International Journal on Advanced Science Engineering Information Technology

Vol.10 (2020) No. 3
ISSN: 2088-5334

Functional Size Measurement Tool-based Approach for Mobile Game

Nur Ida Aniza Rusli[a,1], Nur Atiqah Sia Abdullah[a,2]

[a] *Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, 40450 Shah Alam, Selangor, Malaysia*
E-mail: [1]idaaniza@gmail.com; [2]atiqah@tmsk.uitm.edu.my

***Abstract*— Nowadays, software effort estimation plays an important role in software project management due to its extensive use in industry to monitor progress, and performance, determine overall productivity and assist in project planning. After the success of methods such as IFPUG Function Point Analysis, MarkII Function Point Analysis, and COSMIC Full Function Points, several other extension methods have been introduced to be adopted in software projects. Despite the efficiency in measuring the software cost, software effort estimation, unfortunately, is facing several issues; it requires some knowledge, effort, and a significant amount of time to conduct the measurement, thus slightly ruining the advantages of this approach. This paper demonstrates a functional size measurement tool, named UML Point tool, that utilizes the concept of IFPUG Function Point Analysis directly to Unified Modeling Language (UML) model. The tool allows the UML eXchange Format (UXF) file to decode the UML model of mobile game requirement and extract the diagrams into component complexity, object interface complexity, and sequence diagram complexity, according to the defined measurement rules. UML Point tool then automatically compute the functional size, effort, time, human resources, and total development cost of mobile game. Besides, this paper also provides a simple case study to validate the tool. The initial results proved that the tool could be useful to improve estimation accuracy for mobile game application development and found to be reliable to be applied in the mobile game industry.**

***Keywords*— software effort estimation; functional size measurement; automation tool; UML model; mobile game.**

I. INTRODUCTION

Software effort estimation is crucial and remains one of the challenging tasks in software engineering since the 1940s. Rapid advances in software technology have shown the growing size and complexity, which can lead to difficulties for system analysts or project managers to predict the cost of software development projects. Accurate cost estimation is essential to both developers and customers. It can be practically applied in the software industry to support activities such as project planning, product monitoring and control, and assessment of team performance [1], [2].

Since the early 1950s, software development practitioners and researchers have been trying to develop methods to estimate software cost and schedule. Functional Size Measurement (FSM) is one of the methods in software cost estimation by measuring the amount of functionality to be delivered to obtain the project cost, project duration, and a number of resources needed to complete the projects.

FSM is one of the widely used methods for measuring the size of the estimated software system. Allan Albrecht introduced FSM in 1979, and now it is kept updated by International Function Point User Group Function Point Analysis (IFPUG FPA) [3]. Then, several FSM methods have emerged from IFPUG FPA to cope with recent developments in software engineering such as Mark II FPA [4], COSMIC FPA [5], NESMA FSM [6], and FiSMA FSM [7].

There have been numerous researches on adapting the FSM method to traditional software development such as 3D Points [8], Feature Point [9], and Early Function Point [10]. Although different software effort estimation has been proposed, the new software development demands a new context of estimation model as the requirements, new technology deployment, and methodology appeared to be different. It thus resulted in being less accurate since it fails to estimate some features in modern software systems [11], [12].

UML Point is an extended function point method for mobile game application technology by integrating the UML framework into IFPUG FPA base components. The utilization of both concept and method leads to the transformation of the new set of procedures and measurement rules to assist practitioners specifically in mobile game development effort estimation. As almost all measurement methods are conducted manually, it requires both effort and time [13]. Therefore, the objective of this paper is to show the implementation of the estimation tool for UML Point. Namely, as UML Point for Mobile Game, the tool will analyze the requirements from the UML model and estimates underlying software costing for mobile game applications.

A. Related Works

This section presents an overview of the existing automation tool in the field of FSM. In Azzouz and Abran [14], an estimation tool is proposed to generate functional size for systems that are modeled in the Rational Unified Process (RUP). The tool adopts the defined rules for mapping the UML model to the concept of COSMIC FFP. The detailed overview of the tool is presented, including the main results, measurement results by the functional process, and also measurement results by type of data movement. The tool was verified using the Rice Cooker case study. However, there are limitations mentioned in the paper, such as the measurement scope must be identified manually, and the tool has been not tested in the large scale of case studies.

