Journal of Engineering and Applied Sciences 12 (3): 481-487, 2017
ISSN: 1816-949X

UML Point for Mobile Game A Measurement Method for Sizing Mobile Game Design

Nur Ida Aniza Rusli and Nur Atiqah Sia Abdullah
Centre of Computer Science Studies,
Faculty of Computer and Mathematical Sciences, Universiti Teknologi Mara (UiTM),
Shah Alam, Selangor, Malaysia

Abstract: This study proposes a method for estimating the size of mobile game application. UML Point presents rules to be mapped to UML model from the mobile game requirements. Methods/statistical analysis: This study provides specification of UML model to be adapted in the mobile game design. In addition to model the requirements, this study also introduces UML stereotypes for grouping certain elements in mobile game. We present mapping rules for measure the functional size of mobile game using IFPUG Function point analysis and provide a case study to show the possibility the proposed method to be applied in mobile games. From the case study, the use of UML model and the proposed stereotype were found useful in presenting the requirements. The results from the case study also support the claim that the proposed rules can be use in early estimation of mobile game application. Application/improvements: the improvements in this research involve the development of automated tool by directly exporting the UML game model to the tool for estimation process and conduct experiment for validation.

Key words: Software setrics, application effort estimation, functional size measurement, UML, mobile game

INTRODUCTION

These days, the mobile game is tremendously well known in the gaming industry. Evolution from gaming machine, PC and console games to portable application, mobile game comprised of entertaining contents which are composed of a large number of functionalities can be fulfilling to gamers. Gaming industry therefore, facing inevitable innovations that resulting from the wide range of operating systems such as apple ios, google android, and windows phone os. Fast growth, strong competition, and upgraded technology compel, developers offer interactive 2 or 3D games and social interactions among players to generate more profit for the company.

The mentioned characteristics along with limited duration of development life cycle which mostly take from 2-6 months become a challenge for developer to specify properly the requirements as it will impact the project schedule, cost and the whole life project value (Alves *et al*., 2007). Thus, the development costing of a mobile game is difficult to be estimated. Furthermore, there is no standardized method in estimating the mobile game application and the costing factor may be varied or changed by different person and perspective. Therefore, mobile game companies tend to approximate the application based on previous similar projects (Kim, 2012).

In larger context of software (including mobile games application), accurate estimation is important as the report indicates that 45% of IT projects were over budgets, 7% were late deliveries and 18% were discontinued altogether (Abdullah and Rusli, 2015; Wijayasiriwardhane and Lai, 2010).

Functional size measurement is one of the formal methods to estimate the cost of project by measuring the size of the software from the delivered functionality and called as Function Point (FP). This method was originally proposed by Alan Albrecht and maintained later by International Function Point User Group (IFPUG). Aside from IFPUG, there are several other methods in FSM, for example, MarkII, NESMA, FiSMA and COSMIC FSM; however, this paper only utilizes IFPUG base components to improve the mobile game measurement.

The extension of UML modelling in FSM is not a new concept in cost estimation. The UML model's flexibility, which is independent of any types of programming language and its ability to capture project's functionality at early development, motivates researchers to apply

Corresponding Author: Nur Ida Aniza Rusli, Centre of Computer Science Studies,
Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA (UiTM), Shah Alam, Selangor, Malaysia

UML model in function point measurement process (Saxena and Shrivastava, 2009). In addition, most of the current mobile games were developed through UML approach (Zhang *et al*., 2007). This has become motivation to use this object oriented technologies in sizing the functional size of mobile games.

MATERIALS AND METHODS

Uml modelling: This study describes the UML model to express the mobile game application design. The input to the UML modelling consists of use case diagram, component diagram, class diagram and sequence diagram together with stereotypes to be adapted in the mobile game design.

Stereotype is an extension concept in the UML model where it allow a software system to add new elements from the existing components in a particular domain or environment. Stereotypes also could be used as a simple indication of the elements, represents the nature behaviour of the object. This study stereotypes certain elements and made it accessible in accordance with mobile game requirements using guillemets («») symbol.

Use case diagram: Use case diagram is used to design the high-level requirement of a mobile game. Composed of actors and use cases, UML use case diagram describes the subjects of the application and general activities without illustrating the internal structure or detail elements in a system.

UML use cases conveyed a series of activities in the application. This phase defines a specific stereotype in the use cases by using «start», «inplay» and «end» notation. The implementation of the stereotypes indicates the basis activities in the mobile game that can be performed by the use case actors such as game player or server. The descriptions of UML use case diagram stereotypes are shown in (Table 1).

