdullah Khanfor
School of Systems and Enterprises
1 Castle Point Terrace
Stevens Institute of Technology
Hoboken, New Jersey 07030
Email: akhanfor@stevens.edu

Ye Yang
School of Systems and Enterprises
1 Castle Point Terrace
Stevens Institute of Technology
Hoboken, New Jersey 07030
Email: ye.yang@stevens.edu

***Abstract*—Functional Programming (FP) is a programming paradigm in which the mathematical function evaluation is the main block in building the software. FP languages are more suitable for handling parallelism and concurrency. Over the years, many different FP languages and programming style had been proposed and studied. However, FP is generally considered to be difficult to learn and master than another programming paradigms such as object-oriented programming. In this paper, we aim at deriving an understanding of practical impacts of using this paradigm in the software engineering practices through a literature review.**

Keywords—Functional Programming, Software Engineering, Literature Review.

I. Introduction and Background

The limitations on producing a faster clock rate Central Processing Unit (CPU), the growing number of interconnected devices, and the demanding for more computing power raises significant challenges for developing software that overcomes these constraints. Therefore, fundamental changes are required for programming methodologies and paradigms to utilize the existing computing power. Different programming paradigms have been proposed and applied in order to solve different problems such as Object-Oriented Programming (OOP), Functional Programming (FP), Symbolic Programming, etc. The simplicity of OOP eases the representation of many problems, and the numerous OOP tool suites speed up the wide-spread of OOP. Moreover, a large number of programmers familiar with OOP paradigm is a cornerstone of its popularization in building complex systems.

Nowadays, with a physical limitation of faster CPU [1] that adopt Moore's Law [2] is the primary motivation to employ parallelism and concurrency of distributed multi-core systems. Parallel computing is hard to achieve [3], but the more demand with multi-core processors and distributed systems that require higher and well-structured computation processes. However, McKenney highlighted that parallel programming is considered to be hard topic [3], largely due to difficulty in FP paradigm. The high cost of parallel systems and the relatively rare demand made many developers not exposed to it. This resulted in the lack of expertise in parallel programming within typical researchers and practitioners; the low accessibility to working parallel code; and the high overhead communication between different parts. Hughes [4] demonstrated modular programming in FP that it's the essential aspects in improving developers productivity in the early 1990's. Unfortunately, general claims with each programming paradigm, including FP, that will improve the software development process [5]. To develop further understanding on practical evidences of FP in software development life cycle, the main motivation of this study is to conduct in-depth research to identify the effects on software development life-cycle caused by adopting FP.

II. Literature Search and Selection Criteria

The early introduction of FP in the late 1960s propose a challenge to collect all the possible research papers. Not only that, the growing trend over the years for the usage of keyword "*Functional Programming*" in software engineering and computer science research, is another layer to collect papers. Especially, the influence of FP in software development life cycle (SLDC). Our paramount inclusion criterion is to examine the papers that show any evidence of FP impact in SDLC such as Design, Implementation, Testing, Maintenance, etc. Therefore, we exclude papers unrelated to our criterion; e.g., papers concern in the programming language design. Afterwards, published papers in one of the conferences and journals related to FP or Software Engineering are collected for the next phase of our research. The literature language, of the gathered papers, is in English. Therefore, An exclusion for publications in other languages from our literature review process. Lastly, all the accumulated papers are before August 2017.

We perform the search in different online libraries. Considering the publishers related to computer science and software engineering topics. The digital libraries that we conduct are: *IEEE Xplore*, *ACM Digital Library*, *Elsevier ScienceDirect*, *Springer Link Online*, *Wiley Online Library*, *Taylor & Francis*, *Oxford Journals* and *Google Scholar*. We apply *Functional Programming* as search term to all selected online libraries. This end up with 184 artifacts relevant to the topic. A first round of screening process by reading the title and abstract in which is performed to the collected research papers, talks, books, reviews and articles. Papers with titles relevant to our criterion had been examined carefully to extract informative ideas and conclusions related to this paper. Figure 1 reflects the collected paper for the study and shows the cumulative frequency over the years of publication.