µcROSE was introduced by Diab et al. [15], for predicting the functional size of systems from Rational Rose Real-Time (RRRT) models. The tool is principally incorporating the mapping concept of RRRT models and COSMIC FFP. The architecture of µcROSE generally consists of three panels (three views of objects panel, attributes list panel, and selected capsule panel) to extract the RRRT model in XML format into COSMIC FFP components. µcROSE has been verified in terms of correctness and completeness with a different set of test scenarios, but the results of the evaluation are not discussed in detail in the paper.

While in Abrahao and Insfran [16], the authors proposed an automation tool for counting the functional size from the UML model. The tool named REST fully complies the measurement concept in the previous work and measurement concept of IFPUG FPA. The tool has been validated using the Car Rental System case study, and the results obtained by the tool were compared with the manual counting performed by the experts.

Lind et al. [17] present the CompSize, a tool developed by the authors that automatically estimate software code size. The tool converts the information modeled using UML Profile into COSMIC FFP based on the mapping rules proposed in the previous work [18]. The tool has been applied in a case study. Results from the case study were compared with the measurement using the Saab method [19].

In De Souza et al. [20], the authors use the Experience Service tool to estimate mobile software development projects. The tool captures all information needed for functional size measurement according to the measurement concept of FiSMA. The system interface of the tool is presented; however, no case study is provided in the paper, and there is no further explanation regarding the validation of the tool.

Our tool is different from the studies that were mentioned here. We present an automation tool designed to be able to estimate the effort and cost of mobile game application development. As the input of the tool, we consider UML model to be adapted in IFPUG measurement concept with the help of mapping rules proposed in the previous work [21].

B. Overview Of IFPUG FPA

The function point method or "Function Point Analysis (FPA)" was developed by Allan J. Albrecht in the 1970s. It was an attempt to overcome the difficulty of "lines of code (LOC)" method and measures the software size from the end user's perspectives. The method was initially published in 1979 and later in 1983. Refinement has been made in 1984 and in 1986 when Albrecht introduced "International Function Point User Group (IFPUG)" as a standard in FPA. The concept of IFPUG is the measuring system from the amount of functionality required by the system. The measurement procedure consists of five components; Internal Logical File (ILF), External Interface File (EIF), External Input (EI), External Output (EO) and External Inquiry (EQ). ILF and EIF are identified as data function; meanwhile, EI, EO, and EQ are defined as transaction functions.

ILF is defined as a user identifiable group of logically related information maintained within the boundary of the application. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted. An EIF is similar to ILF, but it is maintained within the boundary of another application. Both data function contributes to a number of function points that depends on its complexity. The complexity of data function is based on Data Element Type (DET) and Record Element Type (RET) appeared in the data function. IFPUG define DET as a unique user recognizable, non- recursive field on the ILF or EIF; meanwhile, an RET is a user recognizable subgroup of data elements within an ILF or EIF. Table I is used to classify the complexity of ILF and EIF either it having "low", "average" or "high" complexity.

TABLE I
IFPUG FPA ILF/EIF COMPLEXITY

RET	**DET**		
	1-19	**20-50**	**51+**
1	Low	Low	Average
2-5	Low	Average	High
5+	Average	High	High

Transaction function represents the functionality provided to the user for the processing of data by an application. The definition of External Input (EI), External Output (EO) and External Inquiry (EQ) are described as follows:

- EI is defined as external input processes data or control information that comes from outside the application's boundary. The external input itself is an elementary process.
- EO is an elementary process that generates data or control information sent outside the application boundary.
- EQ is defined as an elementary process made up of input-output combinations that result in data retrieval. The output side contains no derived data. Here, derived data is the data requires processing other than direct retrieval and editing of information from ILF and/or EIF. No ILF is maintained during processing.

For transaction function, the complexity of each function is based on the number of DET and File Type Referenced

(FTR). An FTR can be an ILF or EIF; meanwhile, DETs of transaction function is considered data that cross the application boundary when the transaction is performed. The weighting complexity of transaction function of EI, EO, and EQ are shown in Table II and Table III respectively. Finally, the IFPUG function point is obtained by giving the value to five IFPUG components complexity as shown in Table IV.

