FPA Findings for Flutter against the existing Development Frameworks

Sana Rizwan, Shoaib Uzair, Zaid Saeed and Abdullah Naveed

Abstract— Rise of web/mobile applications running different operating systems, the problem arises to cater to their needs. The need of cross-platform development comes into play. These platforms ease the complexity for the developer by allowing them to write a single code that would work on multiple OS. However, cross-platform development comes at a cost of trading native speed and accuracy for portability. Nevertheless, Flutter is an open-source Software Development Kit (SDK) that combines high performance and reliability for Android, iOS and now with the advent of Flutter 2.0 Web Applications into one package. Cost/ budget effectiveness can calculated for the differentiation with Effort calculation, productivity index measurements by weighting factors in PL size depending upon Functional Point Analysis technique.

Index Terms— Flutter, Google, iOS, Android, Cross-platform Development, React Native, Kotlin, Swift, Functional Point, External Inputs, Internal inquiry

1 INTRODUCTION

Contemporary practice in the software world is there are two major mobile application operating systems i.e, Android and iOS. These two platforms have little to nothing in common, so developing applications that look, feel and perform the same is very challenging and require different skill sets. For instance, Android Native applications are written in Java and/or Kotlin and iOS is written in Swift and/or Objective-C. This results in companies having to double the production cost in developing their application and having to settle for performance deficits in order to make the two applications look and feel the same. Using a cross-platform framework is usually slower than native because it has to bridge itself to the OS thus costing performance. However, Flutter uses its Rendering engine to solve this problem. More over applications, developed using cross-platform frameworks have slower releases whenever the main OS updates. The framework would have to update itself in order to accommodate to the new changes. To tackle this conundrum Google, in August 2017, developed Flutter. Written in Dart programming language, Flutter provides the usability of writing one code that runs on both Android, iOS devices and just recently with its stable 2.0 release Web applications [1]. All this with minimal performance penalty compared to their native programming languages.

- *Sana Rizwan, Assistant Professor, Department of Computer Science COMSATS University Islamabad, Lahore Campus Pakistan. E-mail: sana@cuilahore.edu.pk*
- *Shoaib Uzair, Student of BSCS, Department of Computer Science COMSATS University Islamabad, Lahore Campus Pakistan. E-mail: fa17-bse-012@cuilahore.edu.pk*
- *Zaid Saeed, Student of BSCS, Department of Computer Science COMSATS University Islamabad, Lahore Campus Pakistan. E-mail: fa17-bse-015@cuilahore.edu.pk*
- *Abdullah Naveed, Student of BSCS, Department of Computer Science COMSATS University Islamabad, Lahore Campus Pakistan. E-mail: fa17-bse-018@cuilahore.edu.pk*

2 EASE OF USE

Taking advantage of its cross-platform framework, flutter is being used to develop high-speed applications for mobile devices. Flutter does this by using the OS native widgets instead of sourcing web views, and then rendering it using its high-speed rendering engine to render each view [2]. This helps in the applications looking similar and perform as fast as if they were coded in their respective native languages.

An application written in native code can access the OS features directly as shown below (Figure 1). The operating system either Android or iOS lets the native code access its OEM Widgets [3]. If for example, a code written in Java can access Android Widgets but cannot under any circumstance access the native widgets of iOS (Figure 2). In order to do so cross-platform frameworks such as ReactJS uses a bridge that takes care of the communication with the OS regardless of which it is (Figure 3), but this creates its own problem; a bottleneck. Animations, swipes etc [4]. could slow down the application so Flutter has a slightly different approach to this that gives it a drastic performance edge over its competitor cross-platform rivals. Flutter has a rendering engine that paints over the native widgets that increases performance regardless the OS (Figure 4).