978-1-5386-2649-8/17 $31.00 © 2017 IEEE
DOI 10.1109/APSECW.2017.27

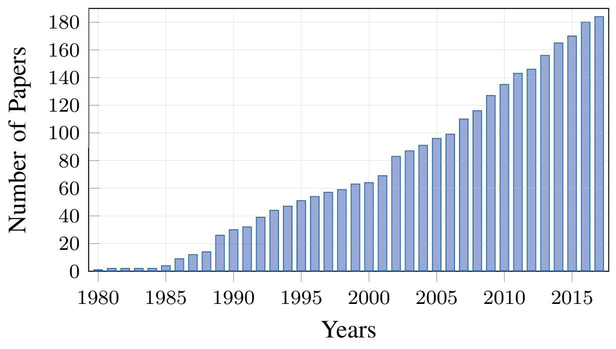

Fig. 1. Cumulative frequency of papers over the years of publication based on the literature search.

III. Study Design

In mind, the criterion for collecting papers; we designed the following research questions (RQs), to answer from the collected literature:

- **RQ1: What are the research outlook in FP paradigm?**
  By analyzing the literature and measuring the interaction between the authors to draw an understanding of what has been done and the research direction expected.
- **RQ2: In which software engineering development practice FP paradigm had been studied?**
  To understand the practical impacts of using this paradigm in the software engineering practice.

To answer *RQ1*, Analysis of the research landscape by studying the collected literature. The categorization based on different topics that effects SDLC such as Cost Estimation, Defects, etc. Not only that, an additional categories to help us answer *RQ2* such as Paradigms Comparative Studies, Education, etc. Table I shows each paper studied and it corresponding category.

TABLE I. Research paper covered and the corresponding Software Engineering topic.

Paradigms Comparative Studies	[6]
Performance	[6] [7]
Developer Productivity	[4]
Testing	[6] [8] [9]
Security	[10] [11] [12]
Cost Estimation (Effort)	[13] [14] [6]
Reusability	[15] [13] [16] [4] [17]
Design	[18] [19]
Defects & Bugs	[6] [8] [9]
Distributed	[20]
Parallel & Concurrency	[20] [21] [22]
Education	[20] [23]

IV. Functional Programming in Software Engineering Practices

FP programming used in software engineering practices since 1960's. As an example of early studies, McCarthy [24] and Landin [25] provide papers in this topic. FP is a style, in which, one could write functional programs in any language, but of course with vastly differing levels of effort. Generally, we call the languages that encourage FP if its main goal to encourage and enforce the developer to use the mathematical functional style as the main block to bulid the software [20]. No doubt that FP approach to develop applications is not a question of the suitability of this paradigm to the industry.

We found one qualitative study by Pankratius et al. [6] that reflects examine the effects of FP with practitioners. The study covers the level of effort to write the code in FP differs than other paradigm. As a matter of fact, it surveyed 13 developers using Scala and Java to represents FP and imperative paradigms, respectively as a controlled comparative. This type of studies are required to showcase the effect of such a paradigm in the developer productivity, and the risks might impose by using FP. Finally, it concludes practical insights for FP in industry. In our case, we try to explore more related studies that impact software development to answer *RQ1* and *RQ2*.

Before we start our study, an important term differentiate between Pure and Impure FP languages. Pure functional languages, such as Haskell, provide lazy evaluation and equational reasoning; Impure functional languages, such as SML, provide features such as state, exception handling, or continuations [13]. The advantages of using pure or impure differ on the application and requirements, but generally state and exception handling are important features for developers in industry.

A. Functional Programming and Software Design

Software design is a critical phase in software development life-cycle. Despite that, it is essential to consider the programming paradigm that could be used during that phase. Although, Kiczales et al. [19] stated that OOP, Procedural Languages, and FP have a fundamental abstraction and composition mechanisms rooted in some form of Generalized Procedure (GP) languages. In which GP languages break down the system into units of functions. Besides that, scholars mainly focuses on the FP languages design and implementation and smaller set of studies focus on the software processes aspects. First, with purely FP on large scale software and with the lack of expert developers in this paradigm could affect significantly in software management, in general. In particular, Henderson [18] stated that the complexity and the cost of software design tasks using this paradigm plays a significant role in the lack of adoption in the industry. Moreover, he proposed a particular mixture of FP, formal specification, and rapid prototyping as an effective methodology for software design.

