1.Vaishali CHOUREY, 2.Meena SHARMA

FUNCTIONAL FLOW BLOCK DIAGRAM WITH UML FOR TEST VISUALIZATION

1.IET- DAVV, Indore, Madhya Pradesh, INDIA

Abstract: The use of UML models for testing context is gaining momentum in the existing model based testing scenario. Appending the model semantics and annotation are important mechanisms to represent functional and nonfunctional properties of design elements. This also leads to evolution of new models preserving the semantics of underlying models like UML as reference. In current research such models depend on complex mathematical and graphical notations for its realization. This makes model evolutions or transformations difficult to generate and tedious to use. However, if the development model is adopted for representing test conditions, it will bridge the semantic and operational gap between developer and tester. The structural composition and the interaction pattern during runtime are different aspects of visualization. The analysis of system requires both interpretations to conform for expected behavior. This will serve as an insight for test preference also. This paper focuses on developing semantics of UML applicable to visualize tests. This includes an abstract notation, Functional Flow Block Diagram (FFBD) that incorporates the flow semantics into component diagram to better visualize the execution context. This diagram is adopted from UML and is drawn with intent to visualize test requirements of the system. An important application for reliability analysis is the motivation to the work. The FFBDs are easily mapped into Reliability Block Diagram (RBD) that associated reliability constraints to the diagram. This facilitates visualization of quality which is the contribution of the paper.
Keywords: UML models, Functional Flow Block Diagram (FFBD), Functional Flow Block Diagram (FFBD)

INTRODUCTION

Testing is an activity that assures successful execution of functionality. It focuses on validating the requirements with possible input and achieving desired behavior. Systems generally fail due to incomplete or incorrect interactions which were not estimated during design phase. These interactions include sharing of data or functionalities within or beyond a system. This, however are not explored unless executed. Testing designs for early detection of flaws is a matter of discussion since years. The understanding of the structure of the system, data that flows through various components and the dynamic nature of software today need to be understood. Various diagrams in the UML have significantly accommodated structural and dynamic constraints of execution. When we evaluate the software architectures today, we find that the demand for quality (not only functionality) is the concern of engineers. The applications draw data from various sources, manipulate them with various algorithms for a wide range of applications. In the view of customer, the system needs to be reliable, secure and robust for giving the performance up to mark. The responsibility of the developer and the tester has escalated simultaneously. New methods for the evolving architectures and the testing strategies need to be devised to deliver quality with functionality.

The current scenarios in software development exploits reuse oriented design patterns that are deployed to fulfil the functional requirements of the system. They are more of coalition based systems which come from varied sources, technologically and vendor based. The concept of design is casually taken with a fact that they do not adapt to implementation requirements and do not scale or evolve with code. They even lack updated versions while the interaction of components is required from the various sources. Our paper is intended to handle these exceptional conditions of designs. The beauty of the design is however versatile to answer any such query. Software reuse makes existing components fit into the design. The quality of the components taken together form the metrics of quality of the system. A decision-making capability can be drawn during design stage to select components of high quality to achieve a better software. The functional attributes along with non-functional ones can be annotated to serve the purpose. The strategy will enable design to predict the capabilities of the system in terms of quality. The challenge is to define such annotations in the semantics of existing models to make them applicable in industry practices. The choice of UML models is the preference in our work. These models have grounded concepts through UML infrastructure and specifications formally defined and used as standards. The capability of the UML to extend gives a suitable dimension for framing our concept into modeling terminologies.

Several models have been used in the analysis of software. The models for testing applications are an interesting research in the subject area. Several researches have been made to employ one or another diagram out of UML for the testing signifying the area of Model Based Testing. The area is emerging with new technologies in development. The challenges also float with trends to mark thrust in the research in the state of art of model based testing. May it be service oriented systems or designing models or metamodels for them, the concern for quality is an upright choice. A general overview of the various issues pertaining to the model based testing is presented in the figure 1. The concept of component based testing is relevant to our paper and the domain in which it shall be applicable is shown in figure 2.

Testing is important for quality. A software requirement if marked with perspectives and so the testing must also conform to the same perspectives. Sufficiency is attainted when all the aspects of testing are fully covered and most of the code is verified. The test oracles with specific test criteria and proper planning for test process are important. With the concepts of object oriented and component based testing, major areas of research has evolved like aspect oriented, domain oriented or performance testing. This is reflected from the new architectures to develop software. Thus a challenge for the existing design models is to cope up with the new architectures.

