TOWARDS SOFTWARE DEVELOPMENT WORKFLOW PROCESS FOR SAFETY-CRITICAL SYSTEMS IN AVIONICS

EMANUEL S. GRANT

University of North Dakota, North Dakota, USA
E-mail: grante@aero.und.edu

Abstract - In the field of software development, systems that are classified as safety-critical are considered to be those wherein failure may result in harm to life and possible death. Consequently, development of such systems necessitate the adherence to established standard that specify processes and workflows to ensure the most reliable software systems. In the realms of avionic software systems, there are established standards for developing such systems. This report documents on-going work in defining software engineering processes and workflow that incorporates software development methodology guidelines for safety-critical systems, focusing on avionic systems. The defined processes and workflow will be used in developing course curriculumfor an undergraduate degree in software engineering that are associated with a set of program outcomes.

Keywords - Software Engineering, safety-critical systems, software development methodology, workflow, development processes

I. INTRODUCTION

With the ubiquitous use of software systems in products and services and the growing demands for these systems, it is necessary that reliable strategies be developed for addressing these needs. The inability to deliver software systems that are correct and within cost and time budgets has been referred to as the *software crisis* [1]. The software crisis is greater today than it was decades ago, when it was first debated. Today's software systems are more complexed, being used in new and unique domains, and have greater consequences to the successful or unsuccessful outcomes of their executions. The presence of issues relating to the software crisis in safety-critical system is most undesirable. Safety critical-systems are those in which operational failure may result in harm to or loss of life.

A challenging feature of safety-critical systems is the high level of complexity in their design and implementation. Examples of some of these systems wherein failure has occurred are the THERAC-25 [2], the French Arian-5 rocket inaugural launch [3], and Air France flight 447 (AF447) of June 1, 2009 [4]. These failures overshadow the many successful applications of software systems in safety-critical environments, because of the high cost in property (Ariane-5 development cost US$7 billion, payload US$500 million), lives (Air France 447, 216 passengers and 16 crewmembers).

In order to develop safety-critical systems that are free of software crisis issues, well-defined processes and workflows must be instituted during their development processes. One such body of work is documented in the RTCA DO-178C Software Considerations in Airborne Systems and Equipment Certification [5] for USA development, and the corresponding European EUROCAE ED-12C Software Considerations in Airborne Systems and Equipment Certification. These documents set out a series of objectivities, activities, and data items that are required for the certification of on-board avionic systems.

The work presented in this paper documents a part of the effort of defining a model-driven software development methodology for safety-critical systems that is based on a recently updated specification for avionic system development. The work herein specifically looks at the processes and workflows of the various phase of the software development life cycle (SDLC). The work involves the use of the Unified Modelling Language (UML) [6] in representing the processes and workflows of the RTCA DO-178C Specification. This work effort is also being used in developing curriculum for software engineering courses.

The rest of the paper is as follows: Section II presents a background of the research of the work, with Section III setting out the methodology. Section IV documents the result to date, with the Conclusion and Future Work in Section V.

II. BACKGROUND

Software development life cycle models identifies four phases to the process, namely requirements engineering, design, implementation, and testing. At each of these phases various processes and workflows may be defined, depending on the application domain, and requirements of the system under development. For safety-critical systems, specifically in the avionic filed the DO-178C sets out the requirements for such system and it is up to developers to identify and define the processes to achieve the DO-a78C development objectives. In defining the DO-178C compliant processes and workflows the UML (Unified Modelling Notation) [6] is applied.

2.1 Unified Modeling language (UML)

UML [6] is the de-facto modelling notation for object-oriented software development. UML models

facilitate better communication among customers and developers. Customer and developer get an opportunity to understand the project and its requirements before implementation commence. It also assists software developers to specify the design, implementation, and testing of the system. A feature of the UML is exploited in model-driven development methodologies [7], wherein the models are refined and transformed from the requirements phase of development, through to the detailed design phase. At that detailed level of design, the models may be semi-automatically transformed into code. This provide traceability from requirements to code modules.