B. Functional Programming and Security

State changes during the execution of the code before returning the final value rises unintended side effects, especially, in critical mission and real-time software's [20]. In fact, some studies claim that writing programs using functional languages increases the security by enforcing the programmers to a particular method of writing the code that mitigates security risks by forbidding the state changes in pure FP. Doligez et al. [10] by using OCaml's, a FP language semantics to fit the security requirements of XSVGen to investigate and build a secure XML validator by ensuring a software robustness and security in three levels, the source, compilation, and at the execution level. With such study that adopts from FP semantics an improving in the security of the proposed tools and languages had been noticed. Another example of utilizing FP concepts to build an Attack Tree Domain Specific meta-Language introduced by Damjanovic and Djuric [11]. By using a Magic Potion [12], a purely functional Domain Specific meta-Language that is grounded in FP and DLs is commonly used to model different ways in which a system can be attacked. We concluded from the different studies claims that using the pure FP languages will ensure some security aspects by ensuring the no state changes in software execution.

C. Functional Programming and Cost Estimation

The imperative languages tend to be longer in lines of code comparing to FP languages. Thus, an effect in the software cost and effort estimations. Moreover, Pure FP small changes may require an extensive restructuring of the program to meet these changes. Therefore, the productivity of software developer. By using the impure functional languages wisely, these changes can reduce the effort in few lines of codes [20]. Coleman and Johnson [14] conduct an empirical study by investigating a Scala projects from GitHub. The study concludes a contradiction in the belief of mathematical expressiveness of functional languages will not affect the modular writing codes. However, the authors suggested a future research in this direction for better understanding the differences between FP and other programming paradigms. Also, Coleman and Johnson [14] notice that hard decision counting, such as if statements, alone are deficient for solid estimating *M* because Scala high-order functions could make decisions that take Boolean returning function objects. It is also known "predicate contexts.".

D. Functional Programming and Software Reuse

FP languages, such as Ensemble and Erlang, provide a sophisticated module system. Higher-order functions encourage and promote the reuse and the most well-known example is *map* and *fold* functions [26]. For that, Functional developers claim an increase in productivity comparing to conventional counterpart because of the short functional programs [4]. Harrison et al. [15] conducted a compression between OOP and FP by using C++ and Standard ML (SML) to represent each paradigm. The results show that the number of the function definitions in both paradigms had no significantly different. However, 60% more functions were called in the SML code than in the C++, and nearly four times as many library functions. It concludes that metric for reuse the code is one and a half time higher using SML comparing to C++ code. For that, using FP languages encourage the reuse practice in programming than OOP languages. In addition, a conceptual limitation in conventional languages regarding modular the problems. By encouraging the developer to recursive calls of other functions we promote the reuse of codes and functions. Moreover, a framework is a helpful approach to provide semi-finished classes in intend to help developers to build solutions on top of it. Hence, it encourages the reusability to enhance codes and minimizes the effort needed to redo it. The frameworks usage and proposal is a concrete topic of software engineering. However, designing reusable library components consider to be a hard problem and requires expertise in the language and dedication over a period to ensure a working framework. No doubt, building a framework popularize the usage of language in industry and therefore, one of the first framework proposed for FP language by et al. Budimac [16] in 1996. Another influential framework Genome Analysis Toolkit (GATK), by McKenna et al [17], which uses FP concepts of MapReduce.

E. Functional Programming and Software Testing and Defects

Harrison et al. [15] that the time taken for testing an SML (FP) code comparing to C++ (OOP) is 105% more. Hence, it needs more time to build test cases for FP comparing to one written in C++. This may be caused by the familiarity of the developers to write test cases that are suitable for OOP languages. Furthermore, there is a big increment of 118% in SML (FP) of the number of known errors per 1,000 non-comment source lines comparing to C++ (OOP). As an example of difficult bugs can be found, by only examining the code, is a race condition. During the system testing, a vast amount of code will be tested that often hard to developers to diagnose the error resulting from the race conditions [8] [9]. In summary, testing in FP require more time and effort from the developers to write test cases.

F. Functional Programming for Parallel and Distributed Systems

Building a programs for parallelism is consider to be matter of a black art. it is a trial and error with experience to make the program speedups in parallel environment; with less feedback from the runtime system [7]. FP immutable data structure results in advantage for distributed systems. The messages between different nodes in the system need to be consistent and the mutable data structure, in imperative languages, can results in changing of the semantics and the end result of the execution [20]. To make FP suitable for parallel programming is the abstraction level provided by this paradigm with higher order functions and polymorphism [20]. Moreover, pure functional programs which the main properties is no-side effects. Therefore, an advantage for parallel evaluation [21] [22]. As result, it is easier to identify parallel tasks in program comparing to imperative programs [20]. In fact, Googles MapReduce [27] model is inspired by the map and reduce functions used in FP. Thus, A parallel and distributed computing made accessible for developers in a field that was exclusive for experts and specialized programmers [20]. As an example for providing a tool to understand and help in parallel performance and finding bugs; Jones et. el [7] for a Parallel Haskell programs.

