Modeling RTCA DO-178C Specification to Facilitate Avionic Software System Design, Verification, and Validation

Emanuel S. Grant and Tanaya Datta

***Abstract*—Avionic systems are safety-critical systems because system failure can be catastrophic resulting in loss of life and/or resources. Consequently, safety-critical systems must be developed carefully and adhere to standards such as DO-178C. As these types of systems need to maintain critical mechanisms, some of the mechanisms can be complicated and difficult to use appropriately, written descriptions can be vague and error prone. This paper aims to explore graphical specification method as alternative to represent the textual DO-178C. Several UML diagrams are created in order to represent DO-178C in a format that is easier related to model-driven software development. Specifically, UML Package Diagrams, Activity Diagrams and Class Diagrams are used to illustrate the various processes, sub-processes, activities and contents as defined in the DO-178C specification.**

***Index Terms*—DO-178C specification, model transformation, safety-critical systems, UML diagrams.**

I. Introduction

The impacts of safety in avionic systems that comprise software and hardware integration have to be considered at system development level. For this reason, such systems should be planned and developed in accordance with airborne software considerations and system standards. DO-178C is the "Software Considerations in Airborne Systems and Equipment Certification" [1]. DO-178C is the modified edition of DO-178B that was published in December 2011. DO-178C has the purpose similar as its predecessor, which is "providing guidance for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements" [2]. There are changes in DO-178C which tighten some previously established controls and also established concrete guidance for more flexibility in development approaches. These flexibility includes examination of costs and benefits, establishment of certifiable product development approach [3].

The RTCA DO-178C specification "Software Considerations in Airborne Systems and Equipment Certification" is an internationally accepted document that sets out guidelines to plan, design, development, and certify, software systems for the avionic industry. In a research project to develop a model-driven object-oriented software development methodology, for avionic systems it was determined that the textual format of DO-178C was a hindrance to the research effort. Consequently, work was directed at developing an alternative representation of DO-178C that would complement and enhance the research on developing the aforementioned methodology. The approach taken is to transform the textual representation of DO-178C into a set of UML (Unified Modeling Language) [4] models that would be in the same representation models of the research methodologies. The transformed representation of DO-178C would be in the form of UML package, class diagram, and activity diagrams. This would allow for a seamless transition from the DO-178C specification to the model-driven object-oriented software development methodology. The goal of this research is to develop a subset of the UML model representation of DO-178C as a precursor to defining the research methodology.

This paper is a description of a particular section of "RTCA DO-178C Software Considerations in airborne Systems and Equipment Certification guidelines" titled Software Configuration Management Process [1]. This is one of the integral processes mentioned in this guideline in Section 7.0. This research transforms the textual description of this section into UML notation in the form of package diagrams, class diagrams and activity diagrams.

The structure of the remainder of the paper is as follows. Section II presents a background of the research project.

II. Background

A. Software System Modeling

Modeling is a problem solving technique that is popular in many engineering disciplines [4], [5]. A model is defined as a representation of a system from a specific perspective that is intended to promote understanding of the real system and to provide the capability to serve as a high fidelity validation test beds prior to construction of the potential system [6]. The advantages of the modeling method include understanding complexity, evolution, and manipulation of the modeled systems, and simulations. Thus, models are used to better our understanding of fundamental processes. This can be useful in policymaking and interventions, concerning efficient training, scenario development, risk assessment, forensic analysis, and future trends prediction; resulting in simplifying the problem

Manuscript received February 28, 2016; revised April 18, 2016. This work was supported in part by a grant from the University of North Dakota Faculty Seed Grant Program 2014.

E. S. Grant is with the University of North Dakota Department of Computer Science, North Dakota, ND 58202, USA (e-mail: grante@aero.und.edu).

T. Datta is with the Department of Computer Science, the University of North Dakota, North Dakota, USA (e-mail: Tanaya.datta@und.edu).

solving processes for complex systems.

The benefits of modeling have made it a popular problem solving approach in many disciplines, including the engineering disciplines where it has brought forth the Model Driven Engineering concept (MDE) [7]. MDE refers to Engineering approaches that use models as primary artifacts through the engineering lifecycle. Engineering models are built and tested before development of the real system. The use of models in engineering processes have significantly minimized errors and reduced the cost of the final product. In the Software Engineering field [8], Model Driven Development (MDD) constitutes the MDE vision of software development processes [9]. The MDD approach is characterized by the use of modeling languages and formal specification methods to describe the software system under development; analogous to other engineering disciplines where the created models can be tested and validated before the actual system is developed.