The UML model that is used in this work, to specify processes and workflow, is the activity diagram. In UML, activity diagrams capture the sequence of tasks that are needed to complete an activity (process) or specify a workflow. An activity diagram represents tasks and events. It is a variation of a state machine, which is similar to that of a flowchart. Activity diagrams are made up of Action State, Sub Activity State, Call State, Decision, and Sync State. An example of UML activity diagram is presented in Fig.1. The activity diagram process (workflow) commences at the start state (filled circle), and sequences through a series of activities (ovals), decision points (diamonds), and transitions (directional lines) to the terminating state (filled concentric circle). Transition from an activity is executed when all the steps of the activity are completed. Other UML activity diagram semantic concepts implement, concurrency, time constraint, input and output parameters (objects, data, etc.), and repetition (iteration).

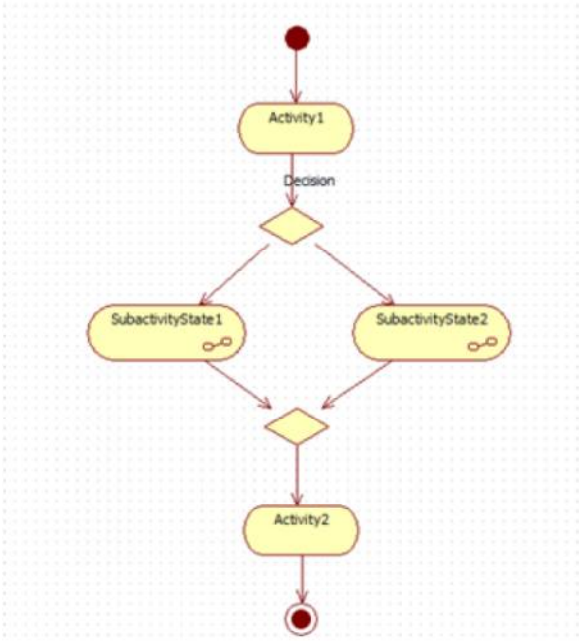


Fig.1. Example UML Activity Diagram

2.2 RTCA DO-178C Specification

The RTCA DO-178C is the de facto standard guideline for avionic software development. The DO-178C describes the requirements for certification of software system for airborne operation, but does not specify how these requirements are to be met. It is up to software developers to implement notations, processes workflows, and methodologies that comply with the stated objectives for certification. This standard encourages the use of formal methods as a means of meeting strict objectives of validation and verification of the system being developed.

In this work, the derivation of SDLC processes and workflows are obtained in the form of activity diagrams. The primary goal for conducting this work is to define a model-driven SDLC methodology for safety-critical system development that is compliant with the RTCA DO-178C specification. This work has been partially applied in the development of the user interface system for an unmanned aerial system that was developed at the University of North Dakota [7].

III. METHODOLOGY

The DO-178Cspecification for avionic software development is made up of twelve sections, two annexes, and two appendices. DO-178C specification has a single statement of purpose, which is "to provide guidance for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that compiles with airworthiness requirements" [5]. In order to develop a set of processes and workflows to implement DO-178C specification a series of UML Package, Class, and Activity diagrams were developed. The Package diagrams were used to capture the high-level descriptions of the specification. Figure 2 illustrates the upper-most package diagram which presents the relationships between the twelve sections of the document.The numerical references in packages and activities of the figures are the section numbers from the DO-178C specification, and have been added for traceability.Each package of Fig.2 is decomposed into a lower-level package diagram, as illustrated in Fig.3, wherein the DO-178C Software Planning Process 4.0 UML Package Mode is presented. The stereotypes<<data item>>are decomposed into UML class diagrams, and the <<process>> stereotypes are decomposed into activity diagrams, which are the focus of this work. Fig.4 is a UML activity diagram that illustrates the workflow of the model-driven methodology process that has been defined for this work. The process model of Fig. 4 is incomplete, with the continuation representing the processes of the implementation (coding) and testing phases of the model-driven SDLC processes. The instance of the methodology of Fig. 4 uses the UML models as the required data items, but another instance may make

use of other modeling notation and language. Similarly, the verification process (Verify High-Level Design 6.3, Verify Low-Level Design 6.3) will be instantiated from a variety of formal specification techniques and notations, depending on the level of rigor required for the application domain.