G. Functional Programming and Education

Functional languages influence are increasing in industry and academia [28]. Functional paradigm importance introducing the strong connection between programming and mathematics [23]. In fact, the history of teaching a functional language go back to 1980s as one of the first books to taught for functional language scheme for beginners called "Structure and Interpretation of Computer Programs" [29] by MIT. The increasing attention to this paradigm is seen through European universities as a higher number of institutions teaching and researching this paradigm comparing to US universities counterpart. In fact, a few number of Top US universities teaches FP according to a survey (http://www.pl-enthusiast.net/2014/09/02/who- teaches-functional-programming/) mentioned in Hu et al. [20] paper. Additionally, As we research through the literature, we found a recommendation from Bailey's to his book, Functional Programming with Hope [23], as a good resource for education and beginners to understand this paradigm.

H. Functional Programming and Programming Languages Support.

We call a language functional if its design encourages or to some extent enforces a functional style [20]. We collect the top 10 languages that support FP paradigm. The support, in our study, means a higher-order function and lambda calculus that are features provided by the language design. In Table II shows the top 10 languages based on TIOBE Index August 2017 [30] programming languages ranking. We show the languages that support that paradigm ranked and from the collected literature that study each language.

TABLE II. FUNCTIONAL PROGRAMMING LANGUAGES POPULARITY BASED ON TIOBE INDEX RANK AND THE FREQUENCY OF LANGUAGES IN THE COLLECTED LITERATURE.

Programming Language		
#	**TIOBE Index Rank**	**Literature Rank**
1	Python	Haskell
2	PHP	Java
3	JavaScript	Clojure
4	Swift	Erlang, Lisp, and OCaml
5	R	Merinda
6	Dart	Scala and SML
7	D	C++, F#, and Scheme
8	Scala	Bootstrap
9	Prolog	C#
10	Erlang	Clean

As we see from Table II, Python is the most widely used programming language according to TIOBE index that support FP paradgim. Where the literature we collected and examined, in our study, shows Haskell is the most language had been used and/or studied based.

The answer for both *RQ1* and *RQ2* are overlapping. From the previous sections, the answer for *RQ1* that, we only highlighted a small set of research papers in FP for each category we propose. But from the initial effort, notice a great amount of well-established research in different software engineering aspects in FP. Despite that, the answer for *RQ2*, we found a lack of qualitative and survey-based research that conduct a field studies and getting a feedback from developers to inspect the effects of FP in the software engineering practices. We are confident such studies could give a insightful knowledge for researchers and practitioners to ensure some concepts and claims around FP.

V. CONCLUSION & FUTURE WORK

FPs are getting increasing interests and applications by the practitioners. The growth of programming languages and the research trends emphasize the value of the paradigm. We examined 184 scientific papers to understand the landscape of the field. Moreover, a considerable amount of papers identified the effects of functional programming in software life-cycle. Believing that more studies need to be conducted from the software engineering research community to propose the sufficient tools and frameworks in order to help the developers. A need for more qualitative studies, in general, to highlight the benefits and drawback of adopting FP in software engineering processes. Finally, FP is well-established research topic and there are more papers had investigate the software engineering practice aspects that we didn't include in our paper that worth to be examined the future work.

REFERENCES

[1] L. B. Kish, "End of moore's law: thermal (noise) death of integration in micro and nano electronics," *Physics Letters A*, vol. 305, no. 3, pp. 144–149, 2002.

[2] I. Present, "Cramming more components onto integrated circuits," *Readings in computer architecture*, vol. 56, 2000.

[3] P. E. McKenney, "Is parallel programming hard, and, if so, what can you do about it?(v2017. 01.02 a)," *arXiv preprint arXiv:1701.00854*, 2017.

[4] J. Hughes, "Why functional programming matters," *The computer journal*, vol. 32, no. 2, pp. 98–107, 1989.

