Estimated Value of Software Developer Productivity at the Software Implementation Stage Using Function Points

Asri Maspupah[1], Lukmannul Hakim Firdaus[2], Wendi Wirasta[2]
[1,2,3] Jurusan Teknik Komputer dan Informatika, Politeknik Negeri Bandung, Bandung, Indonesia
[1] asri.maspupah@polban.ac.id, [2] lukmannul.hakim@polban.ac.id, [3] wendi.wirasta@polban.ac.id

Accepted 10 October 2022
Approved 28 November 2022

***Abstract*— Most Software Development Processes (SDP) project failures occur due to errors in estimating the cost, time, and effort during the planning stage. This happened because the planning still relied on the intuition and experience of the programmer. One approach that can be taken to plan the right SDP is to know the value of SDP productivity. The focus of this research was to determine the value of productivity based on the differences in programmers' skills. This case study was conducted to determine the productivity value of the web-based software that has been built, namely McDelivery. The productivity value was calculated based on the ratio of software size to effort. In this case, the software size was obtained by calculating the Application Function Point Count (APFC). Meanwhile, the effort was obtained from expert judgment to determine the time needed by the development team at the junior, middle, and senior software developers to implement software functionality in person-day to the form of program code. The result showed that the productivity value of SDP was directly proportional to the level of ability of the programmers. These productivity values can be used as a solution option to calculate the estimated time, cost, and even the availability of programmers that were adjusted to the conditions faced in planning software development.**

***Index Terms*— application function point count; function point analysis; productivity metrics; software development process; software developer**.

I. Introduction

The accuracy of activity planning determines the success factor of the software development process (SDP) project at the beginning of a development which is directly proportional to the realization of the implementation of activities at the end of time [1]. This accuracy is indicated by the software meeting the requirements within a period and incurring reasonable and planned costs [2]. Most SDP project failures occurred due to incompatibility of planning with actual implementation. Generally, the cause of the discrepancy lies in the estimation of cost, time, and effort [3]. Meanwhile, the estimated effort is used to plan and calculate the software development costs.

Based on the research conducted by Usman in 2015 concerning the measurement of the accuracy of planning estimates with the implementation in agile software development through a survey of 60 companies showed that approximately 78.33% of the companies stated that there was an inaccuracy estimation between the implementation and the planning [4]. The inaccuracy estimation occurs when the implementation of software development exceeds the planning estimate (60%) or solves faster than the planning estimate (18.33%). One of the contributing factors is that software development planners need to think about the best scenario based on the development team's ability to deal with software complexity. In addition, planners also need to pay more attention to the software development effort.

Proper SDP planning can be done by considering realistic costs, time, and effort [5]. However, as SDP planners, IT project managers have difficulty in making accurate effort estimates [6]. In general, the calculation of each functional software's development effort depends on intuitive experience, which is a subjective assessment of the SDP planner. In this case, development effort is defined as person-hour working on several SDP activities [7]. At the same time, the determination of person-hours depends on the software developer's ability level and the work's difficulty [8].

The accuracy of SDP planning is related to productivity metrics because the value of productivity can measure the development process's effectiveness at the project's end [9]. In general, productivity is defined as the ratio between input and output. Input is a resource to produce output. This definition is very suitable for the manufacturing industry because it clearly shows the quality standard of input and output measurement units [8]. In the field of software engineering (SE), the term productivity refers to the effectiveness of development project efforts measured by the output rate per unit of software [10] [7].

The calculation of the SDP productivity value is generally done twice; those are at the beginning and the end of the project. The difference in productivity values shows the inaccurate prediction of project planning effort estimates. However, using historical SDP productivity data from previous projects in similar software developments can increase the accuracy of project planning parameter estimates (effort, time, and cost) [9].

On the other hand, most IT projects have used productivity metrics to measure SDP productivity based on the comparison between software size (output) and effort (input) required in developing software [11]. In this case, the accuracy of the productivity value of the productivity metric must consider three criteria: (1) the scope of the resource; (2) the scope of input; and (3) the scope of the output to be calculated [12]. Thus, practitioners and academics must carefully know the scope of effort and software size that will be useful for measuring the SDP productivity.

Most previous researchers used time as the definition of effort in measuring SDP productivity. However, the definition of productivity in the SE concept should focus on the level of complexity of work done by each person [8]. Using time as a criterion for measuring productivity inputs can lead to the question, what time should be used? In this case, the time is the duration spent by the person doing a job or time paid (contract) person within the worked hour range. On the other hand, each person has a different level of ability, so the level of productivity in completing the work is also different. In the context of SE, software developers' ability can affect the productivity level in software development [8]. Meanwhile, software metrics can quantify software size as a productivity output criterion [2]. Software metrics as software measurement standards aim to get a value for the size of software complexity [13].

In previous research, the researchers proposed several software metrics, including line of code (LOC), constructive cost model (COCOMO), and function point analysis (FPA). LOC is the most straightforward software metric using the actual line code as a criterion [2]. However, LOC is so dependent on programming languages and development technologies that it cannot be used to measure the productivity of non-technical activities and is challenging to be measured at the beginning of development [14][2]. Furthermore, COCOMO uses a mathematical formula to determine software development efforts. COCOMO involves line code information and justification of development efforts by domain experts [15].

Meanwhile, FPA uses the requirement specification functionality as the basis for measurement. The FPA calculation uses the standard method to measure software engineering based on the scope of the software. Thus, the FPA calculation, regardless of technology and programming language, is more straightforward and meaningful from the end user's point of view [16] [15].

