British Journal of Mathematics & Computer Science

British Journal of Mathematics & Computer Science
10(1): 1-9, 2015, Article no.BJMCS.17075
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

Abuse of Unified Modeling Language Diagrams in Software Development

M. Peter Ogedebe[1*] and Faki A. Silas[2]

[1]Faculty of Computing and Applied Science, Baze University, Abuja, Nigeria.
[2]Department of Computer Science, Faculty of Science and Technology, Bingham University, Karu, Nasarawa State, Nigeria.

Article Information

DOI: 10.9734/BJMCS/2015/17075
Editor(s):
(1) Kai-Long Hsiao, Taiwan Shoufu University, Taiwan.
(2) Tian-Xiao He, Department of Mathematics and Computer Science, Illinois Wesleyan University, USA.
Reviewers:
(1) A. Papazafiropoulou, Department of Internal Medicine and Diabetes Center, Tzaneio General Hospital of Piraeu, Greece.
(2) Anonymous, Universiti Sains, Malaysia.
(3) Shihan Yang, Guangxi University for Nationalities, China.
(4) Anonymous, Texas A&M University, USA.
(5) Anonymous, Mansoura University, Egypt.
Complete Peer review History: http://sciencedomain.org/review-history/9919

Short communication

Received: 25 February 2015
Accepted: 30 April 2015
Published: 27 June 2015

Abstract

The Unified Modeling Language (UML) is a general-purpose visual modeling language for specifying software-intensive systems. More precisely, it is a graphical language for visualizing, specifying, constructing and documenting the artifacts of software-intensive systems. UML is a key enabling technology for Software Developers and Software Engineers who seek to transition from traditional, human-intensive, code-centric software development processes to Model-Driven Development (MDD) processes that are requirements-driven and architecture-centric. However, Due to the lack of skills by developers and general purpose nature of UML diagrams, many developers abuse it by drawing diagrams that did not match particular activities or scenarios in the software project. This study makes a review of how UML is abuse and also makes a simple representation of UML diagrams.

Keywords: Universal modeling language; structure diagrams; behavior diagrams; use case; software developer.

1 Introduction

Modern software development extensively uses objects technology in its process from analysis to implementation. Objects technology is suited most to the current business environment, where changes are continuous and software support is required to handle it effectively. In view of the aforementioned, it

**Corresponding author: peter@binghamuni.edu.ng;*

becomes necessary for developers to understand Unified Model Diagrams (UML) and its uses in appropriate stages in software development process.

UML provides notations and diagrams to model a system in different views. The views and diagrams put together accurately represent the real-world scenario. UML supports both static and dynamic modeling. Static modeling represents a stand-alone system while its dynamic counterpart shows the changing behavior of the business system.

UML benefits software developers by improving communication among projects teams and visualization of the complex system [1]. This study state in some ways in which UML is abused and also make a review of the basic UML diagrams.

2 About Unified Modeling Language (UML)

Most people refer to the Unified Modeling Language as UML. The UML is an international industry standard graphical notation for describing software analysis and designs. When a standardized notation is used, there is little room for misinterpretation and ambiguity [2]. However, due to different versions and notations of UML diagrams, a variety of modeling tools in conjunction with the changing complex business world in software programming has made the idea of Standardization elusive. Though, standardization is striving to provide an efficient communication which proves the assertion that "a picture is worth a thousand words".

In UML 2, two basic categories of diagrams exist. There are, structure diagrams and behavior diagrams. Every UML diagram belongs to one these two diagram categories. The structure diagrams show the static structure of the system being modeled. They include the class, component, and object diagrams. Behavioral diagrams, on the other hand, show the dynamic behavior between the objects in the system, including their methods, collaborations, and activities [3].

It worth mentioning that UML is not designed for a specific and limited uses rather it is tool for sketching class diagrams in java and other objects oriented programs. Also believe to be the prototype language of next generation [4].

The applicability of UML to all phases of software development life cycle makes it a subject of abuse. This is due to the fact that most software developers lack methodologies, understanding, practical experience and modeling tools to communicate exactly user requirement to programmers.

3 Abuse of UML Diagrams

The ability of a developer to communicate his intention through UML diagrams to the user is one of the main focuses of software modeling. But the concept has been abuse in so many ways especially by voodoo developers who lack training in software development and perhaps UML diagrams. To them, every object is important and must be modeled even though; it plays no role in the system. This result in software artifact having too many UML diagrams that are not actually implementable during coding [5].