Fig. 5 is an elaboration of the "Conduct Software Requirement 5.0" of Fig. 4. Herein, this e workflow does not specify the particular types of representation of the input or output of the defined activities; it is the responsibility of the developers/organizations to make that determination. Each task in each activity diagram is decomposed to a detailed level that negates the use of the initial textual document. The intent is that users of the DO-178C documentation then relies on an ontologically related set of models for implementing the processes of the DO-178C specification as a set of graphical (UML) workflow models.

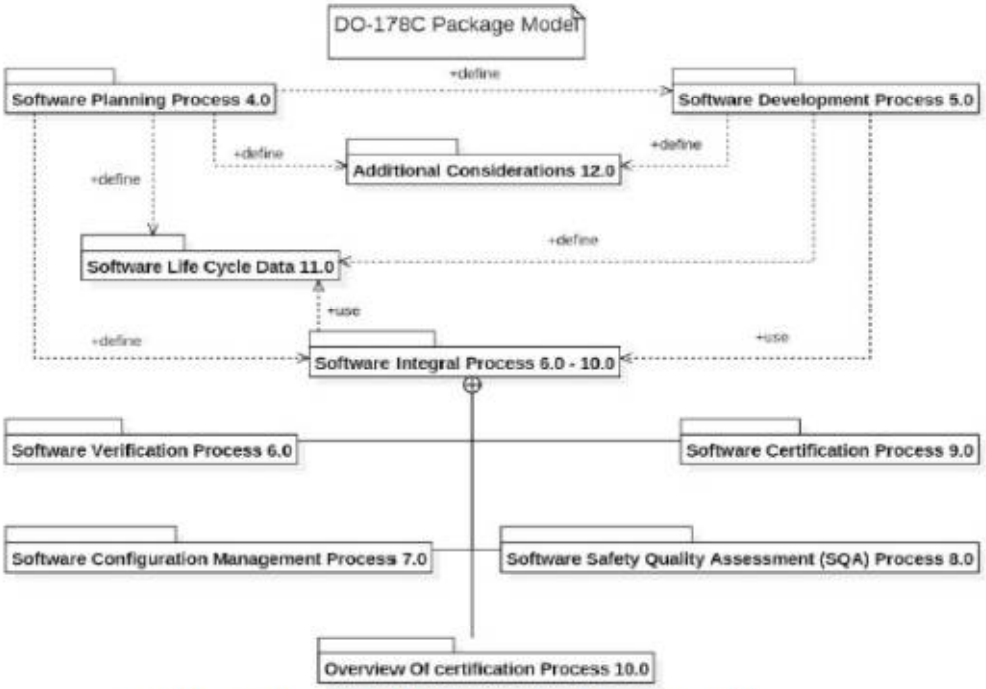