Among the various variations of software metrics, FPA is the most commonly used approach [5]. Alan J. Albrecht introduced FPA from IBM in 1979 [3]. At the end of the FPA measurement process, the software has a function point (FP) value. In this case, FP shows the value of software functionality as the basis for successful product delivery to end users [8]. Thus, FP is a unit of software size for software development analysis. For example, realistic cost estimation, measurement of SDP productivity value based on the ratio of effort spent on each FP, and measurement of software quality based on the ratio of the number of defects found in each FP [17].

Based on the explanation above, this study aimed to obtain the value of software developer productivity at the software implementation stage. The focus of the study was the analysis of effort and software size from the side of software developers with different levels of programming ability in the case of software development from similar applications that already exist. In this case, the SDP productivity measurement parameters were Effort and software size. The definition of the software developer's programming ability specifications is through the justification of domain experts, namely software developer experts with 12 years of experience as a team leader of software developers. Furthermore, the value of software development productivity is used as an indicator of time and cost estimation so that SDP planning becomes realistic. In addition, the productivity measurement parameters used time-person as input and FP as output. At the same time, the selection of FP is a unit of software size because it is more appropriate to use functional measurement software independent of technology to be used as a parameter for calculating SDP planning.

II. METHODS

The research stage started with the analysis of problems in software coding productivity. The next stage was selecting a productivity measurement method—finally, the measurement of productivity in this case was carried out through an experiment. Figure 1 shows the research flow.

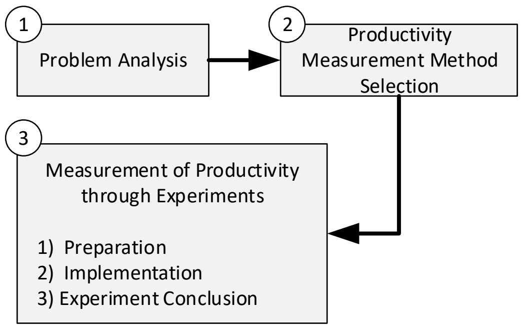
1
Problem Analysis
2
Productivity Measurement Method Selection
3
Measurement of Productivity through Experiments
1) Preparation
2) Implementation
3) Experiment Conclusion

Fig. 1. The Research Stage

The productivity measurement method used productivity metrics. Then, the input and output terminology were mapped into the software development process. Furthermore, the product output calculation used FPA, while software coding input used calculating effort (person-day) required by the developer. The productivity measurement experiment consisted of 3 stages. First, experimental preparation including preparing the case studies of products to be built and mapping the developer programming skills into three levels (junior, middle, and senior). Second, the implementation of experiments to calculate the input, output, and productivity of the implementation software stage. Third, the conclusion of the experimental results, which was to analyze the productivity value at each developer level.

A. Function Point Analysis (FPA)

A measure of software complexity is the output produced during software development. Furthermore, the output was used as a parameter to generate productivity values by comparing the effort when carrying out activities. Software complexity becomes a number through software measurements. FPA is a method used to measure the complexity and functionality of software in SDP projects [2] [18].

The method used in FPA was dividing the size of the software into smaller components so that it was easier to analyze [3]. The critical value in the FPA measurement is the calculation of software function based on five standard functions set by the International Function Point Users Group (IPPUG), namely internal logical files (ILF), external logical files (EIF), external input (EI), external output (EO), and external inquiries (EQ) [5]. In this case, the FP calculation process started from a high level by analyzing the software functionality specifications. It consisted of six steps, namely (1) determining the type of FP count; (2) identifying the software scope; (3) weighing the software based on standard function; (4) calculating the unadjusted function point (UFP); (5) justifying the value adjustment factor (VAF), and (6) calculating the adjusted function point (AFP) [19] [3].

Determining the type of FP Count

The calculation of software size depends on the purpose of the type of software to be analyzed. In 2010, IFPUG divided the types of FP calculations into three categories. First, the development project function point count (DPFPC) is the type of FP count intended for software developed for the first time and released to end users. Second, enhancement project function count (EPFC) is the type of FP count intended for software developed in adaptive maintenance projects. Adaptive maintenance projects aim to improve performance and implement change requirements from end users that must be matched in the first stage of development. Third, application function point count (AFPC) is the type of FP count intended for software developed on existing software products. However, adding new functionality are needed when there is changing requirements from end users.

This study used APFC to calculate software size with the formula shown in equation 1.

$$AFP = ADD \times VAF \qquad (1)$$

Information:

- AFP : *application project function point count*
- ADD : *unadjusted function point count from application functionality added to an existing application*
- VAF : *value adjustment factor*

1) Identifying the scope and limitations of the software

The scope and limitations of the software form the basis of software development. The scope defines a set of software functionality that includes data, screens, and reports. At the same time, the limitation of software as an interface between software and end users defines things outside the software's scope. Thus, determining the scope and limitations of the software can provide information on the size of the software based on functionality at the end of development so that the FP count can be carried out [20]. The steps for determining the scope and limitations of the software are:

a) defining a set of sub-processes within the scope of the software.

b) understanding the purpose of measuring the FP count on the software.

c) defining the software process flow in managing data into information.

d) defining business areas to support each process in the software.

e) defining logical data both within the scope of the software and logical data originating from outside the scope of the software.

2) Identifying the scope and limitations of the software

Unadjusted function points show the value of software complexity by weighing each functional requirement based on five standard functions [3] [2]. The weighting of software complexity consists of three categories, namely low (L), average (A), and high (H) [20]. Meanwhile, the five standard functions are divided into data and transactional [20] [5].

The data function shows software functionality on internal and external data storage requirements. Data functions include logical data in software (ILF) and external interface files that connect data from outside with internal software (EIF). The transactional function shows the software's functionality in processing data when there is an interaction between the software and the end user. The transaction process includes transactions receiving input data (EI), displaying output (EO), and querying data (EQ) [20].