TABLE II
IFPUG FPA EI COMPLEXITY

FTR	**DET**		
	1-4	**5-15**	**16+**
1	Low	Low	Average
2	Low	Average	High
3+	Average	High	High

TABLE III
IFPUG FPA EO/EQ COMPLEXITY

FTR	**DET**		
	1-5	**6-19**	**20+**
1	Low	Low	Average
2-3	Low	Average	High
4+	Average	High	High

TABLE IV
IFPUG FPA COMPLEXITY WEIGHT

IFPUG	**Low**	**Average**	**High**
ILF	7	10	15
EIF	5	7	10
EI	3	4	6
EO	4	5	7
EQ	3	4	6

II. MATERIALS AND METHOD

This section presents an overview of proposed UML Point method. The UML Point consists of two major steps; the first step describes the generic design of UML model that associate to mobile game architecture and the second step summarizes the measurement rules to assemble the concept of UML model into IFPUG FPA method.

A. UML Design

At the design level, four main models were used: use case diagram, component diagram, class diagram and sequence diagram. The center of UML Point method is the use case diagram. It describes the high-level activities in the mobile game. This method introduces specific stereotypes to indicate basic element in a mobile game that can be performed by use case actor. The stereotypes are described as follow:

- «start»: Activities for player enter a new game; all particular components are rendered in the application. Player may get a set of instructions or tutorial before start playing the game
- «inplay»: List of actions that can be taken during the gameplay such as pause, resume or change setting. Player may get rewards such as experience points or items during the game
- «end»: Activities to end the current level or terminate the application. All displayed characters are destroyed

Component diagram is often used to model a complex system. It manages the complex functionalities of a system by decomposing it into smaller parts. At this stage, all actors appeared in the use case diagram are visualized or transformed into the component and further categorized into proposed stereotypes. The stereotypes of the component diagram are purposely to categorize the main service of a component and it described as follow:

- «user»: Player of the game. It can be single player or multiplayer
- «UI»: A medium to display the game objects, game scene and control the interaction process
- «library»: Mobile game assets such as animations, physics or network
- «executionObject»: Describes the static and dynamic game objects. It describes set of characteristics, behaviours or actions to be performed
- «storage»: Handling the data of the game such as properties, game items or score points. It can be local database or external storage
- «executionEnvironment»: Targeted platform or devices

Component interface is a generic way to link the interactions among the component diagram. At this step, the component interface or introduced it as object interface is implemented in a class diagram way in order to describe functionality that should be provided. Stereotyped as «interface», the implementation of object interface, however, is different from class diagram as it only includes operations that can be used by other components.

The concept of class diagram is used to represent the core concept or internal structure of a component. It is useful in presenting the detail requirement of a component such as game characters, input control or game mechanics. Class diagram allows the developer to enhance the idea during the design process and capture potential features as much as possible to support the core gameplay. The characteristic of the class diagram may result to share similar elements of functionalities. Therefore, class diagram relationships (generalization/inheritance, association, aggregation, and composition) are included as part of the process to form the application structure.

Sequence diagram is designed purposely to summarize the internal interaction between objects. Several sequence diagrams should be modeled to depict all possible behavior of a system. To visualize the complete interactions, sequence diagrams usually developed from the context of use case scenarios. Therefore, from the mentioned use cases, at least the mobile game consists of three sequence diagrams; sequence diagram for «start» activity, a sequence diagram for «inplay» activity and sequence diagram for «end» activity.

B. Measurement Rules

To implement the UML design, we proposed three-step measurement procedure, which can be mapped in IFPUG FPA complexity. The measurement steps are described as follow:

- Count data function for Component Diagram
- Count data function for Object Interface
- Count transaction function for Sequence Diagram

In the IFPUG FPA method, the data function is divided into internal logical file (ILF) and external interface file (EIF). ILF file is defined by logical file that are maintained by the application; meanwhile, EIF file are those referenced by the application but maintained by other applications. In our method, the proposed stereotype in component diagram is used to differentiate the data function. A set of rules to categorized ILF and EIF is describes as follow:

- Rule 1: Every component diagram become a candidate of data function
- Rule 2: Accept each of «user» data function as EIF
- Rule 3: Accept each of «UI» data function as ILF
- Rule 4: Accept each of «library» data function as EIF
- Rule 5: Accept each of «executionObject» data function as ILF
- Rule 6: Accept each of «storage» data function as ILF or EIF. Internal storage is accepted as ILF and external storage is accepted as EIF
- Rule 7: Accept each of «executionEnvironment» data function as EIF

The complexity of ILF and EIF are evaluated by counting the number of DET and RET appeared in both data functions. As an internal part of the component, a structured class diagram is used to identify the complexity for both ILF and EIF data functions. Therefore, the number of DET and RET can be classified by using the following rules:

- Rule 8: Count RET and DET as one for component that does not contain any class(es)
- Rule 9: If component contains class(es) and there are no relations (generalization, association, aggregation or composition) between classes, the RET is counted as 1 to each class(es)
- Rule 10: If generalization relation connects two classes, the RET is counted as subclass only
- Rule 11: If association relation connects two classes, the RET is counted as both superclass and subclass
- Rule 12: If aggregation relation connects two classes, the RET is counted as both superclass and subclass
- Rule 13: If composition relation connects two classes, the RET is counted as superclass only
- Rule 14: Count DET to each non-repeated attribute in class diagram

To avoid tight coupling counting between component diagram and object interface, object interface has also become a candidate to the data functions. The ILF and EIF rules of object interface are immediate:

- Rule 15: Every object interface is mapped each of object interface into logical file
- Rule 16: Accept each of object interface(s) belongs to «user» and counted as EIF
- Rule 17: Accept each of object interface(s) belongs to «UI» and counted as ILF
- Rule 18: Accept each of object interface(s) belongs to «library» and counted as EIF
- Rule 19: Accept each of object interface(s) belongs to «executionObject» and counted as ILF
- Rule 20: Accept each of object interface(s) belongs to «storage» and counted as ILF or EIF
- Rule 21: Accept each of object interface(s) belongs to «executionEnvironment» and counted as EIF

Both of ILF and EIF object interface then need to be rated as low, average or high complexity by identifying the number of RET and DET that captured in object interface. The RET and DET of object interface are based on the following rules:

- Rule 22: Count RET as 1 to each object interface
- Rule 23: Count DET to each non-repeated attributes in object interface

In counting the transaction functions, the measurement process is captured from the sequence diagram; considering the proposed UML use case diagram does not provide sufficient information to complete the sizing process using IFPUG FPA base components. Each of «start», «inplay» and «end» use cases are transformed into sequence diagram and become a candidate for transaction function.

To count the transaction function, we provide mapping rules for UML sequence diagram to be applied in the IFPUG FPA transaction functions components. The rules are described as follow:

- Rule 24: Every «start», «inplay» and «end» use cases become a candidate of transaction function
- Rule 25: Accept each of «start» transaction as EI
- Rule 26: Accept each of «inplay» transaction as EI, EO or EQ
- Rule 27: Accept each of «end» transaction as EO

Counting the EI, EO, and EQ complexity is very simple; the adjustment is based on the number of file type referenced (FTR) and DET that appear in the sequence diagram. The proposed rules of FTR and DET are described as follow:

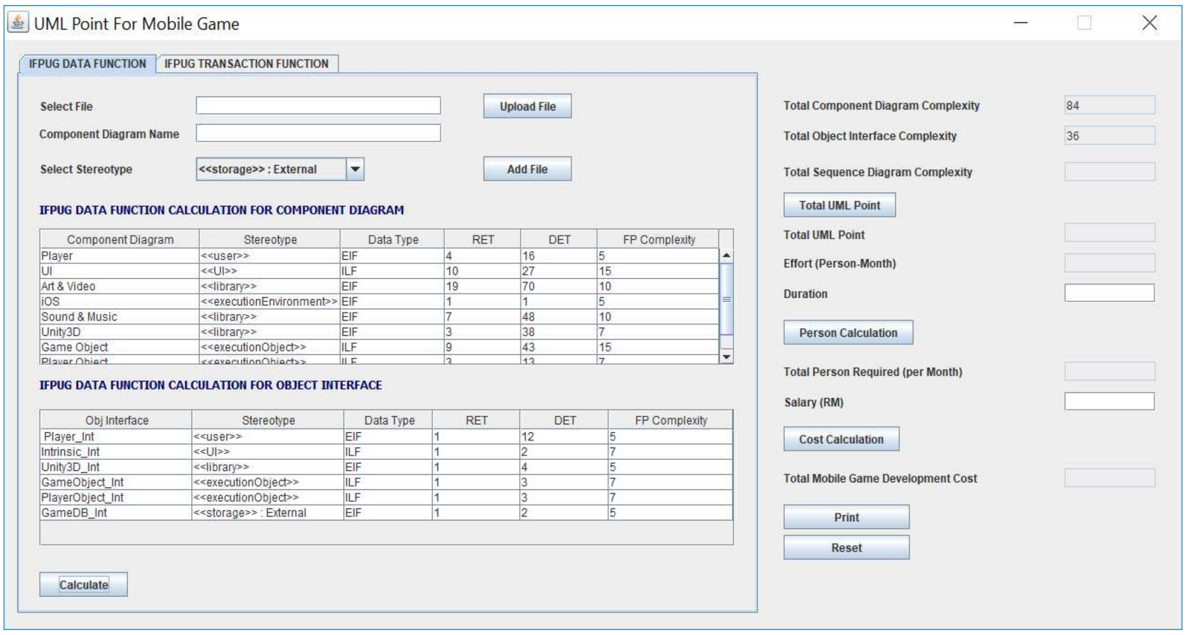
- Rule 28: FTR is counted from ILF and EIF of object interface that appears in the sequence
- Rule 29: Count DET to each message between FTR

III. RESULT SAND DISCUSSION

With the respect of using UML framework, the proposed tool requires the user to sketch the requirements in UML tool. UMLet is an open-source tool, contains several characteristics and a significant portion that are useful for object orientation. UMLet saves the project file in UXF format, which is flexible to encode the design and decode it in any medium conversion including UML Point for Mobile Game tool.

This section provides a case study to demonstrate the measurement tool. The Intrinsic Game is a 2D battle game style which was designed to be played on iPad. The requirement of the game is modeled in UML as mentioned in the previous guidelines using UMLet tool. The UXF files of Intrinsic Game are then imported into the software tool.

The main interface of the tool consists of IFPUG Data Function and IFPUG Transaction Function tab window. Fig. 1 shows the first tab of the tool. The measurement process starts with user upload of the UXF file of UML component diagram into the application. The user needs to choose appropriate stereotypes associated with the uploaded file in order to make the tool accurately differentiate the file into ILF or EIF.



Select File

Component Diagram Name

Select Stereotype <<storage>> : External

IFPUG DATA FUNCTION CALCULATION FOR COMPONENT DIAGRAM

Component Diagram	Stereotype	Data Type	RET	DET	FP Complexity
Player	<<user>>	EIF	4	16	5
UI	<<UI>>	ILF	10	27	15
Art & Video	<<library>>	EIF	19	70	10
iOS	<<executionEnvironment>>	EIF	1	1	5
Sound & Music	<<library>>	EIF	7	48	10
Unity3D	<<library>>	EIF	3	38	7
Game Object	<<executionObject>>	ILF	9	43	15
Player Object	<<executionObject>>	ILF	3	13	7