[5] T. Mens, "Introduction and roadmap: History and challenges of software evolution," in *Software evolution*. Springer, 2008, pp. 1–11.

[6] V. Pankratius, F. Schmidt, and G. Garretón, "Combining functional and imperative programming for multicore software: an empirical study evaluating scala and java," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012, pp. 123–133.

[7] D. Jones Jr, S. Marlow, and S. Singh, "Parallel performance tuning for haskell," in *Proceedings of the 2nd ACM SIGPLAN symposium on Haskell*. ACM, 2009, pp. 81–92.

[8] M. Christakis and K. Sagonas, "Static detection of race conditions in erlang," *Practical Aspects of Declarative Languages*, pp. 119–133, 2010.

[9] K. Claessen, M. Palka, N. Smallbone, J. Hughes, H. Svensson, T. Arts, and U. Wiger, "Finding race conditions in erlang with quickcheck and pulse," *ACM Sigplan Notices*, vol. 44, no. 9, pp. 149–160, 2009.

[10] D. Doligez, C. Faure, T. Hardin, and M. Maarek, "Avoiding security pitfalls with functional programming: a report on the development of a secure xml validator," in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 2. IEEE, 2015, pp. 209–218.

[11] V. Damjanovic and D. Djuric, "Functional programming way to interact with software attacks and vulnerabilities," in *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*. IEEE, 2010, pp. 388–393.

[12] D. Djuric and V. Devedzic, "Magic potion: Incorporating new development paradigms through metaprogramming," *IEEE software*, vol. 27, no. 5, pp. 38–44, 2010.

[13] P. Wadler, "The essence of functional programming," in *Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1992, pp. 1–14.

[14] R. Coleman and M. A. Johnson, "Power-laws and structure in functional programs," in *Computational Science and Computational Intelligence (CSCI), 2014 International Conference on*, vol. 2. IEEE, 2014, pp. 168–172.

[15] R. Harrison, L. Samaraweera, M. R. Dobie, and P. H. Lewis, "Comparing programming paradigms: an evaluation of functional and object-oriented programs," *Software Engineering Journal*, vol. 11, no. 4, pp. 247–254, 1996.

[16] Z. Budimac, M. Ivanovic, and S. Zivkov, "On the design of a functional programming framework," in *Systems, Man, and Cybernetics, 1996., IEEE International Conference on*, vol. 2. IEEE, 1996, pp. 1548–1552.

[17] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly *et al.*, "The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data," *Genome research*, vol. 20, no. 9, pp. 1297–1303, 2010.

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

[19] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, “Aspect-oriented programming,” *ECOOP’97Object-oriented programming*, pp. 220–242, 1997.

[20] Z. Hu, J. Hughes, and M. Wang, “How functional programming mattered,” *National Science Review*, vol. 2, no. 3, pp. 349–370, 2015.

[21] M. M. Chakravarty, R. Leshchinskiy, S. Peyton Jones, G. Keller, and S. Marlow, “Data parallel haskell: a status report,” in *Proceedings of the 2007 workshop on Declarative aspects of multicore programming*. ACM, 2007, pp. 10–18.

[22] K. Hammond, “Why parallel functional programming matters: Panel statement.” in *Ada-Europe*, vol. 6652. Springer, 2011, pp. 201–205.

[23] R. Bailey, *Functional programming with Hope*. Ellis Horwood, 1991.

[24] J. McCarthy, “Recursive functions of symbolic expressions and their computation by machine, part i,” *Communications of the ACM*, vol. 3, no. 4, pp. 184–195, 1960.

[25] P. J. Landin, “The mechanical evaluation of expressions,” *The Computer Journal*, vol. 6, no. 4, pp. 308–320, 1964.

[26] P. Wadler, “How to solve the reuse problem? functional programming,” in *Software Reuse, 1998. Proceedings. Fifth International Conference on*. IEEE, 1998, pp. 371–372.

[27] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[28] M. Boespflug, “Conversion by evaluation,” in *International Symposium on Practical Aspects of Declarative Languages*. Springer, 2010, pp. 58–72.

[29] H. Abelson, G. J. Sussman, and J. Sussman, *Structure and interpretation of computer programs*. Justin Kelly, 1996.

[30] T. software BV. (2017) Tiobe index for august 2017. Accessed: 2017-08-28. [Online]. Available: https://www.tiobe.com/tiobe-index