Following the claim for universality of UML makes it possible to be applicable to all stages of software development (analysis, design, and implementation). Most software developers abuse these diagrams by not clearly bringing out which of the diagram represent behavioral or static process. Also, some developers mix up interpretation of diagrams. This is due to the fact that assumptions base on manipulating a system are not imbedded in the UML drawings during t design.

Also, the ambiguity of developers drawing diagrams that can be interpreted in many different ways also gives room for abuse. These may be due to the fact that they don't understand the logic flow of the system being model or the system requirements.

Furthermore, due to the complex nature of some systems, it may not be possible to capture all the sub-system of a system even though there is room for extended diagrams. These makes developers make use of other diagrams that are not part of UML to further model their scenarios which fail to communicate the users required intentions. These could also be termed abuse of UML be structural decomposition.

Another abuse of UML diagrams is by abstraction. In as much as abstraction is good and encourage especially in object oriented programing, it leads to trouble when drawing UML diagram to represent abstracted scenarios. Most developers are unable to come up with UML diagram to depict exactly how abstraction is done. This is due to the fact that, abstraction goes beyond level of their comprehension. Because of the abuse of UML diagrams, some of the major ones are review below.

4 Use Case Diagram

A Use case model is a representation of sequence of transactions initiated by the user (actor) from outside the system [2]. In this process, the transaction uses internal objects to take the transaction to completion. Use case defines and describes what happens in the system in logical order. They represent the flows of events that the user triggers.

It is not necessary that the user be a human being. It could be an external hardware like barcode reader, a card-swiping machine or any other system with an interface. Users can be of the same kind, type, role and responsibility performing the same use case.

By standard, a use case is modeled by:

- Boundary or frame
- Line of communication of participatory association between actor(s) and the use case
- Transaction steps
- Generalization among use case.

A use case may begin with a precondition or no precondition. It concludes with the achievement of a specific goal.

4.1 Use Case Scenario

Shown in Fig. 1 is a use case scenario on how actors are involved in obtaining a ticket from a train ticket booking clerk.

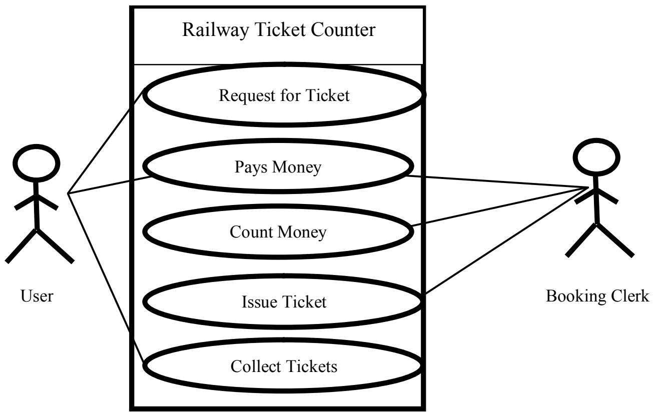

Fig. 1. A use diagram showing how a ticket can be obtained at the counter by a customer in a railway ticket office

Though use case is drawn as illustrated in Fig. 1, a lot needs to be said to make use case diagrams more useful to the developer. They need to be written in standard format and documented properly for use in development. In fact, this use case document is an important voluminous document, referred to very often for an understanding of the system because they give detail of a scenario.

4.2 Benefits of Use case-Driven Development to Developers

Use case describes the scenario in terms of transactions, users and system performance. It is based on the user tasks and explains user activities in the natural English language. In addition to the aforementioned, use cases help to:

- Define the scope of the use case, use cases and applications and system.
- Measure the size of the software development projects in a number of use cases.
- Tract and monitor progress of the development in terms of use cases.
- Validate the requirement of the system as told by the user.
- Design test cases by each use case to ensure completeness.
- Ensure complete system documentation, as all use cases are included with model, design and details.
- Reduce the user's need for training and hand-holding in learning to operate the system.
- Reduce time for system maintenance.
- Obtain user's fast acceptance of the system.

Though use cases does not solve all problems of system development. It should not be taken as a panacea for all issues.

5 Diagram

Class diagrams are used in both analysis and design phases of a software life development cycle [2]. During analysis phase, a very high-level conceptual design is created. At this time, a class diagram might be created with only class names shown or possibly some pseudo code-like phrases may be added to describe the responsibilities of the class. The class diagram created during analysis phase is used to describe the classes and relationships in the problem domain, but it does not suggest how the system is implemented. By the end of the design phase, a class diagram that describes how the system is implemented is developed. The class diagram created after the design phase has detailed implementation information, including the class names, methods and attributes of the classes, and the relationships among classes as shown in Fig. 2.