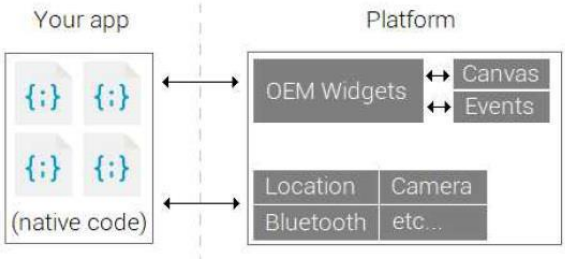


Figure 1 Native application can access the Native Widgets

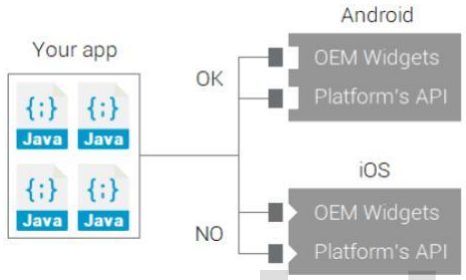


Figure 2 A Java Application can access Android Widgets BUT NOT iOS

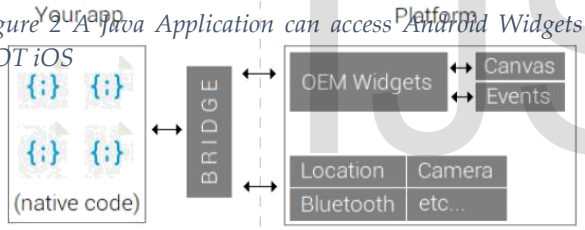


Figure 3 Cross-Platform with a bridge [4]

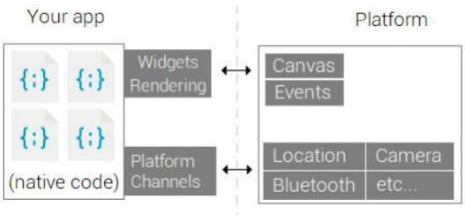


Figure 4 Flutters Skia Rendering Engine

3 FLUTTER ARCHITECTURE

3.1 Dart

In Flutter, every application written in Dart programming language. This language is used extensively within Google and therefore has been developed and maintained by it as well. Originally Dart was developed to replace JavaScript that is why it has most of the important characteristics of that language. It contains Keywords such as 'async' and 'await' [3]. Moreover, as Dart is a modern programming language it provides the developers with better memory optimization with the help of Generational Garbage Collection.

3.2 Flutter

Flutter uses Androids Native Development Kit (NDK) and iOS Low Level Virtual Machine (LLVM) for the compilation process of the Dart code that compiled into Native code. Another feature that Flutter has is Hot Reload. What this does is that it sends the updated code into the already running Dart Virtual Machine (DVM) [2]. This enables the developer to look at changes made to the code on the go without having to compile the code every time after the first compilation of the session.

3.3 Widgets

In Flutter, every object on the screen is a Widget that nested inside another widget. That is because all classes are dependents of the Widget class. Once we nest widgets into each other, we create a hierarchy known as the "Widget Tree" which contains parents and children. Text fields, containers, image boxes, scrolling bars, etc. everything is classified as a widget.

4 FLUTTER AGAINST ITS COMPETITION (NATIVE APPLICATIONS)

4.1 Code

To put in perspective on how the above-mentioned positives play out for Flutter a comparison was carried out in which three source codes were created for on particular application. One was in Flutter the other in Kotlin (Android Native) and Swift (iOS Native). According to Figure 5 we can see that both the lines of code and the number of files needed to code the same application is less in Flutter compared to both Android Native and iOS native [3].

Type	lines of code	Code files that were needed for the application
Android native	217	9
Android Flutter	125	3
iOS native	363	6
iOS Flutter	125	3

Figure 5 Code Comparison

4.2 Development Time

Next comes the coding time. Like mentioned before that Flutter can produce both Android and iOS applications with the same piece of code, it is also faster to produce that particular code. Both native applications have the advantage of using a drag and drop facility but that is useful only till a certain extent, linking that drag and dropped code creates a bit of difficulty to integrate as compared to writing the code out right in the first place. The development time statistics as shown in Figure 6.

Type	Total	Navigation Base	First View	Second View
Android native	12h	3h	2h	7h
iOS native	8h	2h	0.5h	5.5h
Flutter	6h	4h	1h	2h

Figure 6 Development Time

4.3 CPU Performance

Like stated before the direct communication between the Native applications with their respective OS is the quickest but the performance deficit caused by Flutters Rendering Engine Skia is minor [2]. In addition, the average is better than both Native versions as shown in Figure 7.

Type	Language	Highest	Lowest	Mean	Standard Deviation
Native iOS	Swift	92.7%	14.3%	32.9%	13.75360872
Flutter iOS	Dart	101.7%	18.8%	35.3%	18.00680891
Native Android	Kotlin	34.6%	1.0%	11.7%	6.88638675
Flutter Android	Dart	32.3%	1.0%	13.2%	9.29106696

Figure 7 CPU Performance

5 FLUTTER AGAINST REACT NATIVE

Flutter and React Native have little similarities; For starters, two enormous tech companies, Google and Facebook respectively, back both. Both frameworks are open source and free to use. However, that is where they almost end.