The weighting of complexity in each standard function is conducted based on the functional software's RET, DET, and FTR values. In this case, RET (record element type) is a subgroup of data elements in data storage, DET (data element type) is an attribute in data storage and application, and FTR (file type reference) is a type of data files read/managed in a transaction. The data function weighting matrix involves RET and DET values, while the transactional function weighting matrix involves FTR and DET values [5].

Based on the explanation above, the calculation of the unadjusted function point is divided into two parts: the standard weighting of the function and the calculation of all weight values.

3) *Weighing the Standard Function*

a) *External Input (EI)*

EI is an elementary software process related to receiving data from outside the system so that changes in software behavior and changes in ILF data occur. Examples of EI are input data from end users or other systems. Table 1 shows the EI complexity matrix and the weight value for each level of complexity.

TABLE I. EI COMPLEXITY MATRIX

Number of File Type Reference (FTR)	**Number of Data Element Type (DET)**		
	1-4	**5-15**	**> 16**
0 - 1	Low (3)	Low (3)	Average (4)
2	Low (3)	Average (4)	High (6)
> 2	Average (4)	High (6)	High (6)

b) *External Output (EO)*

EO is an elementary software process that sends data from inside to outside the system. The logical process of retrieving data from within the system contains at least one of the processes between the mathematical calculation process, the process of making derived data, and changing the data of one or more ILFs. An example of EO is creating an output file sent to another system [20]. Table 2 shows the EO complexity matrix and the weight value for each level of complexity.

TABLE II. EO COMPLEXITY MATRIX

Number of File Type Reference (FTR)	**Number of Data Element Type (DET)**		
	1-5	**6-19**	**> 19**
0 – 1	Low (4)	Low (4)	Average (5)
2 – 3	Low (4)	Average (5)	High (7)
> 3	Average (5)	High (7)	High (7)

c) *External Inquiries (EQ)*

EQ is an elementary software process that sends data from inside to outside the system. The difference between EQ and EO lies in data collection. EO does not create data from the mathematical calculation and derived data process [20]. Table 3 shows the EQ complexity matrix and the weight value for each level of complexity.

TABLE III. EQ COMPLEXITY MATRIX

Number of File Type Reference (FTR)	**Number of Data Element Type (DET)**		
	1-5	**6-19**	**> 19**
0 – 1	Low (3)	Low (3)	Average (4)
2 – 3	Low (3)	Average (4)	High (6)
> 3	Average (4)	High (6)	High (6)

d) *Internal Logical File (ILF)*

ILF is a logical group of corresponding data within the software scope and managed by one or more leading software processes. An example of an ILF is a table in a relational database and a collection of files stored in an application [20]. Table 4 shows the ILF complexity matrix and the weight values for each level of complexity.

TABLE IV. ILF COMPLEXITY MATRIX

Number of File Type Reference (FTR)	**Number of Data Element Type (DET)**		
	1 - 19	**20 - 50**	**> 50**
1	Low (7)	Low (7)	Average (10)
2 – 5	Low (7)	Average (10)	High (15)
> 5	Average (10)	High (15)	High (15)

e) *External Output (EO)*

EIF is a logical group of interrelated data from outside the scope of the software and managed by one or more of the leading software processes. Logical data EIF is a source of reference data by the software being measured [20]. Table 5 shows the EIF complexity matrix and the weight values for each level of complexity.

TABLE V. EIF COMPLEXITY MATRIX

Number of File Type Reference (FTR)	**Number of Data Element Type (DET)**		
	1 - 19	**20 - 50**	**> 50**
1	Low (5)	Low (5)	Average (5)
2 – 5	Low (5)	Average (7)	High (10)
> 5	Average (7)	High (10)	High (10)

4) *Calculating Unadjusted Function Point (UFP)*

The calculation of the UFP value is carried out by adding the weights of EI, EO, EQ, ILF, and EIF, which are calculated based on the complexity value of the software functionality [3]. Table 6 shows the process of calculating UFP.

5) *Justifying the Value Adjustment Factor (VAF)*

Value adjustment factor (VAF) is a set of factors that affect software complexity [20]. VAF uses standardized questions of general system characteristics (GSCs) to assess general characteristics of software functionality. GSCs have 14 characteristics that reflect the degree of influence of requirements on functional software. The VAF value is calculated based on the justification of the domain expert who knows the software domain by giving weight to each characteristic between the ranges of 0 (not essential) to d. 5 (very important) [3]. Table 7 shows a list of VAF questions.

Furthermore, the TDI value was used in calculating the VAF value using Equation 2.

$$VAF = 0.65 + (TDI \times 0.01) \quad (2)$$

Information:

- VAF : *value adjustment factor*
- TDI : *total degree of influence*

TABLE VI. UNADJUSTED FUNCTION POINT (UFP) CALCULATION [3]

Standard Function	Software Complexity			
	Low (L)	Average (A)	High (H)	Total
External Input (EI)	__ x 3 =	__ x 4 =	__ x 6 =	Total wight of EI
Enternal Output (EO)	__ x 4 =	__ x 5 =	__ x 7 =	Total wight of EO
External Inquries (EQ)	__ x 3 =	__ x 4 =	__ x 6 =	Total wight of EQ
Internal Logical Files (ILF)	__ x 7 =	__ x 10 =	__ x 15 =	Total wight of ILF
External Interface File (EIF)	__ x 5 =	__ x 7 =	__ x 10 =	Total wight of EIF
Unadjusted Function Point (UFP)				The sum of weight EI EO, EQ, ILF, EIF

TABLE VII. UNADJUSTED FUNCTION POINT (UFP) CALCULATION [3]