Component diagram and object interfaces: A component diagram is used to design the group of data or organize information such as software codes, scripting and command files. Component diagram is suitable for showing the structural architecture and managing the complex functionalities of mobile game. At this step, all actors that appear in the UML use case diagram are transformed into component diagram and further classified into the proposed stereotypes. In proposed UML component diagram are standardized in order to generate the primary elements that can be executed by the application. (Table 2) shows the descriptions of proposed stereotypes for component diagram.

Table 1: UML use case diagram stereotypes

Stereotypes	Definition
"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

Table 2: UML component diagram stereotypes

Stereotypes	Definition
"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
"Execution object"	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
"Execution environment"	Targeted platform or devices

When executing component, it is always necessary to implement interfaces to link the interaction process. Therefore, object interface is used for linking the components by implementing a class diagram 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.

Class diagram: The implemented components may consist of several classes to presents the detail requirements such as game characters, input control or game mechanics. The class diagram allows developer to enhance the idea during the design process and capture potential features as much as possible to support the core gameplay. Class diagram may share same elements or functionality. Therefore, the generalisation/inheritance, association, aggregation and composition relationships are included as part of the process to form the application structure.

Sequence diagram: The sequence diagram is designed purposely to summarize the internal interaction between objects. Several sequence diagrams should be modelled to depict all possible behaviour 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, sequence diagram for "inplay" activity and sequence diagram for "end" activity.

RESULTS AND DISCUSSION

Measurement rules: In IFPUG FPA, the main concept of measurement consists of two parts; data functions and transactional functions. Data functions consist of Internal Logical Files (ILF) and External Interface Files (EIF); meanwhile, transactional functions consist with External Inputs (EI), External Outputs (EO) and External Inquiries (EQ). These five components are needed to be assigned the weighting complexity in low, average or high complexity in order to obtain the Function Point (FP) (Lang *et al*., 2013). To apply the IFPUG FPA base components, this paper introduced three major steps for mapping the concept of IFPUG FPA to the proposed UML. Named as UML Point each of the steps will be composed of formal rules to improve the measurement process. 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

Data function for component diagram: The implementation of component diagram is suitable for determining the candidate of data functions. The "user-identifiable group" definition in IFPUG is rather vague; therefore to assist the mapping process, this study proposed a set of rules to differentiate each of component stereotypes into ILF and EIF:

- 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 "execution object" 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 "execution environment" data function as EIF

The relative complexity to each ILF and EIF are captured by counting the number of DET and RET for 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 contains 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 two classes are connected by generalization relation, the RET is counted as subclass only
- Rule 11: If two classes are connected by association relation, the RET is counted as both superclass and subclass
- Rule 12: If two classes are connected by aggregation relation, the RET is counted as both superclass and subclass
- Rule 13: If two classes are connected by composition relation, the RET is counted as superclass only
- Rule 14: Count DET to each non-repeated attribute in class diagram

Data function for object interface: As part of component diagram, object interface has also become a candidate to the data functions. The ILF and EIF rules of object interface are immediate:

- Rule 1: Every object interface is mapped each of object interface into logical file
- Rule 2: Accept each of object interface (s) belongs to "user" and counted as EIF
- Rule 3: Accept each of object interface (s) belongs to "UI" and counted as ILF
- Rule 4: Accept each of object interface (s) belongs to "library" and counted as EIF
- Rule 5: Accept each of object interface (s) belongs to "execution object" and counted as ILF
- Rule 6: Accept each of object interface (s) belongs to "storage" and counted as ILF or EIF
- Rule 7: 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 8: Count RET as 1 to each object interface
- Rule 9: Count DET to each non-repeated attributes in object interface

Transaction function for sequence diagram: In counting the transaction functions, the measurement process is captured from the sequence diagram; considering the proposed UML use case diagram does not provide