A need for standards arose from the various implementations of MDD. The need for standards was addressed by the Unified Modeling Language (UML) [7]: a general purpose standardized specification language for object oriented software design. UML uses a set of graphical notations to create an abstract model of a system. In addition to providing a standard for MDD with a wide variety of diagrams at many levels of abstractions, UML offers different views of a system model, which facilitates emphasis on a particular aspect of the modeled software product. With its visual communication, various levels of abstractions, and the modeling of different views of a system, UML has become the de-facto standard in the software development industry. Empirical evidence supports the claim that UML is an effective approach to modeling software systems [10]. The popularity of the UML standards gives evidence of usefulness and lower difficulties of implementation as it pertains to the development of new software systems.

The growing complexity of the software is the motivation behind model driven development. The model driven development concentrates on describing the complex systems at multiple levels of abstraction in the form of models and using the models in implementation. Using the model driven software development for space payload systems decreases the complexity and helps in efficient software implementation and testing. The objective of this project is to design and implement a series of UML models that presents a graphical view of the DO-178C specification [1].

B. Unified Modeling Language (UML)

The Unified Modeling Language (UML) is the ISO standard for designing and conceptualizing graphical models of software systems [11]. Since its development by the Object Management Group (OMG) in the early 1990's its use has proliferated in both industry and academia. Graphical software models, such as UML models, possess simplistic designs and promote good software engineering practices.

In the traditional approach to software engineering, graphical models would precede code generation. However, it is common for a prototype to preexist. In such scenarios, reverse engineering activities are used to derive the graphical models.

The UML is an object-oriented modeling language for specifying, visualizing, constructing, and documenting the artifacts of software systems [11]. The UML is used to depict a high-level representation of the proposed system. This is achieved through the design of various types of models, which capture the structure and behavior of the system. Diagrams in UML are categorized as structure or behavior diagrams. Structure diagrams represent the static framework of the system; whereas behavior diagrams depict the dynamic features of the system. These informal models have an advantage of being expressive – which makes them easily conveyed to both technical and nontechnical stakeholders.

This work focuses on use of the UML class and activity diagrams:

Class diagram [11] - is used at the analysis phase to present a view of the static entities in the problem domain, and at the design phase to present a view of the static entities (classifiers) in the solution domain. This is the UML diagram used to captures the static information at the requirement phase of software development.

A class diagram (CD) is described as ". . . a graph of Classifier elements connected by their various static relationships." [11]. The set of classifier elements that may be present in a CD include interfaces, packages, relationships, instances, and links, etc. The UML labels a CD that contains no classes an object diagram (OD). An example of a UML CD is illustrated in Fig. 1.

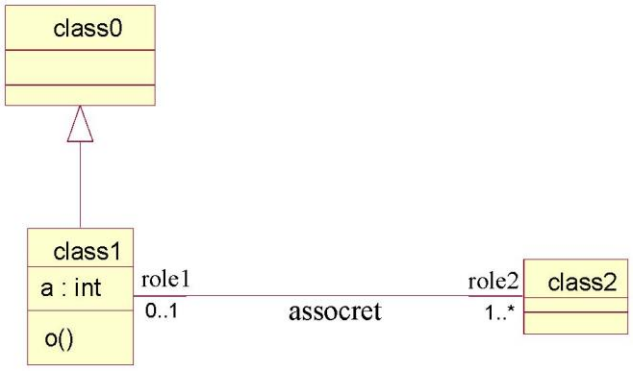

Fig. 1. UML class diagram example.

Activity diagram - is used to present the control or data flow of a process. An activity diagram is a type of state machine, where the states are represented by actions or sub-activities, and transitions are implicitly triggered by the completion of an action or a sub-activity. An activity diagram is associated with a classifier, such as a use case (at a high level or granularity) or an operation (at a more detailed level of granularity). Activity diagrams are made up of a sub-set of: Action state, Call state, Sub-activity state, Decision, Swimlane, and Synch state. An example UML activity diagram is presented in Fig. 2.