No.	Characteristics	Question	Degree of Influence (DI) *)
1.	Data communications	How many communication facilities are there to aid the transfer or exchange of information with the application or system?	_______
2.	Distributed data processing	How are distributed data and processing functions handled?	_______
3.	Performance	Did the user require response time or throughput?	_______
4.	Heavily used configuration	How heavily used is the current hardware platform where the application will be executed?	_______
5.	Transaction rate	How frequently are transactions executed daily, weekly, monthly, etc.?	_______
6.	On-Line data entry	What percentage of the information is entered On-Line?	_______
7.	End-user efficiency	Was the application designed for end-user efficient?	_______
8.	On-Line update	How many ILF's are updated by On-Line transaction?	_______
9.	Complex processing	Does the application have extensive logical or mathematical processing?	_______
10.	Reusability	Was the application developed to meet one or many user's needs?	_______
11.	Installation ease	How difficult is conversion and installation?	_______
12.	Operational ease	How effective and/or automated are start-up, back up, and recovery procedures?	_______
13.	Multiple sites	Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?	_______
14.	Facilitate change	Was the application specifically designed, developed, and supported to facilitate change?	_______
Total Degree of Influence (TDI)			Σ DI$_{1\text{-}14}$

6) *Calculating Adjusted Function Point (AFP)*

Adjusted function point (AFP) is the final value of FP as the value of software complexity calculated based on the type of software [20]. AFP calculation was done by using equation 1 formula with APFC software type.

B. *Software Developer Specification*

Software developers are experts who are engaged in developing software. A software developer career in software engineering consists of three levels based on programming skills: junior, middle, and senior [21]. These levels reflect the specifications of the software developer, who can show the responsibilities, qualifications, and amount of take-home pay, as well as the level of productivity in program coding.

- A junior software developer is a developer who has experience developing software for 1-3 years and is familiar with 1 or 2 programming languages/ development frameworks as well as basic programs, such as programming structures, ACID attributes (atomicity, consistency, isolation, and durability) databases, data transactions in databases, and the basis of database design [21][22].
- A middle software developer is a developer who has experience in developing software for 3-5 years, mastering 2 or 3 programming languages/ framework development, programming with reasonably high complexity, able to work as a problem solver, and able to perform proper debugging but has not been able to make appropriate technology decisions [21] [22]. At this level, the developer is suitable for software development, has enough experience working in the field of software development, and is usually quite proficient at being a full-stack developer (backend, frontend, and database).
- Senior software developers have experience in developing software for at least five years, master new programming languages, adapt quickly, and work as problem solvers by providing the best solutions [21] [22]. Senior software developers generally analyze problems that have not occurred, then take preventive measures by preparing the right technology during software development. At this

level, the developer has experience as an expert in the world of work.

The software developer specifications were further used as the research object to determine the productivity value during software implementation in SDP. Furthermore, the programming skills in software development at each level of software developer were mapped through domain expert interviews.

C. Analytical Hierarchy Process

SDP productivity measures software developer performance by calculating the comparison ratio between software size, the product produced, and the effort spent producing the product (Adrián, 2015). Figure 1 shows an illustration of the SDP productivity model.

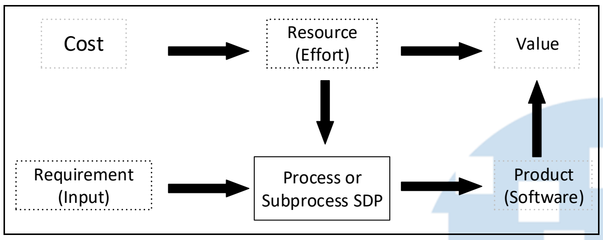

Fig. 2. Productivity Model [9]

Figure 1 is a productivity diagram that shows that there is effort as a resource needed to produce software on SDP. Thus, the measurement of SDP productivity used the Equation 3.

$$Productivity = \frac{Software\ Size}{effort} \tag{3}$$

Software size is a software size calculated based on the value of software complexity. Meanwhile, effort is the time a developer takes to produce software. However, equation 3 still used general parameters because the software size and SDP effort values can be calculated using various approaches, as shown in Tables 8 and 9.

TABLE VIII. SOFTWARE SIZE MEASUREMENT [10]

Size	Parameter	Units of measurement
Task	Number of: Classes, Modifications, Modifications Request, Module Modifications, Modules, Work Items, Pages, Requirements	#Classes, #Modifications, #Modules, #WorkItems, #Pages, #Requirements
EP	FP, CFP, EFP, S, Code Size, OOmFPWeb, UFP, OOFP, SM	function points
LOC	LOC, KLOC, KSLOC, SLOC, ELOC, NLOC, AvgLOC, WSDI, SLC, KNCSS, LOC added, S, SL L, CP, Size, Total Churn, NCLOC, Code Contribution	lines of code

TABLE IX. INPUT MEASUREMENT [10]

Size	Parameter	Units of measurement
person	developer	person
cost	C, man-cost	person-cost
time	hour, T, time, minute, dav-time, time-month, month, cycle-time, year	hour, minute, month, year
effort	developer-hour, developer-quarter, developer-year, E, Eft, engineering-month, H, man-day, man-hour, man-month, effort, PD, PH, man-quarter, PM, man-project-time, SM, person-days, person-month, staff-hour, Staff-month	person-hour, person-day, person-month, person-quarter, person-year

Based on the table above, the SDP productivity parameters used function points as the value of software complexity and person-day as a unit of effort. The selection of person-day as an effort parameter because the FP that is done at one time is easier to analyze in units of days. In addition, planning a software implementation schedule by a domain expert on a feature with low complexity must be done at least one day before making the program. It is necessary to understand software functionality so that there is time allocation for unexpected conditions.