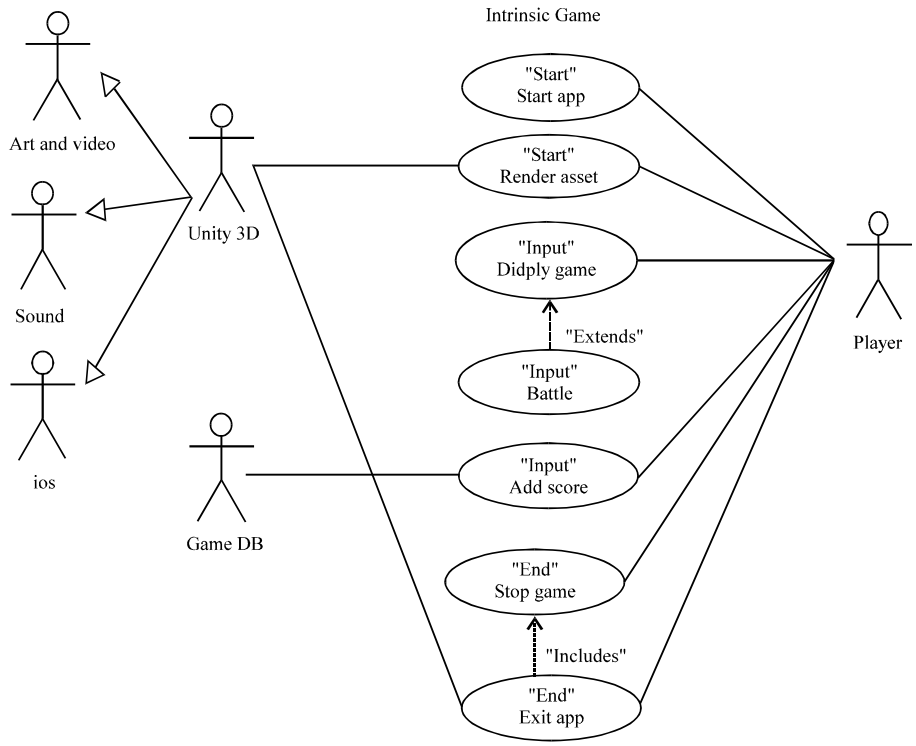

Fig. 1: Use case diagram for the intrinsic game

sufficient information to complete the sizing process using IFPUG 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, this study provides mapping rules for UML sequence diagram to be applied in the IFPUG transaction functions components. The rules are described as follow:

- Rule 1: Every "start", "inplay" and "end" use cases become a candidate of transaction function
- Rule 2: Accept each of "start" transaction as EI
- Rule 3: Accept each of "inplay" transaction as EI, EO or EQ
- Rule 4: 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 5: FTR is counted from ILF and EIF of object interface that appear in the sequence
- Rule 6: Count DET to each message between FTR

From the proposed rules the UML Point can be summarized based on following formula:

$$\text{UML Point} = FP_{Component} + FP_{Interface} + FP_{Sequence} \quad (1)$$

$$FP_{Component} = \sum Component_{ILF(RET,DET)} + \sum Component_{EIF(RET,DET)} \quad (2)$$

$$FP_{Interface} = \sum Interface_{ILF(RET,DET)} + \sum Interface_{EIF(RET,DET)} \quad (3)$$

$$FP_{Sequence} = \sum Sequence_{EI(FTR,DET)} + \sum Sequence_{EO(FTR,DET)} + \sum Sequence_{EQ(FTR,DET)} \quad (4)$$

Case study: A case study is conducted in order to evaluate the proposed UML Point. The requirement documentation of the intrinsic game is used to support the measurement process. The intrinsic game is a 2D battle game style which was designed to be played on iPad. This study provides an overview of UML model of the game and discusses how the measurements are conducted. However, not all parts of UML model are presented here due to the limitation of space.

Figure 1 shows the UML use case diagram for the intrinsic game. The Intrinsic Game is consists of 6 actors;