The level of abstraction provided by models helps developers and stakeholders visualize different aspects of the system while avoiding the details of implementation. This represents two principles of software engineering, namely the abstraction and separation of concern principles [8]. For any given system, a large number of models can exist and it is important to ensure their overall consistency. Model transformation uses a set of rules called transformation rules,

which accepts one or more models as input and produce one or more target models as output [12].

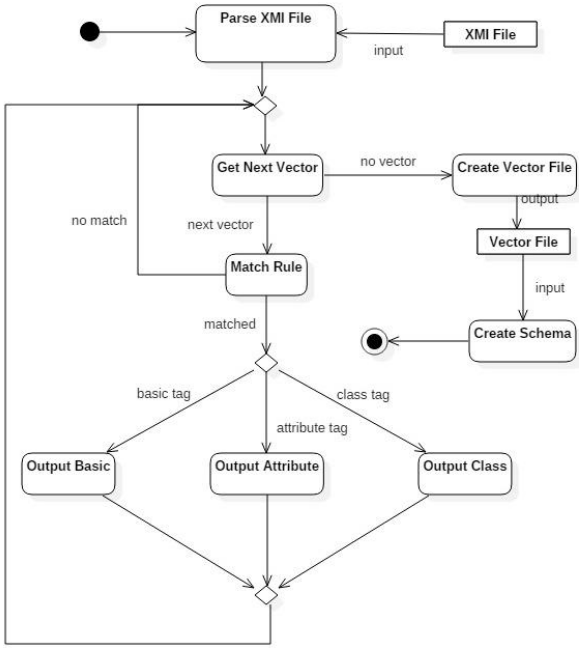

Fig. 2. UML activity diagram example.

C. DO-178C Specification

In order to develop a model-based software development methodology that complies with the DO-178C specification [1] a series of UML models were developed to represent aspects of DO-178C. This approach taken is similar to the approach used in defining the UML specification [11]. Fig. 7 depicts a high-level UML package model of DO-178C, which illustrates that the Software Planning Process defines the Software Development Process and the System Integral Process. The Software Integral Process comprises the Software Certification Process, the Software Safety Quality Assessment Process, the Software Verification Process, and the Software Configuration Management Process. Each package is then further refined to provide the detail content of the package.

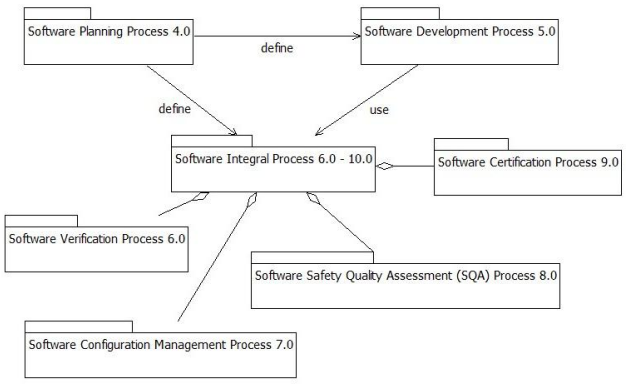

Fig. 3. DO-178C high-level UML package diagram.

Each of the packages of Fig. 3 is decomposed into its components and these components are further decomposed into the low-level constituents of the DO-178C specification. These constituents are made up of processes, data items, and constraints. The goal of this approach is to re-orient the DO178C textual specification into a more understandable hierarchical graphical model that presents an ontological map between the DO178C constituents. Numbers appearing in Fig. 3 denotes the section number in the DO-178C specification [1] for the associated item. Fig. 4 captures a subset of the high-level DO-178C processes that are necessary in order to be compliant. Similar to the Fig. 3, the numbering in Fig. 4 references the relevant section of the DO-168C specification.

III. Research Methodology

In order to develop a model-based software development methodology that complies with the DO-178C specification a series of UML models were developed to represent aspects of DO-178C. This approach taken is similar to the approach used in defining the UML specification [6]. Fig. 3 depicts a high-level UML package model of DO-178C, which illustrates that the Software Planning Process defines the Software Development Process and the System Integral Process. The Software Integral Process comprises the Software Certification Process, the Software Safety Quality Assessment Process, the Software Verification Process, and the Software Configuration Management Process. Each package is refined to the detail content of the package.