Function points were calculated through FPA, while person-days were determined based on the software developer's programming ability in software development. Thus, the calculation of the SDP productivity value used Equation 4.

$$Productivity\ SDP = \frac{FP}{Person-Day} \tag{4}$$

Description:

- FP : *function point,* a measure of software complexity
- Person-Day : total working time (days) per software developer in implementing each functional software.

The SDP productivity measured was the programming ability of software developers at different levels in software implementation, from design to program code. Furthermore, the productivity value was compared to get the percentage level of speed of software implementation on the same FP software.

III. RESULTS AND DISCUSSION

A. Case Study

The focus of the study was FP calculations on software development from similar applications, namely the McDelivery application, which can be accessed at the link https://www.mcdelivery.co.id/id/. McDelivery is a web-based application used for ordering food and paying for restaurants. The application has simple software functionality, such as user authentication, viewing data, inserting data, updating data, deleting data, and validating transactions. In addition, services come from outside the system, such as payment gateways during payment processing and location coordinates from the Google API. These two characteristics are the basis for choosing McDelivery as a case study describing FP as software size calculation. The scope of the McDelivery application is divided into two parts, namely the ordering process and the payment process with the software functionality shown in Table 10.

TABLE X. MCDELIVERY FUNCTIONALITY SOFTWARE

Feature Code	Software Functionality	Specification Details
FR01	Homepage	The interface displays general information about the ordering system.
FR02	The page starts ordering with a page in the form of a pop-up order box	The order registration process interface consists of three ways: • Method 1. Log in for customers who have registered a user. • Method 2. Registration for customers who do not have an account and are visiting the website for the first time. Orders are made using a personal identity stored on the website. • Method 3. Order food with guest status for customers who want to place an order without register.
FR03	Registration page	The account registration process on the website.
FR04	Delivery address input page	The interface for filling out the order delivery address form which includes the process: a. filling in the order field. b. displaying the delivery location map.
FR05	Menu package list page	The interface displays a food menu catalog, and there is a process for adding menus to the order cart.
FR06	Early message pop up	The order process ahead of delivery time.
FR07	Food ordering page	The food ordering process includes the process: a. displaying menu list. b. viewing order list information. c. completing the order.
FR08	Order details page	The food ordering detail process interface includes the process: a. viewing detailed menu information. b. placing an order by inputting the number of orders on the selected menu and entering unique request data. c. calculating the total price of the order. d. adding order to cart.
FR09	Order overview page	The interface displays an order summary which includes the process: a. displaying a list of order details. b. entering unique record data. c. displaying a list of payment bills. d. entering captcha code.
FR10	Payment page	The interface displays the payment type for processing, including a. Entering the payment method by selecting the available payment types. b. Entering the delivery contact. c. showing a list of bills including order code. d. confirmation of order data.
FR11	Payment processing page	The order bill payment process, including the process: a. displaying the billing list and buyer contact. b. displaying a choice of payment methods (debit or gopay). c. processing the payment using the debit or gopay method according to the selected payment method.
FR12	Order confirmation page	The successful order confirmation to the customer includes the process: a. displaying information on successful payment and display shipping address. b. customer can track order. c. customer can add order to favorite list. d. sending an email to provide information about successfully placing an order and displaying the details of the order list.
FR13	Order tracking page	The interface displays the order status after the order is received by McDelivery. There are four order statuses, namely orders received, in process, being delivered, and sent.
FR14	Order page failed	The interface displays an order failed message when the booking time exceeds the order limit of 30 minutes. The failed booking page contains the following order failed information dan cancel order button. The system displays the close application page when the customer presses the cancel order button.
FR15	Great offers page	The interface displays a list of promos offered by McDelivery at the time of food/beverage purchases.
FR16	Website headers	Navigation links are located in the header section at the top of each website page.
FR17	Website Footer	Navigation links are located in the footer at the bottom of each website page.
FR18	Order sidebar	The navigation link on the side is for displaying the menu list category of the food/beverage ordering page.
FR19	Terms and conditions page	The interface displays a list of terms and conditions for placing an order from the menu offered by McDelivery.
FR20	Privacy policy page	The interface displays policies and privacy as long as the customer makes a menu order on the system.
FR21	Question and answer page	The interface displays a list of questions and answers about how to order menus on the McDelivery information system.

B. *Software Size Calculation*

Software size was calculated using the FPA method, which consisted of 3 stages. FP calculations on a McDelivery system are described below.

1) *Unadjusted Function Point (UFP) Calculation*

UFP was used to see software complexity by weighing the software functionality against five standard functions based on process logic design. Table 12 shows the software's weighting results, while Table 13 shows the results of the calculation of UFP.

2) *Value Adjustment Factor (VAF) Justification*

VAF justification was carried out by a senior developer with 12 years of experience developing web-based software who knows the logic's complexity in implementing software functionality into a program code. Table 11 shows the results of VAF justification for GSCs by domain experts.

TABLE XI. NORMALIZED DECISION MATRIX

Serial Number	Criteria				
	C1	C2	C3	C4	C5
A1	0,1672	0,1689	0,4628	0,4036	0,3404
A2	0,2326	0,1639	0,4474	0,5340	0,3341
A3	0,3126	0,4273	0,1645	0,2607	0,3530
A4	0,0872	0,2832	0,1645	0,2607	0,4224
A5	0,3126	0,3329	0,3291	0,2794	0,0756
A6	0,1672	0,4273	0,2725	0,2607	0,2017
A7	0,3126	0,4819	0,2314	0,1490	0,2900
A8	0,3853	0,2782	0,1645	0,2670	0,2837
A9	0,3199	0,2236	0,1182	0,2607	0,2017
A10	0,5816	0,1689	0,5040	0,3290	0,4665

Furthermore, the calculation of VAF used the equation 2.

$$VAF = 0.65 + (TDI \times 0.01)$$

$$= 0.65 + (42 \times 0.01) = 1.07$$

3) *Adjusted Function Point (AFP) Calculation*

The FP value used the equation on the type of APFC software development using the 1 equation.

$$AFP = ADD^{*} \times VAF$$

$$AFP = 496 \times 107 = 530{,}72$$

*) The ADD value was taken from the results of the UFP value in Table 12

C. *Software Size Calculation*

Determination of software developer specifications on programming skills used interview techniques to domain software development experts who have positions as senior software developers. The interview technique aimed to justify the effort as SDP productivity input parameters have a domain scope consistent with developer specifications at each level.