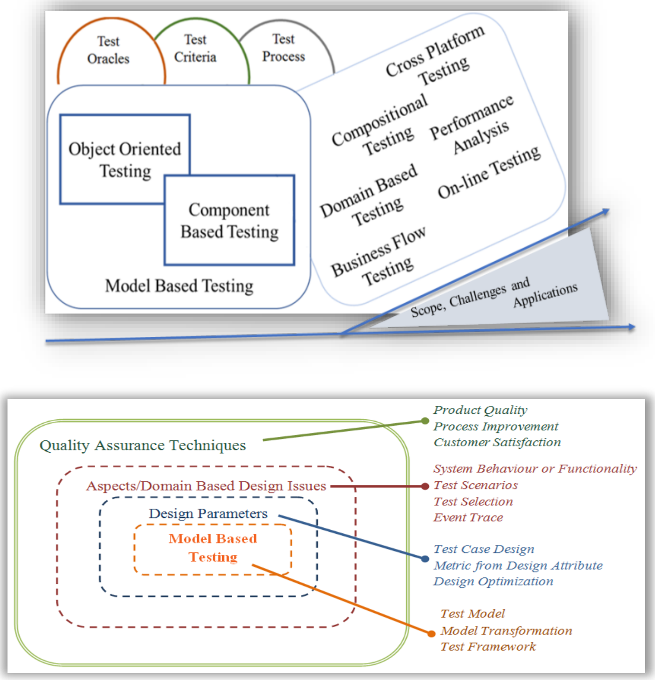

Figure 1: Scenario of model based testing and its application in current research

Another challenge while using models for testing is reading its specification to extract relevant test information [18,19]. The UML specifications effectively model systems with formalism in syntax and semantic rules. The syntax of the specification defines elements of the UML in abstract form as diagram and also as textual descriptions as annotations. The semantic are descriptions in the form of OCL and other constraints. The model properties are extracted and used in test automation methods. Several functional and non-functional characteristics are tested through such annotations and contribute to an essential characteristic of UML that is well-formedness. Description logics [22,23], multi-diagrams [24], are alternatives of knowledge representation and semantics of UML models. The next section details the research based on various models and acknowledges the valuable contributions.

The paper is organized as follows. Section 2 reviews and acknowledges past research in the domain. Section 3 proposes the methodology of deriving a new representation of system. Later in the same section are the algorithmic based specification derivation and its use in test case preparation. It also gives the interpretation of the FFBD for reliability assessment. Finally, in Section 4 the model is created and results are expressed. Section 5 is the conclusion that describes the essential gain from this technique and its scope to automate the process with the newly derived functional flow notation.

RELATED WORK

A critical part of development involves testing the code. Developments in life cycles for Agile Methodology with Test Driven Development, Test Design First Development, Behavior Driven Development, have made testing a decisive activity. Model Based Testing [13, 14] rules with prior system estimates before implementing it. Models in UML are classified structural and behavioral to represent the system with different aspects. Certain algebraic methods and formal / informal models exists that are based on mathematical analysis of model properties and their interconnects. These models are more analytical for research and so fail to be easily adopted for software development practice methods. But to highlight the contributions, it is the activity diagram that is most suitable to understand the flow and then can easily be modulated with graphs for assessment [1-3,11,12]. State diagrams [16,17] also contribute to test case generation based on coverage criteria. Class diagrams [20,21,10] are important to understand the structure of the system, its dependency with other parts or modules of the system, to estimate complexity in realizing the behavior through the collection of classes. These diagrams as stated earlier lead to large amount of tests that are derived from the transactions stated in requirements [5]. The code thereby developed also comes in several iterations with subsequent testing and corrections thereafter. The drawback is that the design does not get updated with it. So the testing stage has the code to test and the design faces neglection.

A combination of diagrams is sometimes adopted to generate tests. A mapped class and state chart [4,7] or an aggregate notation from sequence and collaboration [8] is seen in papers. Model based tests does not restrict to identifying defects only but has a broader value in terms of creating a system model, its analysis in terms of inputs and outputs, reporting flaws in design, formal report generation and generating test requirements for future phases [13]. This is an intelligence imbibed in design which even allows to choose appropriate from a marketplace to create new systems.