The main difference between the two is their performance. Since Flutter does not use a bridge but its own Rendering C++ Engine Skia, it has an upper hand compared to its React Native rival. React Native has to send JSON messages to establish a connection between the source code and OS. Flutter uses in built libraries and Frameworks such as Material Design and Cupertino to establish connection, which is a much more efficient way [2].

Another aspect for better development is documentation. The better documented the framework is the more a developer is attracted to it. Flutter Documentation is user friendly and gives good explanation to the user. This is helpful if the user is a novice. Whereas the React Native 1 is poorly document and is more focus on complicated processes that only people with a good grip with JavaScript (Language React Native is written in java) [3].

Then comes the deployment phase. Applications in Flutter can be publish to the App Store or Google Play with the help of one command line code where as in React Native you need a third-party tool in order to deploy the application. With these little things, Flutter despite being new has gained much popularity [4]. By 2020, Flutter has seen an increase form 3.2pc to 7.2pc since 2019 as shown in Figure 8.

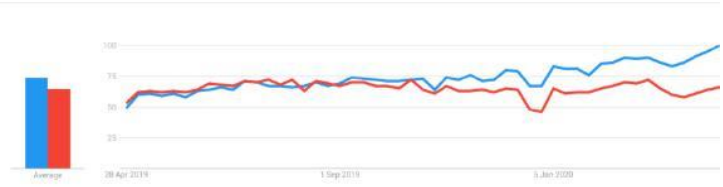


Figure 8 Flutter vs React Native

6 CASE STUDY IMPLEMENTATION AND ANALYSIS

"Click and Pick" android app developed to calculate Flutter and react native with java jdk performance.
The app was about to facilitate customer in a different manner, actually the problems faced with conventional shopping that it was time consuming and tiring for people.

Visiting various stores to find what need and being disappointed when the product want is not available. Then comes the issue of stores being crowded during prime times and in the recent pandemic when people had to maintain social distance conventional shopping was not going to be feasible. Then the problem of online shopping that delivers the items to customer's doorstep also had its issues. Even though getting your order delivered at door seems continent but there is often a delay in delivery, damaged items or sometimes items being lost in transit.
Keeping all of this in consideration, we devised a different approach that was take-aways. If we take a fast-food restaurant as an example, a customer goes to the restaurant, places an order in their vehicle and gets their order. This is fast and convenient. We implied the same concept to Click & Pick. The user places their order on their app; the stores in question pack the order and send it to the collection point. Once their order is ready, the customer just simply drives into the collection point and collects their order.

Functionalities were developed and cost-effective impacts were calculated. Estimating the software cost and price is an important aspect for software development. For efficient and effective project management, the software development cost estimation accuracy will be the major feature for budgeting, tracking, planning, tracing and control.
Before the starting of software engineering process, cost and duration for software project should be settle among customers, financer/s, developers and stakeholders.

7 METHODOLOGY / ALGORITHMIC MODEL

Constructive Cost Model algorithmic model will adapt to find the effort and development time for software project estimations, these estimations will be accurate as it based on formula calculations by the initial calculations of function point analysis scheme (FPA). Formulas for predication effort based on estimate of project size, KLOC (if known) and function point.

Function point analysis (FPA) is use to make estimate of the software project, including its testing in terms of functionality or function size of the software product. However, functional point analysis may be use for the test estimation of the product. The functional size of the product is measure in terms of the function point, which is a standard of measurement to measure the software application.

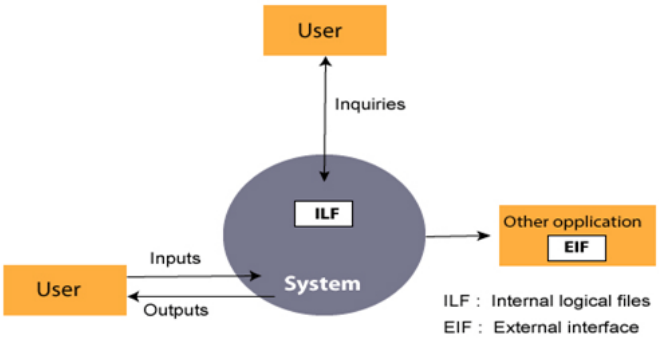
7.1 Objectives of FPA

The basic and primary purpose of the functional point analysis is to measure and provide the software application functional size to the client, customer, and the stakeholder on their request. Further, it will use to measure the software project development along with its maintenance, consistently throughout the project irrespective of the tools and the technologies.