The domain expert has experience developing software on various software functionalities, such as software development on multi-platforms (web, mobile, and desktop), data retrieval into excel and pdf files, coding with scheduled running (schedulers), application programming interfaces (API), and creation of user interfaces in web and mobile form. In addition, the domain expert can map human resources (HR) into software development projects based on the level of software complexity and software developer programming skills. Table 14 shows the results of the analysis of software developer specifications at each level of programming ability.

TABLE XII. WEIGHTING THE FUNCTIONALITY OF MCDELIVERY SOFTWARE

Software Functionality	EI			EO			EQ			ILF			EIF		
	FTR	DET	W	FTR	DET	W	FTR	DET	W	FTR	DET	W	FTR	DET	W
FR01	3	49	H	0	0	L	3	3	L	6	16	A	0	0	L
FR02	1	12	L	0	0	L	1	2	L	1	2	L	0	0	L
FR03	1	31	A	1	4	L	1	28	A	4	13	L	0	0	L
FR04	2	13	A	0	0	L	2	10	A	1	8	L	1	6	L
FR05	1	15	L	0	2	L	1	8	L	3	7	L	0	0	L
FR06	1	7	L	2	5	L	1	9	L	2	6	L	0	0	L
FR07	1	17	A	1	5	L	2	35	H	2	21	A	0	0	L
FR08	1	14	L	2	6	A	1	32	A	3	15	L	0	0	L
FR09	2	14	A	2	5	L	2	16	A	1	19	L	0	0	L
FR10	2	14	A	2	2	L	2	13	A	2	21	A	0	0	L
FR11	2	27	H	0	0	L	2	57	H	4	15	L	1	1	L
FR12	1	5	L	0	0	L	1	22	A	4	14	L	0	0	L
FR13	1	2	L	0	0	L	1	8	L	1	6	L	0	0	L
FR14	0	2	L	0	0	L	0	4	L	0	0	L	0	0	L
FR15	1	4	L	0	0	L	1	11	L	1	6	L	0	0	L
FR16	1	9	L	0	0	L	1	7	L	2	4	L	0	0	L
FR17	1	17	A	0	0	L	1	8	L	3	7	L	0	0	L
FR18	1	3	L	0	0	L	0	11	L	2	7	L	0	0	L
FR19	1	4	L	0	0	L	0	4	L	2	5	L	0	0	L
FR20	1	4	L	0	0	L	1	4	L	2	5	L	0	0	L
FR21	1	7	L	0	0	L	1	10	L	3	6	L	0	0	L

*) W = Weight of Level Complexity Software

TABLE XIII. UFP MCDELIVERY CALCULATION RESULTS

Standard Function	Software Complexity Value			
	Low (L)	Average (A)	High (H)	Total
External Input (EI)	13 x 3 = 39	6 x 4 = 24	2 x 6 = 12	75
Enternal Output (EO)	20 x 4 = 80	1 x 5 = 5	0 x 7 = 0	85
External Inquries (EQ)	13 x 3 = 39	6 x 4 = 24	2 x 6 =12	75
Internal Logical Files (ILF)	18 x 7 = 126	3 x 10 = 30	0 x 15 = 0	156
External Interface File (EIF)	21 x 5 = 105	0 x 7 = 0	0 x 10 = 0	105
Unadjusted Function Point (UFP)				496

TABLE XIV. SOFTWARE DEVELOPER PROGRAMMING SPECIFICATIONS

Level	Category	Programming Ability
Junior	Algorithm understanding	• basic validation logic, namely mandatory, field format, data type, and alignment. • regular operating business. • basic algorithm structure, namely sequence, selection, and repetition. • complex algorithm structure, namely nested if with two levels, nested repetition with 2-3 levels, and combination of if and repetition with two levels.
	Coding	• reading the source code process flow. • creating program code according to software functionality. • proprietary bug fixing program code. • implementation of functions according to the development framework. • understand aspects of clean code.
	Query database	managing database with data definition language and data manipulation language, and retrieve data with the complexity of two tables.
	Technology exploration	installing and adding plugin tools.
	Software testing	self-testing, unit testing, and code quality checker.
Middle	Algorithm understanding	middle developers have all the algorithm understanding abilities of junior-level software developers, business validation, and the use of algorithm structures with a complexity of 3 to 5 levels.
	Coding	middle developers have all the coding skills of a junior-level software developer, bug fixing in other people's code, and can create effective code.
	Query database	middle developers have all the skills to make database queries owned by junior-level software developers, manage databases with additional data control languages, master PL/SQL and retrieve data with a complexity of 3 to 5 tables.
	Technology exploration	installing, adding plugin tools, and modifying tools.
	Software testing	self-testing, unit testing, and code quality checker.
Senior	Algorithm understanding	senior developers have all the algorithm understanding abilities of medium-level software developers and use algorithm structures with six levels of complexity to infinity.
	Coding	senior developers have all the coding skills of medium-level software developers, review the creation of effective program code structures following the software development framework, and can create development frameworks.
	Query database	senior developers have all the capabilities to make database queries owned by medium software developers and retrieve data with six levels of complexity to infinity.
	Technology exploration	senior developers have all the capabilities of a medium-level software developer and decide on the right technology according to the problem domain.
	Software testing	self-testing, unit testing, and code quality checker.