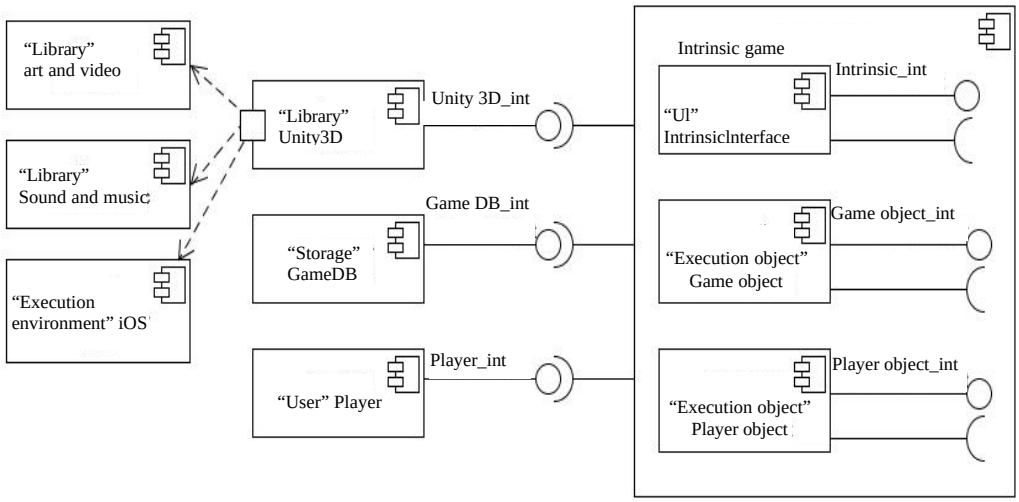

Fig. 2: Component diagram for the intrinsic game

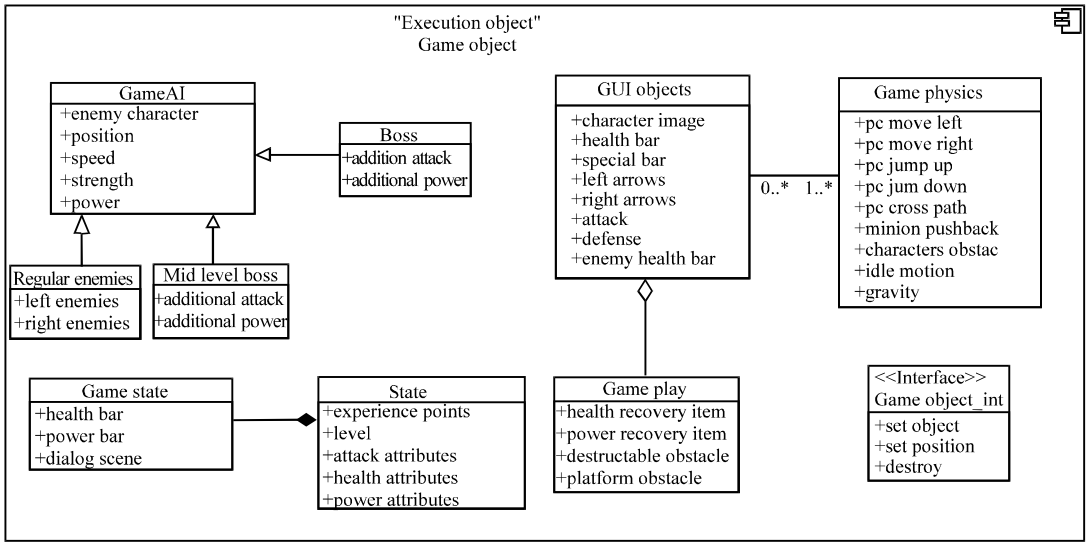

Fig. 3: Detail structure of component gameobject

Player, Unity 3D, art and video, sound, iOS and game DB. The mentioned actors interact with seven use cases, including start application in-battle game and exit application activity.

Figure 2 shows the component diagram of the intrinsic game. Each of actors defined in UML use case diagram are converted into a unit of component and categorized the components into the proposed stereotypes. In realizing the requirements, it is possible to add subcomponents to the main model. The component diagram may deliver "provide" or "request" object interfaces, depending upon the behaviour that the component should perform. Figure 3 shows the detailed structure of component game object, in which the generalization and composition relationships are presented; class regular enemies, mid level boss and boss are generalized into superclass gameAI and class game state and class state interact through composition relationships. In this Fig. 3, set object, set Position and destroy are defined in the game object-Int for interaction process.

The specified use cases in Fig. 1 are then visualized in the set of sequence diagrams. Fig. 4 show the sequence diagram for battle activity. Player sets the level through user interface and all particular objects are positioned according to the requested level. Then the battle activities