In order to generate the diagrams, the architectural constraints must be clearly stated. This depends on the type of software being developed. The quality is an umbrella feature for any software. UML defines QoS and concepts of fault tolerance as the extensions of UML profile and specifications. The attributes of quality can be within a QoS framework and defined appropriately. The same notation can even extend to define attributes like reliability in the same UML notation. The paper actually contributes to the test requirements and quality to represent in developer's conversant modeling format like UML. The practice for quality and its inclusion in early life cycle stages is important to achieve quality throughout the system phases. An example for which is CISQ. The Consortium of IT Software Quality (CISQ) defines certain quality characteristic measures for nonfunctional aspects such as reliability, performance efficiency, security and maintainability. There are recommendations for incorporating quality into development and includes certain best practices for achieving it. For instance reliability can be achieved by good coding practices like protecting state in multithreaded environments, safe use of inheritance and polymorphism, resource bound management and managing the allocated resources. The same can be stated in architectural practices like multilayer design compliance, managing data integrity and consistency, exception handling through transactions, class architecture compliance and so on. Similarly the QoS characteristics and constraints can be placed in UML metamodel to define modeling for quality. QoS characteristics are quantifiable attributes of system and logically constructor for non-functional aspects of quality like latency, throughput, availability, reliability, safety, confidentiality, integrity, probability and accuracy.

The paper explores possibilities to embed these QoS characteristics in design properties for evaluating these parameters in the design stage itself. The next section is a formal definition of our proposed model. The semantics are of UML and scoped to future aspects like sustainable designs.

PROPOSED METHODOLOGY

Models are the artifacts generated and used in the stages of software development. As in conventional systems, all the diagrams that pertain to system representation are drawn. Other than these diagrams, a new diagram inspired from System Engineering Flow Diagram is to be drawn which is illustrated later in this section.

Functional Flow Block Diagram is the formal model that encompasses the components participating in execution context. The notation is actually adapted from the activity diagram, component diagram and system engineering flow diagram. It characterizes activity diagram with its events and the sequence in the flow from beginning of the transaction to its end. This is a most appropriate way to depict the dynamics in the system with available se of components in the design. It also borrows certain features of the class diagram to preserve its structural composition in place.

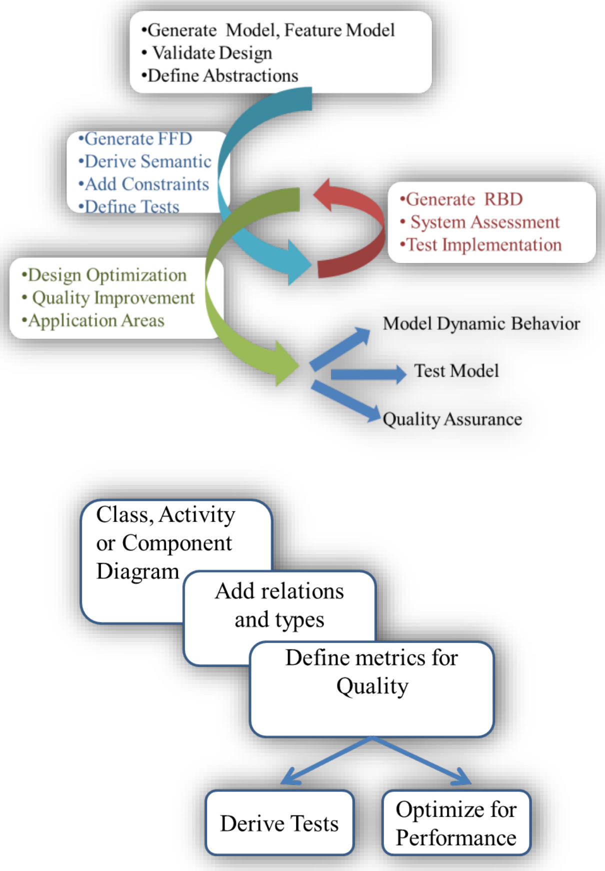

Figure 2: Proposed Methodology for Test Modeling

Our model aggregates the details from both the diagrams and creates explanatory diagram similar to a component diagram with greater details. The process of generating the right model streams from the requirements started in the early analysis phase. The usual practice is specifying the functional requirements in the use cases and realizing it in the activity diagram. We also annotate the non-functional requirements from the beginning so that it is also incorporated in the design. For the same we define our feature model along with use cases. Now the functionalities are drawn hierarchically in the feature model. We define the levels in our feature diagram to bring it in an n-tier architecture. The feature model will somehow represent the component that invokes a set of another classes or components while execution. The level of abstraction resolves to a fine-grained level each time such invocations occur.