D. SDP Productivity Calculation

The effort to calculate the SDP productivity value is the time required for software developers to complete program coding of software functionality in person-day units. It was assumed that work effort for one day is 8 hours. The determination of effort is known through justification by domain experts based on the software developer's level of software complexity and programming ability.

Based on the detailed specifications of the software functionality in Table 10, the SDP productivity value for each software developer is shown in Table 15. The effort value of each software developer can be determined from the programming ability and work attitude so that each developer at the same level of programming ability has a different effort value. However, in this study, the effort justification process only involved programming skills, while the ability-to-work attitude was considered the same.

TABLE XV. SOFTWARE DEVELOPMENT EFFORT

Feature Code	Software Developer Effort (person-day)		
	Junior	Middle	Senior
FR01	3	2	2
FR02	5	3	3
FR03	4	3	3
FR04	4	3	2
FR05	2	1	1
FR06	1	1	1
FR07	3	2	2
FR08	5	4	3
FR09	3	2	2
FR10	2	1	1
FR11	5	3	3
FR12	2	2	2
FR13	2	1	1
FR14	3	2	2
FR15	3	2	2
FR16	3	2	2
FR17	1	1	1
FR18	2	1	1
FR19	1	1	1
FR20	1	1	1
FR21	1	1	1
Total Effort	56	40	37

Furthermore, SDP productivity was calculated using equation 4 in each software developer specification. The software size parameter uses FP, worth 530.72, while the effort value is based on the total effort per person-day. The value of software developer productivity on SDP is the number of function points that can be worked on for one day to create programs based on requirements specifications and software design. Table 16 shows the results of the calculation of productivity.

TABLE XVI. EFFORT SOFTWARE DEVELOPMENT IN CODING STAGE

Software Developer Level	Effort (person-day)	Productivity (FP/person-day)
Junior	56	9.87
Middle	40	13.27
Senior	37	14.34

Table 16 illustrates that junior software developers can implement 9 FPs in one day, middle software developers implement 13 FPs in one day, and senior software developers implement 14 FPs per day in the case of the McDelivery ordering system. Based on the effort given in each software developer specification in the case of APFC software development, it can be concluded that the production value is directly proportional to the level of programming ability. This is in line with the visualization of Figure 3.

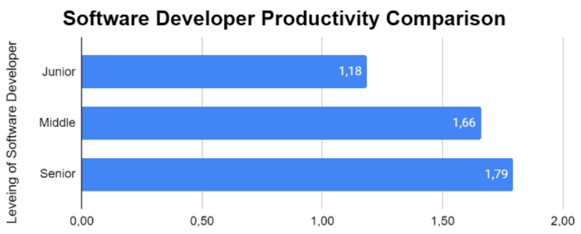

Fig. 3. Software Developer Productivity Comparison

Figure 3 compares software developer productivity at each level in developing software—the developers' ability at every level goes with their experience in software development. The productivity of software developers depends on the software's complexity and the software developer's programming skills. So Figure 3 relates to Table 14 regarding the specification of the software developer's programming abilities for four abilities: algorithms, coding, database queries, technology exploration, and software testing. Junior developers have fewer abilities compared to the two levels of developers above. The middle developer status is between junior and senior, while the senior developer has the highest ability.

The higher the understanding of programming skills when implementing software, the higher the level of productivity. For example, in Table 16 or Figure 3, senior software developers have higher productivity scores and programming skills as problem solvers. Thus, calculating software developer productivity in making program code on SDP using equation 4 can describe productivity quantitatively.

The productivity value at each level of a software developer can be used as an alternative solution to calculate the estimated time, cost, and availability of human resources in planning software development for similar applications that already exist. The estimation is calculated based on the software size, the required software developer specifications, and the productivity value at each developer level who can implement several FPs in one day in the form of program code. Each software developer has a different productivity value based on the level of programming ability and software complexity.

This productivity value is very likely to be used for all software development projects, mainly if the project has limited resources (time, cost, and human resources). For example, suppose a software development project has a time limit that must be completed immediately. In that case, the selection of a software developer is based on the least effort by prioritizing the productivity of highly qualified human resources. Meanwhile, suppose a software development project has limited human resources with senior software developer conditions already mapped out on other software development projects. In that case, the alternative is to choose a software developer with slightly lower productivity—for example, the selection of HR with medium or low qualifications.

However, the estimated cost of software development calculation needs to be studied deeper, to know whether the increase in productivity is inversely proportional to the cost. For example, the higher the value of HR productivity, the lower the development costs or vice versa. On the other hand, increased productivity may be directly proportional to development costs; the higher the value of HR productivity, the more expensive development costs. Development costs are increasing because of the need for highly qualified human resources with higher salaries.

IV. CONCLUSIONS

Information on software developer productivity at every level of programming ability is the primary key to making a more realistic SDP plan, namely determining the duration of software development based on the number of FP/day each software developer can develop. For example, senior software developers have less development time than junior and middle developers. This statement is evidenced by the productivity value of senior developers being higher than that of middle and junior developers, namely 9.87 for junior developers, 13.27 for middle developers, and 14.34 for senior developers.

The estimated productivity value is calculated based on the level of programming ability at the software implementation stage to the complexity of the software functionality. The use of function points in calculating the productivity of specific projects allows it to be used as a comparison with other projects in similar problem domains. The number of FP/day implemented by software developers utilizes previous productivity data so that the productivity measure of software developers can be used as an estimated parameter for software development planning and a better estimate of the budget for new projects. This is because the calculation of development effort in planning, which is initially based on the subjective assessment of the SDP planner, can be replaced with an objective assessment by utilizing the productivity value of each software developer's programming ability calculated quantitatively.