In this research, different UML models were developed to represent particular section of DO-178C. In order to convert Software Configuration Management Process from its textual representation to diagrams, work first started from a high-level diagrams which is created using package diagram. Fig. 4 represents this main package diagram, which illustrates the complete Software Configuration Management Process, consisting of the defined by Software Configuration Management Objectives (SCMPO) Section 7.1 of [1], Software Configuration Management Process Activities (CMPA) Section 7.2 of [1], Data Control Categories (DCC) 7.3 of [1]) Software Load Control (SLC) Section 7.4 of [1], and Software Life Cycle Environment Control (SLCEC) Section 7.5 of [1].

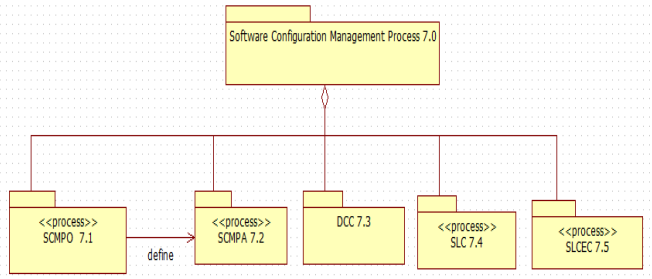

Fig. 4. UML software configuration management process package.

The next Package diagram of Fig. 5 further illustrates the Software Configuration Management Process Activities (Section 7.2 of [1]). Fig. 5 represents this process that is a composition of several activities, namely:

- Configuration Identification(CI) Section 7.2.1 of [1],
- Baselines and Traceability(BT) Section 7.2.2 of [1],
- Problem Reporting, Tracking and Corrective Action PRTCA) Section 7.2.3 of [1],
- Change Control (CC) Section 7.2.4 of [1],
- Change Review (CW) Section 7.2.5 of [1],
- Configuration Status Accounting(CSA) Section 7.2.6 of

[1] and

- Archive, Retrieval and Release(ARR) Section 7.2.7 of [1].

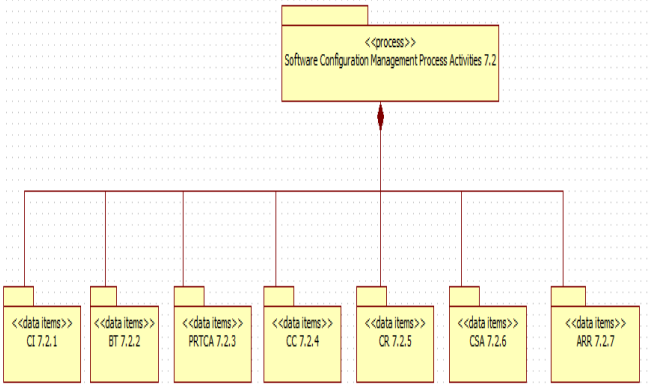

Fig. 5. Software configuration management process activities.

For each package, separate activity diagrams are created. Some activities are associated or connected with other activities, those connected activities are not showed here separately(e.g. Change Control and Change Review are connected with all other activities). Fig. 6 illustrates the activity diagram for Configuration Identification (Section 7.2.1 of [1]). This activity first identifies configuration for each configuration items, based on success, it further proceed for next steps which can be Change Control, Traceability Analysis and Items used or referenced by other software. If all these activities completed successfully, it will generate output, otherwise it will again go to previous stage to perform those activities again.

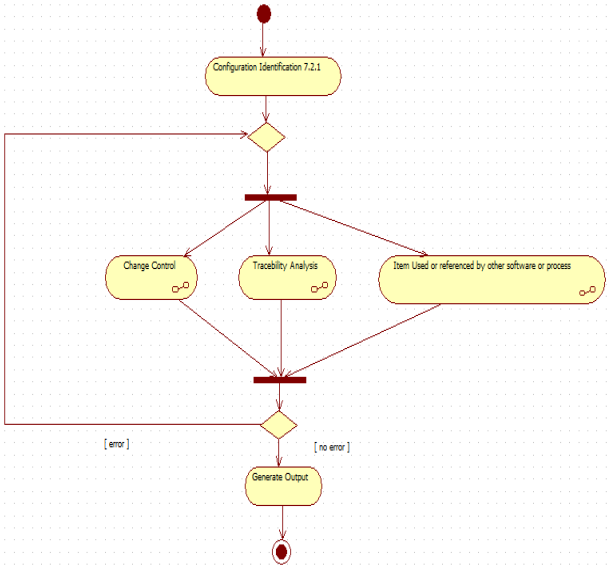

Fig. 6. Configuration identification UML activity diagram.