The second stage evolves the Functional Flow Block Diagrams with the semantics as defined in SysML for blocks. The same can be easily transformed to reliability block Diagram to model reliability within the design. The assessment is done through this RBD.

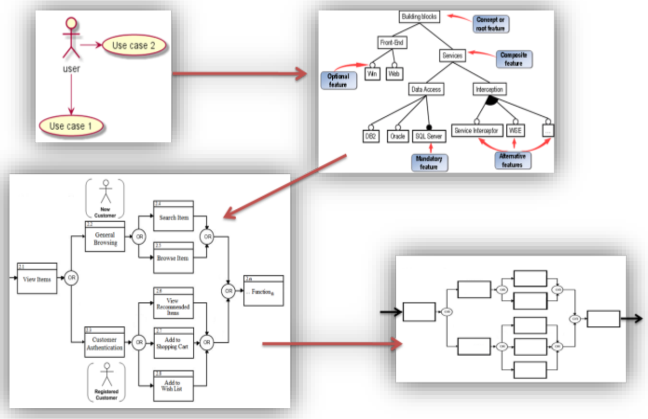

Figure 3: Stepwise Model Transformation

The abstract syntax for Block as specified in the SysML standards is SysML: Blocks: Block which is characterized with constraints, operations, parts, references, values and properties. A SysML block defines a system with its features. The Block contains compartments for each of its characteristics. For example constraints are written to specify any physical constraints in the system. The constraints are the places where we can define the exceptional conditions about the execution of the component. The blocks are further incorporated with internal blocks and their properties. The SysML notation [SysML] is shown as in figure

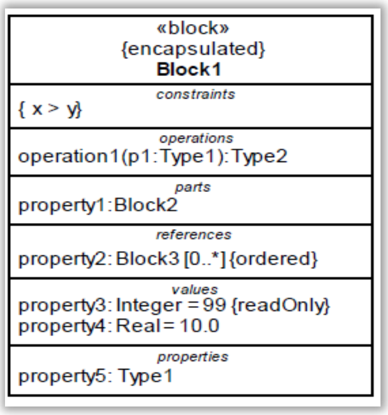

Figure 4: SysML Block

The whole process is a four-stage model transformation strategy which needs formal annotations. The UML semantics for quality has been well formulated as QoS Profiles and we annotate the design elements with the syntax as mentioned with the profile. The parameters for quality are broadly defined and are referred as characteristics. This is described as QoS Characteristics with QoS values. For example, failure occurs when the behavior of system differs from the intended behavior. Fault in the same way is the adjudged or hypothesized cause of an error. The OCL notation for the same is defined for fault and failure as QoS characteristic with domain, perception and consequences.

The same semantics for reliability is stated as <<QoSDimension>>Reliability with the attributes as expected-number-service-failures: {direction(decreasing)} and <<QoSDimension>> operation semantic:string type. It is characterized with the value estimate of failures to define the ability of the system to keep operating correctly over time.

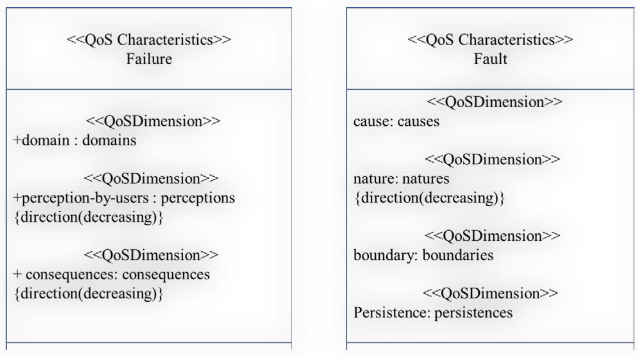

Figure 5: QoS Characteristics in UML Profile

Availability is syntactically annotated as:

<<QoSCharacteristic>>Availability

<<QoSDimension>>

time-to-repair:real

{direction(decreasing), statisticalQualifier(mean)}

<<QoSDimension>>

time-between-failures:real

{direction(increasing), statisticalQualifier(mean)}

In the context of availability ie context availability:: availability-post result OK: result = time-between-failure / (time-between-failures + time-to-repair)

<<QoSDimension>>

availability-value()