IFPUG DATA FUNCTION CALCULATION FOR OBJECT INTERFACE

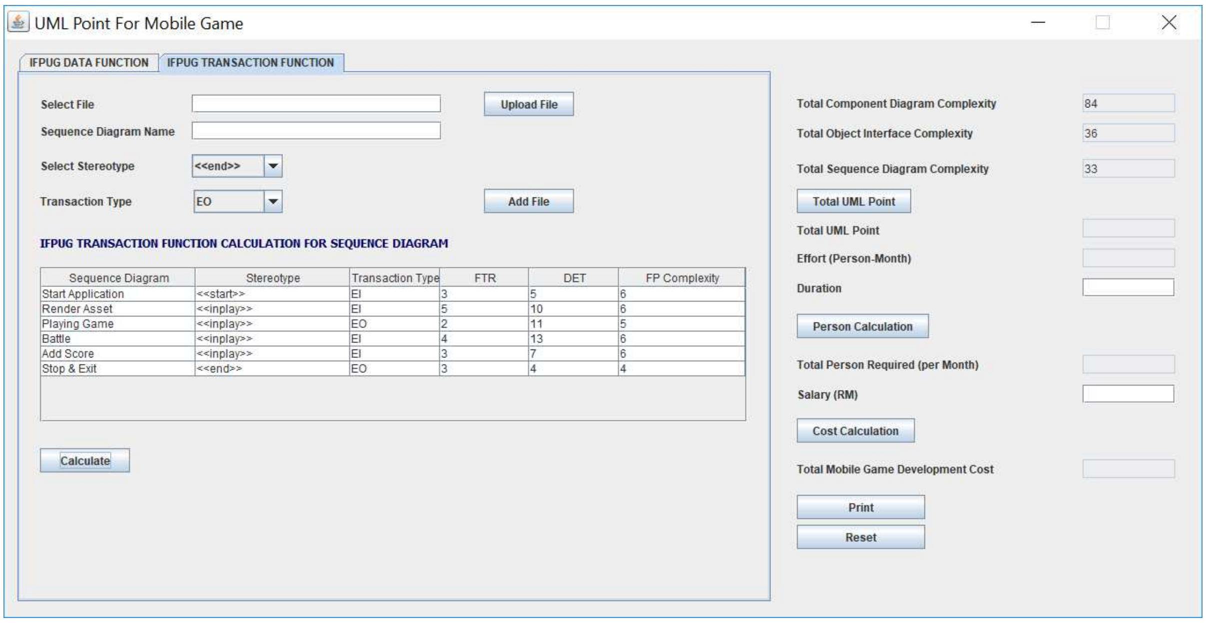
Obj Interface	Stereotype	Data Type	RET	DET	FP Complexity
Player_Int	<<user>>	EIF	1	12	5
Intrinsic_Int	<<UI>>	ILF	1	2	7
Unity3D_Int	<<library>>	EIF	1	4	5
GameObject_Int	<<executionObject>>	ILF	1	3	7
PlayerObject_Int	<<executionObject>>	ILF	1	3	7
GameDB_Int	<<storage>> : External	EIF	1	2	5

Total Component Diagram Complexity	84
Total Object Interface Complexity	36
Total Sequence Diagram Complexity	
Total UML Point	
Effort (Person-Month)	
Duration	
Total Person Required (per Month)	
Salary (RM)	
Total Mobile Game Development Cost	

Fig. 1 IFPUG Data Function Counting Process

The tool then extracts the file between component diagrams and object interface and automatically calculates the file in both data function (data function for the component diagram and data function for object interface). The user can view the number of DET, RET and function point from the respective file after clicking the button the Add File as seen in Fig. 1.

Fig. 1 also shows a list of data function in component diagram and data function in object interface for The Intrinsic Game; results show the game have 84fp for total component diagram complexity and 36fp for total object interface complexity.