Class diagram describes the types of objects in a system and the various kinds of static relationships that exist among them [6]. In UML, a class is represented by a rectangle with one or more horizontal compartments. The upper compartment holds the name of the class. The name of the class is the only required field in a class diagram. By convention, the class name starts with a capital letter. The (optional) center compartment of the class rectangle holds the list of the class attributes/data members, and the (optional) lower compartment holds the list of operations/methods.

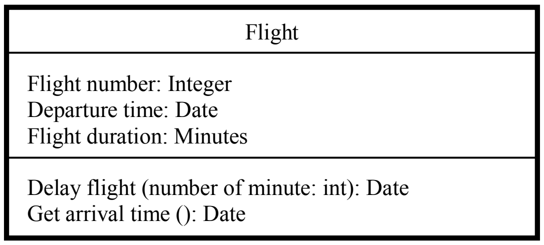

Fig. 2. Class diagram for a class name flight

In Fig. 2, the class name is Flight, with three attributes and their types: flight Number with its type as integer, departure Time with its type as Date and finally, flight Duration with its type as Minutes.

6 Behavior Diagrams

Most software systems consist of sub systems that interact with each other to produce the desired result [1]. It therefore becomes necessary to model the behavior of objects in association with other objects in the system. When a system performs in real time, many objects come into play. They send massages to each other and activate the methods in the object. When a method is chosen and performed on the data, the objects status changes. The static objects assume a dynamic state. A state of the objects is the result of its behavior, which changes dynamically. UML diagrams showing the behavior of objects are called Behavior Diagrams. The most important and common ones are:

- Interaction diagrams (sequence and collaboration diagrams)
- State chart diagrams
- Activity diagrams

6.1 Interaction Diagrams

The purpose of an interaction diagram is to understand the role of other objects that are in collaboration with the objects in question. Object behavior can be better understood in software modeling if the sequence of interaction between objects and collaboration between them is analyzed [1]. Two special UML diagrams to analyzed interaction behavior of objects are sequence and collaboration diagrams.

6.1.1 Sequence diagram

Sequence diagrams indicate how events cause transaction from objects to objects [7]. Once events have been identified by examining a use case, the modeler creates a sequence diagram representation of how events flow from one object to another as a function of time. In essence, a sequence diagram as drawn in Fig. 3 can be seen as a shorthand version of a use case. It represents a key class and events that cause behavior to flow from class to class.

A sequence diagram uses the following notations: A rectangle shows objects that participate in the behavior. Existence of the objects in the behavior is shown by a vertical dotted line. The horizontal arrowhead line shows massage ‘from-to’ massages. The horizontal lines are arranged in the sequence of their occurrence.

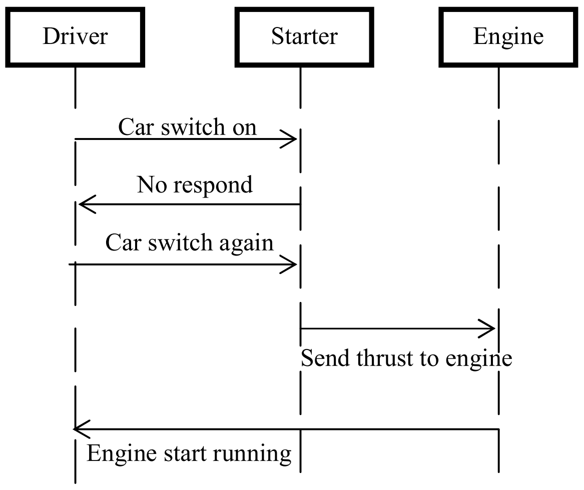

Fig. 3. Sequence diagram of a use case ‘starting a car’

6.1.2 Collaboration diagram

Unlike the sequence diagram which show the task or activity sequence but do not show the relationship between objects through the roles that they play in the interaction. Collaboration diagram as shown in Fig. 4 indicates the objects that collaborate to perform the role based on the massage received.

Both sequence and collaboration diagrams are alternatives to each other. The sequences diagram is easier to understand but does not show the flow of behavior and collaboration of objects. Collaboration shows both collaboration and sequence of activities. Sequence diagram are also easy to read. The collaboration diagram shows the interaction between objects and sequence of activities denoted by numbers.