{direction(increasing), statisticalQualifier(mean)}

Another characteristic is fault tolerance which is also defined as <<QoSCharacteristic>>fault-tolerance

<<QoSDimension>>

max-number-of-faults:

{direction(increasing, statisticalQualifier(maximum)}

<<QoSDimension>>

+error-processing:error-processings

{direction(increasing)}

<<QoSDimension>>

Fault-treatment:fault-treatments

{direction(increasing)}

Once the profile for the component is ready the values pertaining to the attributes are evaluated for the design of system. The actual values attained are used for deciding the component for its inclusion in design. In case the component needs to be backed with error detection, error control and fault treatment properties, then the annotation is marked as model property. This reference to characteristics of QoS in design of system based on reusable components enables test prioritization [26] for such components so that reliability is ensured. Regression testing thus has a ranked set of components based on such properties to clearly state critical functions.

RESULTS

The test visualization is easily managed through the diagram FFBD and the assessment of non-functional requirements of quality is also performed. The structural and dynamic behavior of the system in line with system engineering conventions is applied for software systems here. The large-scale systems can easily be decomposed into hierarchical components and defined within abstractions.

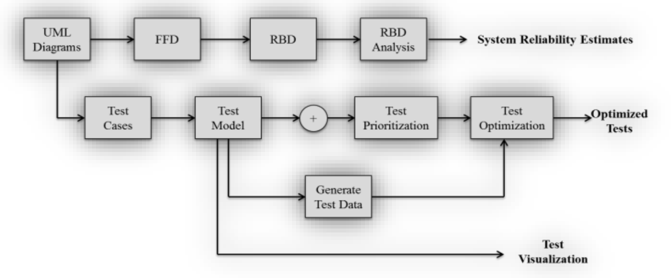

Figure 6: Assessment Model [24]

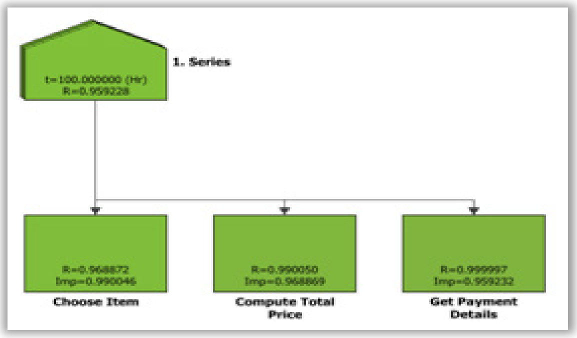

Figure 7: Reliability Importance for Blocks [24,25]

Thus, a complex system is fit into a single view of representation here. The reliability estimates for components are generated that can be annotated into the design. This leads to additional attribute which defines suitability of that component to fulfil quality requirements of the system. Rather than complex mathematical illustrations, the probabilistic modeling confines to a simpler method of assessment. The scope of the paper restricts the implementation related analysis, but the methodology is hereby suggested for the assessment and visualization. The reliability analysis leads to a parameter of Reliability Importance which prioritizes the blocks or components for testing purpose also [figure 7]. This is rephrased in the model properties as model characteristics and subsequently used.

CONCLUSION

This paper focuses on developing FFBDs in the semantics of UML to visualize tests. This includes defining Blocks in its abstract notation then incorporating the flow of control into the definition of components to visualize the execution. This FFBD has been referred from SysML and is drawn with intent to visualize test requirements of the system. An important application for reliability analysis is presented in this paper k. The FFBDs are easily mapped into

Reliability Block Diagram (RBD) for further analysis of reliability of the system. As this is a part of testing, therefore we recommend our methodology for the test based visualization in model based design and testing. This may further be applied with agile or test driven development strategies for evaluation.