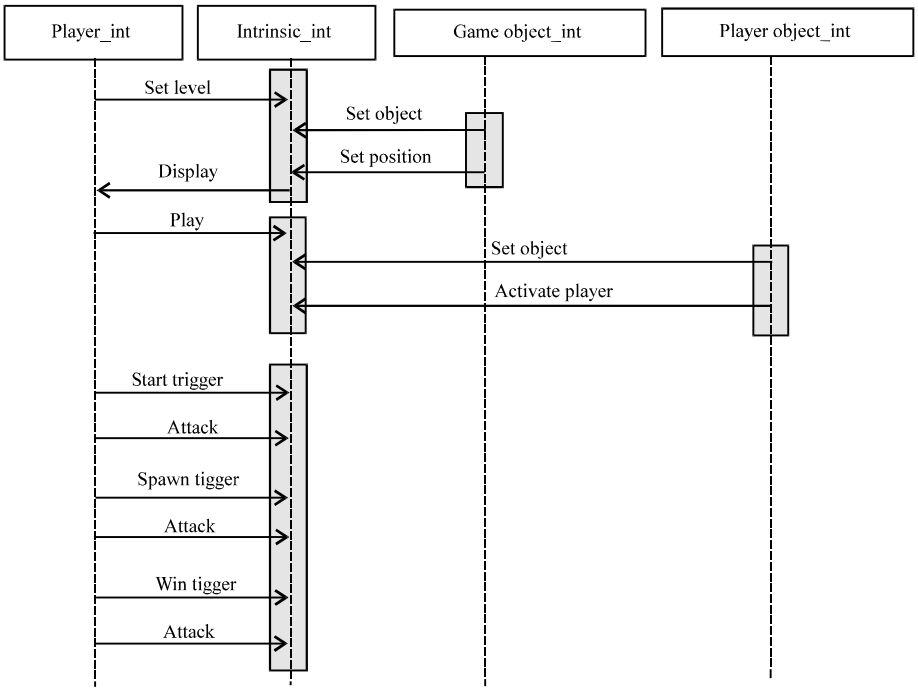

Fig. 4: Sequence diagram for battle

are realized. Player-int. Start trigger, player-int. spawn trigger and player-in win trigger functions are called in the sequence as part of attack process.

The process of counting the UML model in Fig. 2-4 are based on the proposed rules in the previous section. In summary, component game object consists of 7RET, 43DET; object interface game object-Int consists of 1RET, 3DET and sequence diagram for battle consists of 4FTR, 13DET. After all the IFPUG elements (DETs, RETs and FTRs) are identified; the IFPUG weighting complexity (not reported here) are referred to classify each data function and transaction function is having low, average or high complexity. The low, average and high complexity for both data function and transaction function are then weighted to obtain the final functional size of the software application. The case study of the intrinsic game is estimated to have 153 FP (84 FP for component diagram, 36 FP for object interface and 33 FP for sequence diagram). The estimated function points are then can be used as parameters in predicting the effort and cost estimation of mobile games for The Intrinsic Game.

CONCLUSION

To make mobile games more applicable in the measurement process, this study extends the characteristics or requirements of mobile game by implementing it in the UML model. Use case diagram, component diagram, class diagram and sequence diagram are applied to the IFPUG FPA base components. The measurement rules aim to make the measurement process easier and to ensure all functionalities in the UML mobile game design are catered in the sizing process. This study has presented a case study in UML Point as a proof of the proposed measurement concept; however, a proper evaluation of method is needed to ensure the measurement can be applied in various mobile game requirements.

So, the next stage of research will involve the validation of the method and determine if the proposed measurement rules are applicable to the mobile games industry. The future work also involves the context of automation tool for UML Point. The tool will automatically count the functional size of the mobile game and estimates the effort and cost to develop the application based on the given input.

REFERENCES

Abdullah, N.A.S. and N.I.A. Rusli, 2015. Reviews on functional size measureapplication and UML model. Proceedings of the 5th International Conference on Computing and Informatics, August, 11-13, 2015, Universiti Utara Malaysia, Changlun, Malaysia, pp: 353-358.

Alves, C., G. Ramalho and A. Damasceno, 2007. Challenges in requirements engineering for mobile games development: The meantime case study. Proceedings of the 15th IEEE International Conference on Requirements Engineering, October 15-19, 2007, IEEE, Recife, Brazil, ISBN:0-7695-2935-6, pp: 275-280.

Kim, H., 2012. Frameworks for validation of mobile software project performance. Proceedings of the World Congress on Engineering and Computer Science, October 24-26, 2012, Catholic University of Deagu, Gyeongsan, South Korea, pp: 24-26.

Lang, M., K. Conboy and S. Keaveney, 2013. Cost Estimation in Agile Software Development Projects. In: Information Systems Development, Pooley, R., J. Coady, C. Schneider, H. Linger and C. Barry *et al.* (Eds.). Springer, New York, USA., pp: 689-706.

Saxena, V. and M. Shrivastava, 2009. Performance of function point analysis through UML modeling. ACM. Sigsoft Software Eng. Notes, 34: 1-4.

Wijayasiriwardhane, T. and R. Lai, 2010. Component Point: A system-level size measure for component-based software systems. J. Syst. Software, 83: 2456-2470.

Zhang, W., D. Han, T. Kunz and K.M. Hansen, 2007. Mobile game development: Object-orientation or not. Proceedings of the 31st Annual International Conference on Computer Software and Applications, July 24-27, 2007, IEEE, Shanghai, China, ISBN:0-7695-2870-8, pp: 601-608.