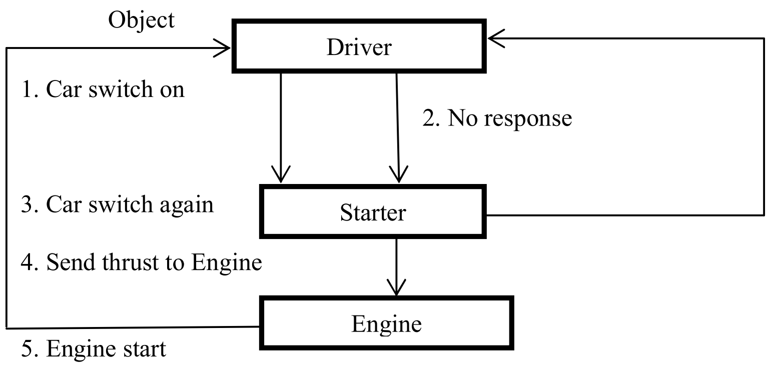

Fig. 4. A collaboration diagram for starting a car

The main disadvantages of interaction diagram are that in complex interaction, with couple of objects, the diagrams are difficult to draw and read. Also interaction diagram are not good enough if the behavior of the object is conditional and gets into loop.

7 State Chart Diagram

A state diagram is one method for representing the behavior of system by depicting its state and the events that cause the system to change state. A state is any observable mode of behavior. In addition, the state diagram indicates what actions are taken as a consequence of a particular event. The advantage of the state chart diagram is that it provides an understanding of the process and algorithm used in the method.

A state chart diagram as shown in Fig. 5 has state at every point in time. An entry event is performed on entering the object state. An exit event is performed on exit from the object state. Transition indicates the relationship and the difference between two objects states caused by events. Transition can be simple or complex. A final state is shown by the terminal symbol.

As shown in Fig. 5, the object gate moves from the state open to close to red light and back to gate reopen state. These states are triggered by events like bell ring, red light on and bell stops ringing.

8 Activity Diagram

An activity diagram is concern with activities and actions that drives events or that change the object from one state to another. Similar to the flow chart, it provides a flow of interaction with specific scenario. An activity diagram as depicted in Fig. 6 uses rounded rectangle to imply specific system function, arrows to represent flow through the system, decision diamonds to depict branching decision and solid horizontal lines to indicate that parallel activities are occurring.

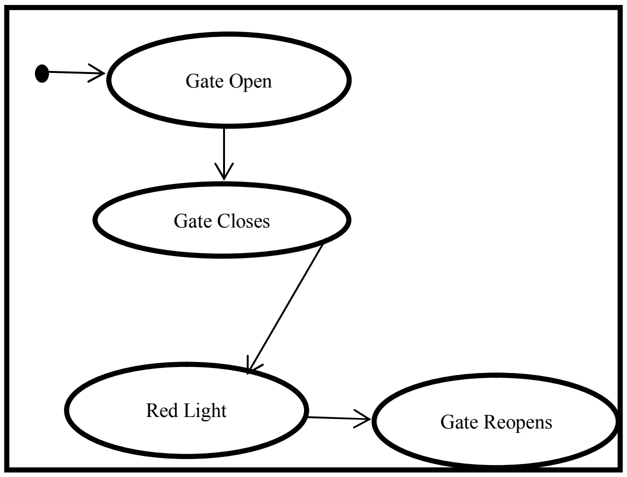

Fig. 5. State chart diagram showing gate operation at a railway crossing

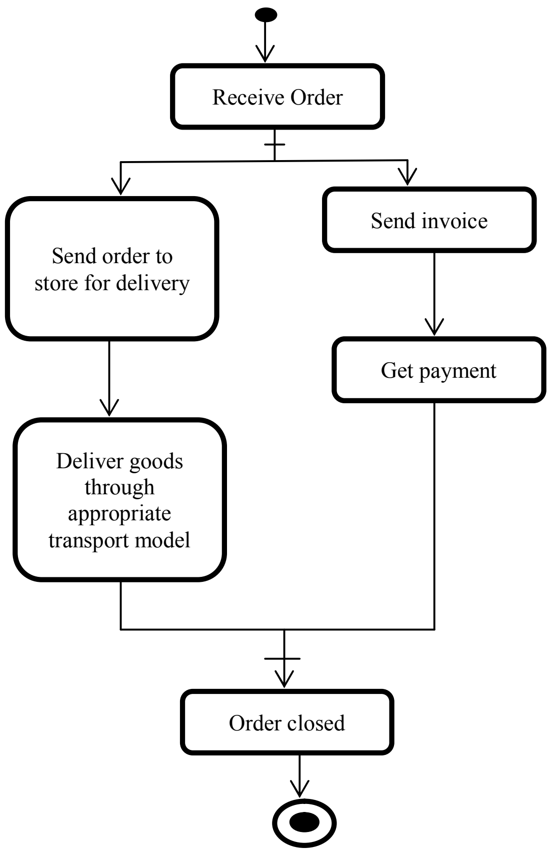
Receive Order
Send order to store for delivery
Send invoice
Get payment
Deliver goods through appropriate transport model
Order closed