Suggestions for further research include adding parameters to calculate software developer effort. For example, the specification of programming skills and work attitude skills, so it is necessary to design a case study on a software development project. In addition, the scope of productivity calculations is not only at the software implementation stage. However, it can involve other development stages, such as functionality specification analysis, design, testing, or software maintenance so that function point calculations can use the DPFPC and EPFC software types.

ACKNOWLEDGMENT

Researchers would like to thank the Center for Research and Community Service (PPPM) Bandung State Polytechnic (POLBAN), which has supported the implementation of this research, for the financial assistance that has been provided.

REFERENCES

[1] N. Rachmat and Saparudin, “Estimasi Ukuran Perangkat Lunak Menggunakan Function Point Analysis-Studi Kasus Aplikasi Pengujian dan Pembelajaran Berbasis Web,” in *Prosiding Annual Research Seminar*, 2017, pp. 3–5.

[2] H. Rohayani, F. L. Gaol, B. Soewito, and H. L. Hendrie, “Estimated Measurement Quality Software On Structural Model Academic System With Function Point Analysis,” in *International Conference on Applied Computer and Communication Technologies (ComCom)*, May 2017.

[3] A. Y. P. Putri and A. P. Subriadi, “Software Cost Estimation Using Function Point Analysis,” in *The 4th International Seminar on Science and Technology*, Aug. 2018, vol. 79, pp. 79–83.

[4] M. Usman, E. Mendes, and J. Börstler, “Effort estimation in Agile software development: A survey on the state of the practice,” in *ACM International Conference Proceeding Series*, Apr. 2015, vol. 27-29-April-2015. doi: 10.1145/2745802.2745813.

[5] M. F. Hillman and A. P. Subriadi, “40 Years Journey of Function Point Analysis Against Real-time and Multimedia Applications,” in *The fifth Information System International Conference* , 2019, pp. 266–274.

[6] B. Prakash and V. Viswanathan, “A survey on software estimation techniques in traditional and agile development models,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 7, no. 3, pp. 867–876, Sep. 2017, doi: 10.11591/ijeecs.v7.i3.pp867-876.

[7] S. Wagner, “Defining Productivity in Software Engineering,” in *Rethinking Productivity in Software Engineering*, First Edition., C. Sadowski and T. Zimmermann, Eds. Apress Oppen, 2019. doi: 10.1007/978-1-4842-4221-6.

[8] A. H. López, R. C. Palacios, P. S. Acosta, and C. C. Lumberas, “Productivity measurement in software engineering: A study of the inputs and the outputs,” *International Journal of Information Technologies and Systems Approach*, vol. 8, no. 1, pp. 45–67, Jan. 2015, doi: 10.4018/IJITSA.2015010103.

[9] E. A. de Oliveira and R. C. Noya, “Using Productivity Measure and Function Points to Improve the Software Development Process,” *Computer Science*, 2013.

[10] E. Oliveira, D. Viana, M. Cristo, and T. Conte, “How have software engineering researchers been measuring software productivity?: A systematic mapping study,” in *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems*, 2017, vol. 2, pp. 76–87. doi: 10.5220/0006314400760087.

[11] A. H. López, R. C. Palacios, Á. G. Crespo, and F. C. Isla, “Software Engineering Productivity: Concepts, Issues and Challenges,” *International Journal Information Technology Project Managemet*, vol. 2, pp. 37–41, 2011.

[12] S. I. Mohamed, “Software development productivity impact from an industrial perspective,” *Int J Sci Eng Res*, vol. 6, no. 2, pp. 1333–1342, Feb. 2015, [Online]. Available: http://www.ijser.org.

[13] J. Rashid, T. Mahmood, and W. M. Nisar, “A Study on Software Metrics and its Impact on Software Quality,”

Technical Journal, University of Engineering and Technology (UET), vol. 24, no. 1, pp. 1–14, 2019.

[14] N. Choursiya and R. Yadav, "An Enhanced Function Point Analysis (FPA) Method for Software Size Estimation," *International Journal of Computer Science and Information Technologies IJCSIT*, vol. 6, no. 3, pp. 2797–2799, 2015.

[15] S. M. R. Chirra and H. Reza, "A Survey on Software Cost Estimation Techniques," *Journal of Software Engineering and Applications*, vol. 12, no. 06, pp. 226–248, 2019, doi: 10.4236/jsea.2019.126014.

[16] P. Vickers, "An Introduction to Function Point Analysis," 2003. [Online]. Available: www.paulvickers.com/northumbria.

[17] D. Garmus and D. Herron, *Function Point Analysis: Measurement Practices for Successful Software Project*. Addison Willey Professional, 2000.

[18] J. Shah, N. Kama, and S. A. Ismail, "An empirical study with function point analysis for software development phase method," in *ACM International Conference Proceeding Series*, May 2018, pp. 7–11. doi: 10.1145/3220267.3220268.

[19] A. Alexander, "How to Determine Your Application Size Using Function Points," 2004. [Online]. Available: http://ifpug.org.

[20] IFPUG, "Function Point Counting Practices Manual," 2010.

[21] AltexSoft Team, "Software Engineer Qualification Levels: Junior, Middle, and Senior," *Altexsoft*, Sep. 23, 2018. https://www.altexsoft.com/blog/business/software-engineer-qualification-levels-junior-middle-and-senior/ (accessed Sep. 27, 2022).

[22] K. Anderson, "Junior vs. Mid vs. Senior software engineers – experience, skills, & expectations," *DEPT*. https://www.deptagency.com/en-us/insight/junior-vs-mid-vs-senior-software-engineers-experience-skills-expectations/ (accessed Sep. 27, 2022).