The next activity diagram is of the activity Baseline and Traceability (Section 7.2.2 of [1]). Fig. 7 illustrates this activity. In this activity first baselines need to be established followed by change control activities which will help to determine derived baselines (such as Intermediate baselines - established as aids in controlling the Software Lifecycle processes, Software Product Baselines - established for Software products and General baselines - established for Configuration Items) and these baselines should be traceable from their output stage.

Applying same methodology, activity diagrams are created for all processes listed in the DO-178 specification. Fig. 8 illustrates the activity diagram for the Problem Reporting, Tracking and Corrective Actions process of the Software Configuration Management Process Activities (Section 7.2 of [1]).

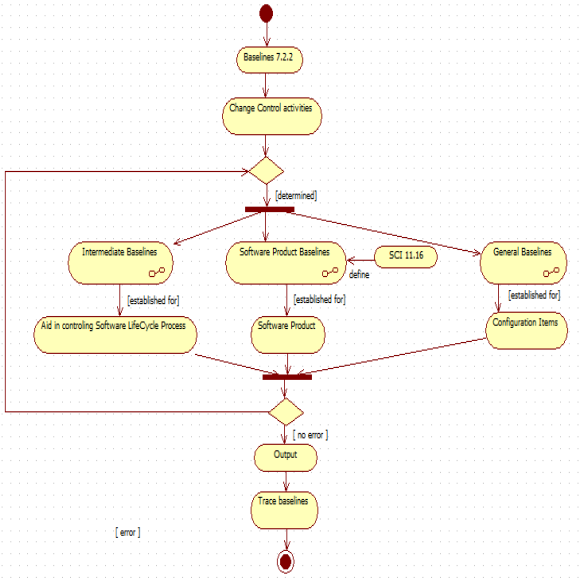

Fig. 7. Activity diagram for baselines and traceability.

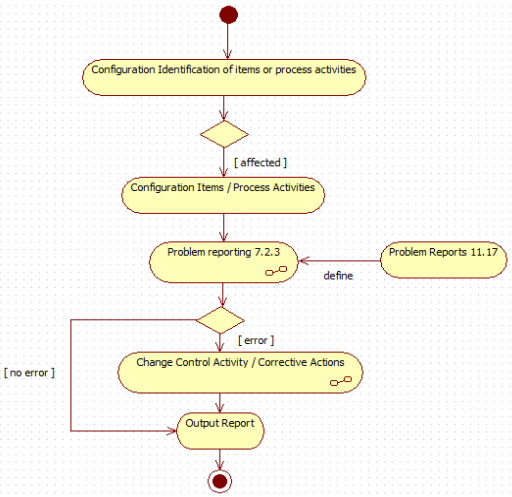

Fig. 8. Activity diagram for problem reporting, tracking and corrective action.

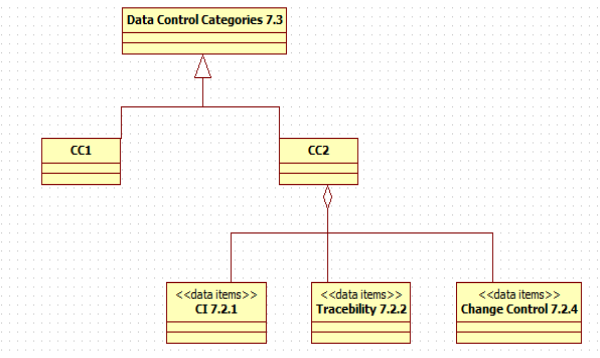

Fig. 9. Class diagram for data control categories.

Along with the UML activity diagrams, the UML class diagrams are created for all other processes which are part of the main configuration process. Fig. 9 is the class diagram for Data Control Categories(Section 7.3 of [1]). It can be

subdivided into two parts, CC1 (Control Category 1) and CC2 (Control Category 2). CC2 activities are subset of CC1 activities [1]. Fig. 9 describes Software Load Control (Section 7.4 of [1]).