Select File

Sequence Diagram Name

Select Stereotype <<end>>

Transaction Type EO

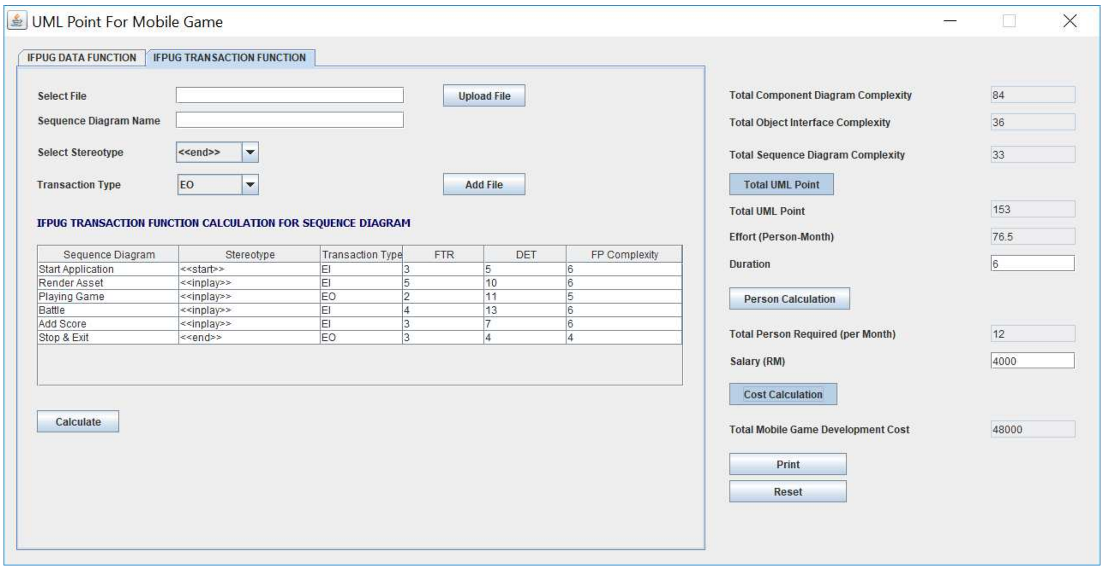
IFPUG TRANSACTION FUNCTION CALCULATION FOR SEQUENCE DIAGRAM

Sequence Diagram	Stereotype	Transaction Type	FTR	DET	FP Complexity
Start Application	<<start>>	EI	3	5	6
Render Asset	<<inplay>>	EI	5	10	6
Playing Game	<<inplay>>	EO	2	11	5
Battle	<<inplay>>	EI	4	13	6
Add Score	<<inplay>>	EI	3	7	6
Stop & Exit	<<end>>	EO	3	4	4

Total Component Diagram Complexity	84
Total Object Interface Complexity	36
Total Sequence Diagram Complexity	33
Total UML Point	
Effort (Person-Month)	
Duration	
Total Person Required (per Month)	
Salary (RM)	
Total Mobile Game Development Cost	

Fig. 2 IFPUG Transaction Function Counting Process

Fig. 2 shows a screenshot of second window tab; IFPUG Transaction Function window tab provides a similar process in the first tab, it requires the user to upload the sequence diagram file in UXF format. The user needs to select enter the name of the file and select the stereotype and transaction type associated with the uploaded file in order to obtain accurate function point. The tool then reports the number of FTR, DET and function point for each sequence diagram file. The Intrinsic Game is estimated to have 33fp for total sequence diagram complexity as shown in Fig. 2. Fig. 3 shows the summary result for The Intrinsic Game. The tool calculates the total of three complexities (total complexity for the component diagram, object interface, and sequence diagram) and The Intrinsic Game is estimated to have 153fp, which match the manual counting as mentioned in previous work [21].



Select File:
Sequence Diagram Name:
Select Stereotype: <<end>>
Transaction Type: EO

IFPUG TRANSACTION FUNCTION CALCULATION FOR SEQUENCE DIAGRAM

Sequence Diagram	Stereotype	Transaction Type	FTR	DET	FP Complexity
Start Application	<<start>>	EI	3	5	6
Render Asset	<<inplay>>	EI	5	10	6
Playing Game	<<inplay>>	EO	2	11	5
Battle	<<inplay>>	EI	4	13	6
Add Score	<<inplay>>	EI	3	7	6
Stop & Exit	<<end>>	EO	3	4	4

Total Component Diagram Complexity	84
Total Object Interface Complexity	36
Total Sequence Diagram Complexity	33
Total UML Point	153
Effort (Person-Month)	76.5
Duration	6
Total Person Required (per Month)	12
Salary (RM)	4000
Total Mobile Game Development Cost	48000

Fig. 3 Function Point Count and Total Software Cost

The total software cost is obtained after the user enters the duration. This is subject to complete project and salary rate for one programmer. As shown in Fig. 3, The Intrinsic Game is estimated to cost around RM48,000 for six months of development time.

IV. Conclusion

This paper aims to present UML Point for Mobile Game, an estimation tool that supports function point counting from the UML design. The tool helps in calculating the functional size of the mobile game from the user requirement, which can further be used to estimate the effort and cost of mobile game application development. The tool reflects the procedure of UML Point counting process by mapping the IFPUG FPA concept to UML model and extracts the counting process in three factors; UXF. Results from the case study have shown that the proposed tool can be applied in the mobile game industry. However, various case studies should be conducted to validate the proposed tool.