Measurement parameters identified for the calculation of software interacting prototypes in forms of input, output, queries, internal logical operations in OS and interface interaction and command handling procedure that are explained in table 1. These 5 parameters are also called functional units of software system as shown in Figure 9.

Measurements Parameters	Examples
1. External Inputs(EI)	Input screen and tables
2. External Output (EO)	Output screens and reports
3. External inquiries (EQ)	Prompts and interrupts
4. Internal files (ILF)	Databases and directories
5. External interfaces (EIF)	Shared databases and shared routines

Table 1 Measurement Parameters with Examples



FPAs Functional Units System

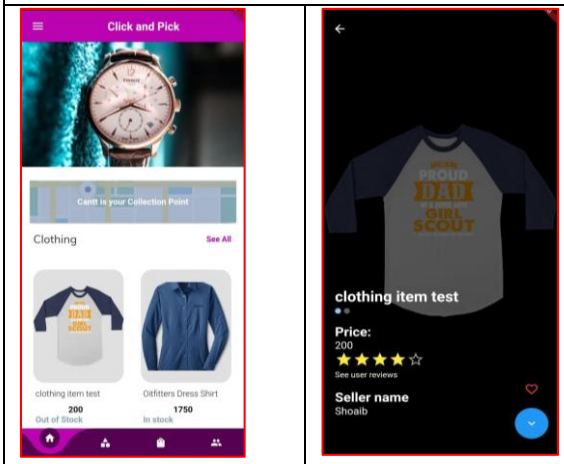
Figure 9 Functional Units system

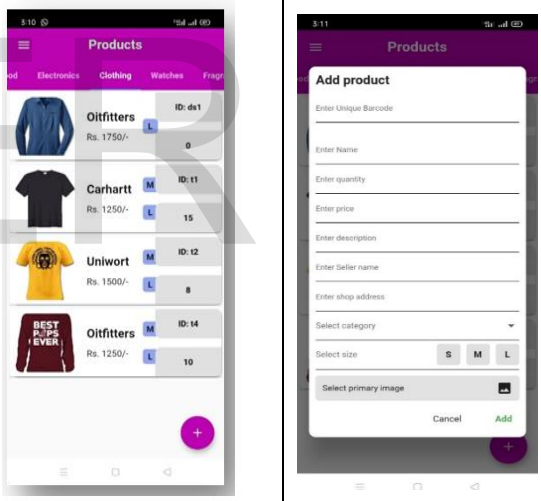
7.2 Functional Prototypes

Software houses and companies can symbolize their requirement for software product, project idea, real time scenario simulation with user experience, find new techniques to discover or improve and spot limitations with functional prototypes [5]. Project budget allocation can identify before the launch of software product after gathering the losses and ignoring the problem commands with the critical analysis of risk details. Although functional prototypes means simply no reason to ignore the actual value of software prototyping. Some are the Functional prototypes with the concern of hi fidelity or low fidelity shown below in Figure 10;

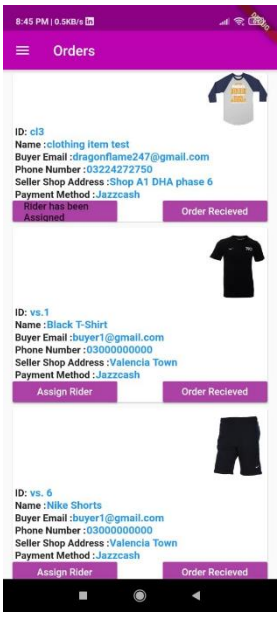
- View product
- Product details
- Product rating
- Reviews
- Shopping cart
- Search
- Select payment method
- Orders
- Add Favorites
- Delivers order
- Product in cart
- Pending orders
- Add new product
- Edit product details

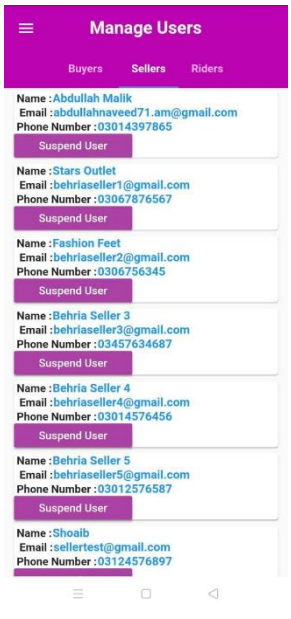
Buyer Screens





Seller Screens





Manager Screens

Figure 10 Functional Prototypes

FP characterizes the quality of the software system depending upon the complexity and thus are often wont to depict the project time and the force demand as needed in the requirements. The effort needed to develop the project depends on what the software system will do.