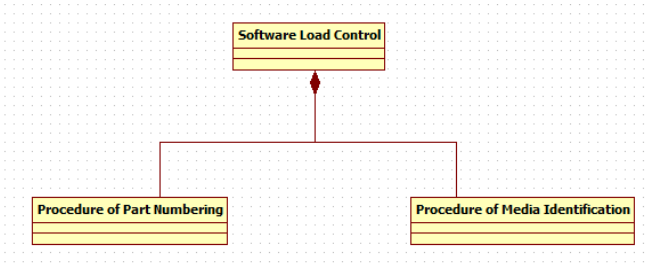

Fig. 9. Class diagram for software load control.

IV. Conclusion

This paper is a description and implementation of UML model transformation in order to represent a particular module of the RTCA DO-178C specification for airborne software systems. Airborne Software systems are complex, they are categorized as safety-critical systems. Textual representation can be difficult to understand and time consuming while gathering requirements during development of such complex airborne software systems. Hence diagrammatic representation can be an aid to the requirements elicitation, documentation, and understanding of the system. In this work, multiple UML diagrams are used in the implementation of a methodology to represent particular module of DO-178C specification.

In future work, this methodology will be used to represent all other modules of the DO-178 specification, which will lead the development strategies to a different level with less time and very comprehensive and clear requirements.

References

[1] RTCA Special Committee 205 (SC-205), "DO-178C — Software considerations in airborne systems and equipment certification, RTCA," Washington DC, DO-178C, Dec. 2011.
[2] C. M. Holloway, "Making the implicit explicit: Towards an assurance case for DO-178C," NASA Langley Research Center, Hampton, Virginia, USA, 2013.
[3] C. Y. Lin, M. Wu, J. A. Bloom, I. J. Cox, and M. Miller, "Rotation, scale, and translation resilient public water marking for images," *IEEE Trans. Image Process.*, vol. 10, no. 5, pp. 767-782, May 2001.
[4] J. Garcia-Molina, A. Moreira, and G. Rossi, "UML and model engineering," *European Journal for the Informatics Professional*, vol. V, no. 2, pp. 3-4, 2004.
[5] B. Selic, "The pragmatics of model-driven development," *Software, IEEE*, vol. 20, no. 5, pp. 19-25, 2003.
[6] A. M. Law, "How to build valid and credible simulation models," presented at Winter Simulation Conference, Berlin, Germany, 2005.
[7] D. C. Schmidt, "Model-driven engineering," *IEEE Computer*, vol. 7, no. 2, pp. 25-31, 2006.
[8] I. Sommerville, *Software Engineering*, 9th ed. Old Tappan, NJ, USA: Addison-Wesley, 2011.
[9] R. France and B. Rumpe, "Model driven development of complex software: A research roadmap," presented at Future of Software Engineering, Minneapolis, USA, 2007.
[10] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, Indianapolis, Indiana, USA: Pressman & Associates, Inc., 2010.
[11] ISO/IEC 19501:2005, Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2.
[12] S. Sendall and W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *Software, IEEE*, vol. 20, no. 5, pp. 42-45, Sept.-Oct. 2003.

Emanuel S. Grant received a B.Sc. from the University of the West Indies, Barbados, a MCS from Florida Atlantic University, USA, and a Ph.D. from Colorado State University, USA, all in computer science. His research interests are in the areas software development methodologies, model driven software development formal specification techniques and domain-specific modeling languages.

He is an associate professor in the Department of Computer Science at the University of North Dakota, USA, since 2008, where he started as an assistant professor in 2002. He is an adjunct professor at the Holy Angel University, Philippines, where he is conducting research in software engineering teaching with collaborators from HELP University College, Malaysia; III-Hyderabad, India; Singapore Management University, Singapore; Rochester Institute of Technology, Baylor University, Montclair State University, and University of North Carolina Wilmington of the USA; and the University of Technology, Jamaica.

Dr. Grant is a member of the Science and Engineering Institute (SCIEI), the Association for Computing Machinery (ACM), Upsilon Pi Epsilon (UPE), and the Institute of Electrical and Electronics Engineers (IEEE), and the International Association of Engineers (IAENG).

Tanaya Datta is pursuing a MS in computer science from University of North Dakota, Grand Forks. She completed her B-Tech in information technology from Techno India, Kolkata. She has industry work experience of almost 4 years in data warehousing, MDM. She has worked in different modeling language, such as UML, SysML, and OCL. Her research interests are modeling of software development, data & web mining.