Fig. 6. Activity diagram processing an order to deliver some goods

9 Implementation Diagrams

There are two types of implementation diagrams, a component diagram to show the structure of the code, and deployment diagram to show the structure of the runtime system.

9.1 Component Diagram

A component diagram (see Fig. 7) is a graphical model of the design's components connected by dependency relationships. A component is shown by a box and dependency by dotted line.

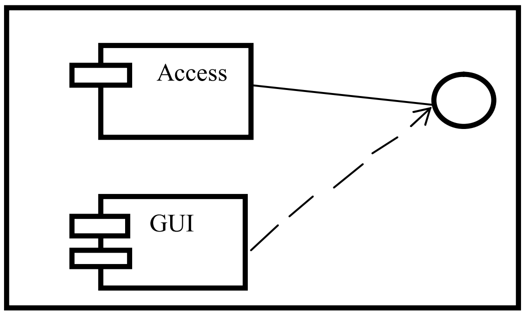

Fig. 7. A component diagram indicating an access component depending on the GUI component to be updated

9.2 Deployment Diagram

Deployment diagrams show runtime configuration of the system on the customer's environment. A deployment diagram is shown in Fig. 8.

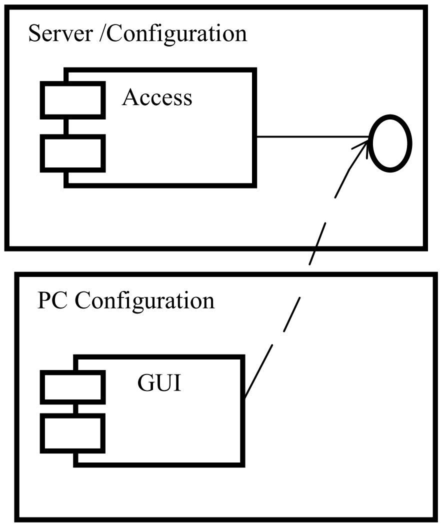

Fig. 8. Deployment diagram of the of the component diagram in figure

10 Future Work

Modeling of software is beyond drawing of UML diagrams alone. UML diagrams most depict and translate to the user's requirement. How these relationships are possible need to be shown in future work. Also based on experience in the software industry, there exists other ways in which UML diagrams are abuse. The study will in the future interview software developers to uncover other ways that UML are abuse.

11 Conclusion

Standardization is a good practice in everything we do in life. It therefore means that using standard UML diagrams will help improve software development because of easy understanding between developers. Also, understanding the abstraction nature of program logic is necessary in drawing UML diagrams. Furthermore, training of software developers who are not conversant with the use of UML aided design tools is necessary. It can be conveniently concluded that good UML diagrams aid a good software development and this can be achieve if modeling of any system using UML design is done in an understandable and transparent way such that every developer can easily understand the UML diagrams.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Waman SJ. Software engineering principles and practice. Tata McGraw-Hill Publishing Company Limited, New Delhi; 2004.

[2] Laurie W. An introduction to the unified modeling language; 2004.
Available: http://agile.csc.ncsu.edu/SEMaterials/UMLOverview.pdf (Accessed on 29th Dec, 2014)

[3] Donald B. UML diagram: The class diagrams. An introduction to structure diagrams in UML 2, IBM Corporation; 2004.

[4] Harlad S, Alexander K. Unified modeling language 2.0; 2006.
Available: http://www.pst.ifi.lmu.de/veroeffentlichungen/UML-2.0-Tutorial.pdf (Accessed on 16th Feb., 2014)

[5] Darius S. Best practices for applying UML. Part I. Magic Draw Inc; 2014.

[6] Bruegge B, Dutoit H. Object-oriented software engineering: Conquering complex and changing systems. Upper Saddle River, NJ, Prentice Hall; 2000.

[7] Rodger SP. Software engineering: A practical approach. sixt edition, McGraw Hill, New York; 2005.

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
http://sciencedomain.org/review-history/9919