Function Point is programming language freelance.

This methodology employed for real time software scenarios, processing systems, business intelligent systems like information systems, MIS etc. The 5 parameters mentioned higher than also are referred to as information domain characteristics [6].

All the parameters mentioned above are assigned some weights that have been experimentally determined and are shown in Figure 11 Function Point Table.

Function Point Table

Number of FPs	Complexity		
External user type	Low	Average	High
External input type	3	4	6
External output type	4	5	7
Logical internal file type	7	10	15
External interface file type	5	7	10
External inquiry type	3	4	6

Figure 11 Function Point Table

Object points square measures the simplest way of estimating effort size, almost like supply Lines of Code (SLOC) or perform Function Points [6]. They are not essentially associated with objects in Object-oriented programming, the objects stated embrace screens, reports, and modules of the language. The amount of raw objects and quality of every square measure calculable and a weighted total Object-Point count then computed and want to base estimates of the trouble required in the form of efforts needed.

Screens	**Complexity**
Buyers screen	Simple
Seller screen	Medium
Manager screen	Medium
Pricing screen	Difficult
Product screen	Medium
Sales screen	Medium

Table 2 Weighting Factors – Screen complexity

Reports	**Complexity**
User review report	Medium
Pending order report	Simple
Deliver order report	Simple
Add Favorites report	Medium
Order history report	Medium
Available rider report	Medium
Buyer report	Simple
Manager report	Difficult
Seller report	Medium
Rider report	Medium

Table 3 Weighting Factors – Report complexity

According to the assigned complexity keeping in view the objects of the functional concerns that are simple, medium and difficult [7]. Weighting factors are assign depending upon the screens, reports and 3GL components comparison Table 2 and 3 as mentioned;

Object Point Analysis - Screen

Number of views contained	Number and source of data tables		
	Total < 4 (<2 server, <2 client)	Total < 8 (2-3 server, 3-5 client)	Total 8+ (>3 server, >5 client)
< 3	Simple	Simple	Medium
3 – 7	Simple	Medium	Difficult
8+	Medium	Difficult	Difficult

Figure 12 OPA -- Screen

Object Point Analysis - Reports

Number of sections contained	Number and source of data tables: Total < 4 (<2 server, <2 client)	Total < 8 (2-3 server, 3-5 client)	Total 8+ (>3 server, >5 client)
< 2	Simple	Simple	Medium
2 or 3	Simple	Medium	Difficult
> 3	Medium	Difficult	Difficult

Figure 13 OPA -- Reports

Object Point Analysis – Complexity Weighting

Type of object	Complexity: Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component	N/A	N/A	10

Figure 14 OPA – Complexity Weighting

6.3 Calculate NOP by Weighting Factors

Weighting factors will extract out depending upon the complexity factors depending upon simple, medium and difficult complexity measures as shown in Figure 12, 13, 14. Total number of object points (NOP) [7] will depend upon the total number of weighting complexity i.e 56 in this case as shown in Table 4.

Functional Concerns	**Objects**	**Complexity**	**Weights**
Buyers	Screen	Simple	1
Seller	Screen	Medium	2
Manager	Screen	Medium	2
Pricing	Screen	Difficult	3
Product	Screen	Medium	2
Sales	Screen	Medium	2
User review	Report	Medium	5
Pending order	Report	Simple	2
Deliver order	Report	Simple	2
Add Favorites	Report	Medium	5
Order history	Report	Medium	5
Available rider	Report	Medium	5
Buyer	Report	Simple	2
Manager	Report	Difficult	8
Seller	Report	Medium	5
Extra or Pending rider	Report	Medium	5
Total NOP			**56**

Table 4 Total Number of Object Points

7.4 Calculate Software Productivity and Efforts

Calculate PROD that depends on data receiver past project data, so that ratio between efforts imposed on total number of object points calculated in software project for estimation. Developer past experience and capability and CASE maturity are the major factors by which estimator can predict the productivity rate for effort based software estimations [8].

PROD = NOP/ Person-Months

Object Point Analysis – Productivity Rate

	Very low	Low	Nominal	High	Very High
Developer's experience and capability	4	7	13	25	50
CASE maturity and capability	4	7	13	25	50

Figure 15 OPA – Productivity Rate

In this particular case complexity is nominal of developer experience, whereas high is CASE maturity and capability capture by Figure 15.