References

[1] Debasish Kundu and Debasis Samantha, A Novel Approach to Generate Test Cases from UML Activity Diagrams, Journal of Object Technology, June 2009.
[2] Dong Xu, Huaizhong Li, ChiouPeng Lam, Using Adaptive Agents to Automatically Generate Test Scenerios from the UML Activity Diagram, IEEE Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05), 2005.
[3] AnnamarialeChandran. Model Based Testing – Executable State Diagrams, Published in STEP-AUTO 2011, AVACorp Technology, Chennai.
[4] PVR Murthy, PC Anitha, M Mahesh, Rajesh Subramanyan. Test Ready UML Statechart Models, SCESM '06 Proceedings of the International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools, 2006.
[5] Bernhard K. Aichernig, HaraldBrandl, Elisabeth Jobstl, Willibald Krenny. UML in Action: A Two-Layered Interpretation for Testing, Published in ACM SIGSOFT January 2011 Volume 36 Number 1.
[6] Mingsong Chen, XiaokangQiu, Wei Xu, Linzhang Wang, Jianhua Zhao And Xuandong Li. UML Activity Diagram-Based Automatic Test Case Generation For Java Programs, The Computer Journal Advance Access published August 25, 2007.
[7] OrestPilskalns, Andrew Knight. Testing UML Designs, Published in Information and Software Technology (49), 2007, Science Direct.
[8] Rumpe,B. : Agile Test Based Modeling. In: Proceedings of the 2006 International Conference on Software Engneering Research and Practice. (SERP). Volume 26. (2006) 10-15
[9] KenroYatake, Toshiaki Aoki. SMT Based Enumeration of Object Graphs from UML Class Diagram, ACM SIGSOFT Software Engineering Journal, July 2012 Volume 37 Number 4.
[10] Wang Linzhang, Yuan Jiesong, Yu Xiaofeng, Hu Jun, Li Xuandong and ZhengGuoliang. Generating UML Test Cases from UML Activity Diagram Based on Gray Box Method, Proceedings of 11th Asia Pacific Software Engineering Conference, IEEE '04.
[11] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem and Mohamed F.Tolba. A Proposed Test Case Generation Technique Based on Activity Diagrams, International Journal of Engineering & Technology IJET-IJENS Vol: 11 No: 03 June 2011.
[12] Jonas Boberg, Early Fault Detection with Model-Based Testing, Erlang'08, September 27, 2008, Victoria, BC, Canada ACM 2008.
[13] MrkUtting, Position Paper: Model Based Testing.
[14] Santoshkumar Swain, Model Based Object-Oriented Software Testing, Journal of Theoretical and Applied Information Technology (JATIT 2005-2010)
[15] Richard Torkar, Robert Feldt, and Tony Gorschek, Extracting Generally Applicable Patterns from Object-Oriented Programs for the Purpose of Test Case Creation.
[16] Harry M. Sneed, ANECON GmbH, The Drawbacks of model-driven Software Evolution, 2010.
[17] Philip Samuel, Rajib Mall.: Slicing Based Test Case Generation from UML Activity Diagrams. In: ACM SIGSOFT Software Engineering Notes, Volume 34, Number 6(2009)
[18] A Cavalli, S.Maag, S. Papagiannaki and G. Verigakis,: From UML Models to Automatic Generated Tests for the dotLRN e-learning Platform. In: Electronic Notes in Theoretical Computer Science, 116:133-144(2004)
[19] Z.M.Ma, Li Yan, Fu Zhang.: Modeling Fuzzy Information in UML Class Diagrams and Object Oriented Database Models. In: Journal of Fuzzy Sets and Systems 18626-46 (2012)
[20] Tong Yi, Fangjun Wu, ChengzhiGan.: A Comparison of Metrics for UML Class Diagrams. In: ACM SIGSOFT Software Engineering Notes, Volume 29, Number 5 (2004)
[21] Berkenkötter, Kirsten. "Reliable UML models and profiles." Electronic Notes in Theoretical Computer Science 217 (2008): 203-220.,
[22] Z.M. Ma, Fu Zhang, Li Yan, Jingwei Cheng, Representing and reasoning on fuzzy UML models: A description logic approach, Expert Systems with Applications 38 (2011) 2536-2549
[23] Luciano Baresi, Angelo Morzenti, Alfredo Motta, Matteo Rossi, A Logic-based Semantics for the Verification ofMulti-diagram UML Models, ACM SIGSOFT Software Engineering Notes July 2012 Volume 37 Number 4
[24] Chourey V, Sharma M. Component based reliability assessment from uml models. InAdvances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on 2015 Aug 10 (pp. 772-778). IEEE
[25] Chourey V, Sharma M. Functional flow diagram (FFD): semantics for evolving software. In Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on 2016 Sep 21 (pp. 2193-2199). IEEE.
[26] Strandberg, Per, Wasif Afzal, Thomas Ostrand, Elaine Weyuker, and Daniel Sundmark. "Automated System Level Regression Test Prioritization in a Nutshell." IEEE Software (2017).

ACTA TECHNICA CORVINIENSIS BULLETIN OF ENGINEERING
ISSN: 2067-3809