Acknowledgment

The authors express appreciation to Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA and MOHE Grant (FRGS/1/2018/ICT04/UITM/02/9) for sponsoring this paper.

References

[1] O. J. Klakegg and S. Lichtenberg, "Successive cost estimation-successful budgeting of major projects," in *Procedia-Social and Behavioral Sciences*, 2016, 226, pp. 176-183.

[2] A. Ismail and V. Cardellini, "Towards self-adaptation planning for complex service-based systems," in *International Conference on Service-Oriented Computing*, Springer, 2013, pp. 432-444.

[3] ISO/IEC 20926, "Software Engineering – IFPUG 4.1 Unadjusted Functional Size Measurement Method – Counting Practices Manual," in *International Organization for Standardization*, Geneva, 2003.

[4] ISO/IEC 20968, "Software Engineering – MkII Function Point Analysis – Counting Practices Manual," in *International Organization for Standardization*, 2002.

[5] ISO/IEC 19761, "Software Engineering – COSMIC Full Function Point Measurement Manual," v.2.2, 2003.

[6] ISO/IEC 24570, "Software Engineering – NESMA Functional Size Measurement Method, v.2.1 – Definitions and Counting Guidelines for the Application of Function Point Analysis," in *International Organization for Standardization – ISO*, Geneva, 2005.

[7] P. Forselius, "Finnish Software Measurement Association Functional Size," in *Finnish Software Metrics Association*, Finland, 2004.

[8] S.A. Whitmire, "3D Function Points: Scientific and Real-time Extensions to Function Points," in *Proceedings of the Pacific Northwest Software Quality Conference*, 1992.

[9] T.C. Jones, "A Short History of Function Points and Feature Points," in *Software Productivity Research Inc.*, USA, 1987.

[10] R. Meli, "Early and Extended Function Points: A New Method for Function Points Estimation," in *Proceedings of IFPUG-FALL Conference*, Scottsdale, Arizona, 1997.

[11] L.S. De Souza and G.S De Aquino Jr, "Estimating the Effort of Mobile Application Development," in *Proceedings of Second International Conference on Computational Science and Engineering*, 2014, pp. 45-63.

[12] M.M. Rosli, N.H.I Teoh, N.S.M. Yusop and N.S. Mohamad, "Fault prediction model for web application using genetic algorithm," in *International Conference on Computer and Software Modeling (IPCSIT)*, 14, 2011, pp. 71-77.

[13] M. Adnan and M. Afzal, "Ontology based multiagent effort estimation system for scrum agile method," in IEEE Access, 2017, vol. 5, pp. 25993-26005.

[14] S. Azzouz and A. Abran, "A proposed measurement role in the rational unified process and its implementation with ISO 19761: COSMIC-FFP," in *Software Measurement European Forum*, Rome, Italy, 2004.

[15] H. Diab, F. Koukane, M. Frappier and R. St-Denis, "µcROSE: Functional Size Measurement for Rational Rose RealTime," 2002.

[16] S. Abrahao and E. Insfran, "A metamodeling approach to estimate software size from requirements specifications," in *Software Engineering and Advanced Applications, 2008. SEAA'08. 34th Euromicro Conference*, IEEE, 2008, pp. 465-475.

[17] K. Lind, R. Heldal, T. Harutyunyan and T. Heimdahl, "CompSize: Automated size estimation of embedded software components," in *Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)*, IEEE, 2011, pp. 86-95.

[18] K. Lind and R. Heldal, "A model-based and automated approach to size estimation of embedded software components," in *International Conference on Model Driven Engineering Languages and Systems*, Springer, Berlin, Heidelberg, 2011, pp. 334-348.

[19] R. Baillargeon and R. Flores, "From Algorithms to Software – A Practical Approach to Model-Driven Design," SAE Technical Paper, 2007-01-1622, 2007.

[20] L.S. De Souza and G.S. De Aquino Jr, "Estimating the Effort of Mobile Application Development," in *Proceedings of Second International Conference on Computational Science and Engineering*, 2014, pp 45-63.

[21] N.I.A. Rusli and N.A.S. Abdullah, "UML Point for Mobile Game A Measurement Method for Sizing Mobile Game Design," in *Journal of Engineering and Applied Sciences*, 12(3), 2017, pp. 481-487.