Developers experience is nominal = 13
CASE maturity and capability is high = 25

Calculated NOP = 56

Productivity (PROD) = Developers experience and capability + CASE maturity and capability

PROD = 13 + 25 = 38

Average = 38/2 = 19

Effort (Person-Months) = NOP / PROD
= 56 / 19 = 2.94 P-M

Function Point Estimation (FP → KLOC)

Functional Requirements	**External User Types**	**Complexity**	**Function Point**
Buyers screen	External Output	Simple	4
Seller screen	External Output	Medium	5
Manager screen	External Input	Medium	4
Pricing screen	External Input	Difficult	6
Product screen	External Interface Files	Medium	7
Sales screen	External Interface Files	Medium	7
User review report	External Input	Medium	4
Pending order report	External Inquiry	Simple	3
Deliver order report	External Inquiry	Simple	3
Add Favorites report	External Output	Medium	5
Order history report	External Output	Medium	5
Available rider report	External Output	Medium	5
Buyer report	External Output	Simple	4
Manager report	External Output	Difficult	7
Seller report	External Output	Medium	5
Rider report	External Output	Medium	5
Total			**79**

Table 5 Total Function Points Calculation

Total Function Points Calculated in Table 5 = 79

To calculate the numbers of line of code, the following formula is used:

LOC = LOC per FP * FP **OR** **LOC = AVC * FP** where:

- AVC: is the average number of LOC/FP for a given language
- LOC: is the numbers of line of code
- FP : is the Total numbers of Function Point

Programming Language	LOC/FP (average)
Assembly Language	320
C	128
COBOL/Fortran	105
Pascal	90
Ada	70
C++	64
Visual Basic	32
Object-Oriented Languages	30
Smalltalk	22
Code Generators (PowerBuilder)	15
SQL/Oracle	12
Spreadsheets	6
Graphical Languages (icons)	4

Figure 16 Programming Languages KLOCs

- Published Figure 16 for Java Language show that:
  - 1 FP = 30 LOC in java (Object oriented languages) [8]
- Estimated size
  - 79 * 30 = 2370
    = 2 KLOC

Flutter Language with the base java contains 2.94-effort person per months, with estimated 2 KLOC with the concerned weighting factors having 38 productivity rate, therefore software development language will give good chances to new project managers as well as new development team to get more and more experiences with the recurrence procedure in SDLC stack.

7. CONCLUSION

The Flutter framework provides the user with great ease due to its cross-platform characteristics. The developer can write code for both Android and iOS easily with little performance drawbacks compared to its other cross-platform rivals. Not only has that Flutter possessed a highly organized documentation that supports amateur developers to start their projects. This has resulted in people moving towards the platform quickly, and now with the fresh launch of Flutter 2.0, the web version has now a stable release. However, it cannot be demanded that Flutter would always be the best choice. If the requirements of OS specific opting for the Native Framework would be the best way to go. That said with the current trajectory of the growth of Flutter is a promising addition to the application development industry with debating the cost Effectiveness, part of effort requirements and productivity index by emerging development team.

ACKNOWLEDGMENT

The authors wish to thank Department of Computer Science COMSATS University Islamabad, Lahore Campus for the support and help.

REFERENCES

[1] John Wiley & Sons, "Beginning Flutter®: A Hands On Guide To App Development"

[2] Marco L. Napoli, Beginning Flutter: A Hands On Guide to App Development, 2019, ISBN: 978-1-119-55082-2

[3] Adam Boduch, Roy Derks, "React and React Native: A complete hands-on guide to modern web and mobile development with React.js", 3rd Edition

[4] Adam D. Scott, "JavaScript Everywhere: Building Cross-Platform Applications with GraphQL, React, React Native, and Electron", 1st Edition

[5] Anatoly Kotlyar, "The Role of a Functional Prototype in Software Engineering", https://medium.com/cxdojo/the-role-of-a-functional-prototype-in-software-engineering-1588a56ce56e

[6] David Garmus, David Herron, "Function Point Analysis: Measurement Practices for Successful Software Projects (Addison-Wesley Information Technology Series)", 1st Edition

[7] Hareton Leung and Zhang Fan, "Handbook of Software Engineering and Knowledge Engineering—Software Cost Estimation", pp. 307-324 (2002), https://doi.org/10.1142/9789812389701_0014

[8] Software cost estimation COCMO II, Chapter 26, https://ifs.host.cs.standrews.ac.uk/Books/SE7/SampleChapters/ch26.pdf