Figure 2. DO-178C High-Level Package Diagram

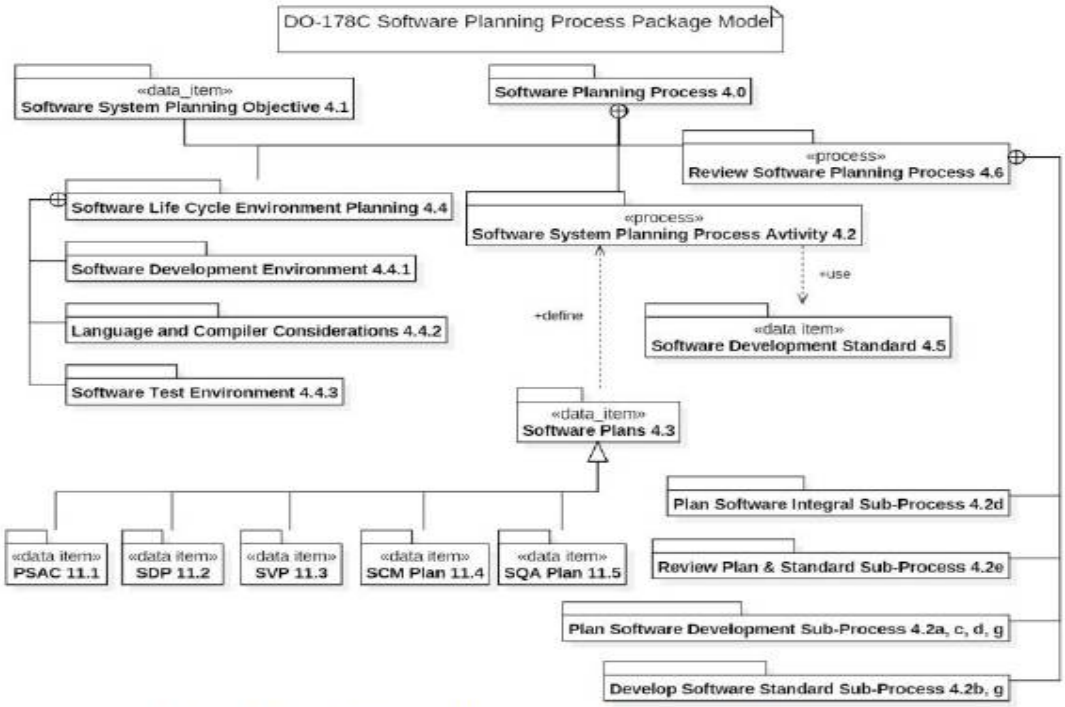


Figure 3. DO-178C Software Planning Process 4.0 UML Package Model

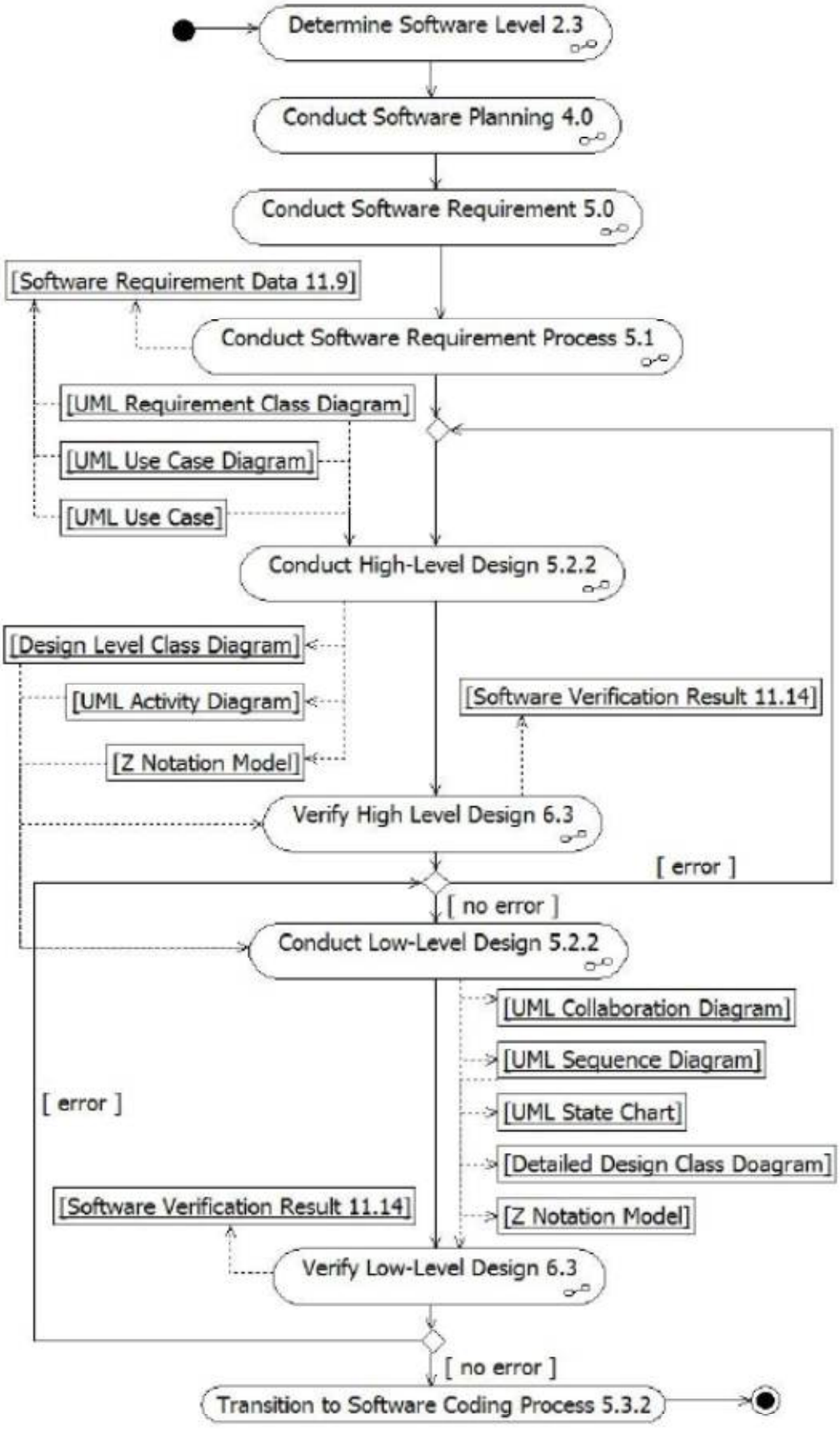


Figure 4. UML Activity Diagram of MDM Workflow

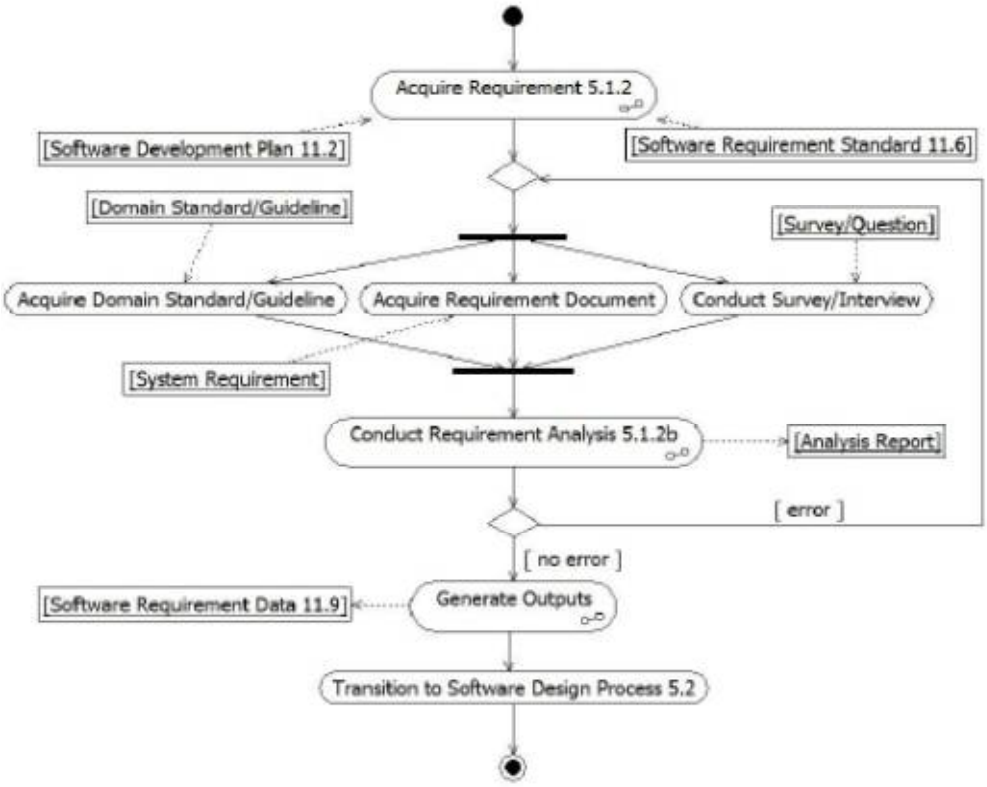


Figure 5. UML Activity Diagram of MDM Requirements Analysis

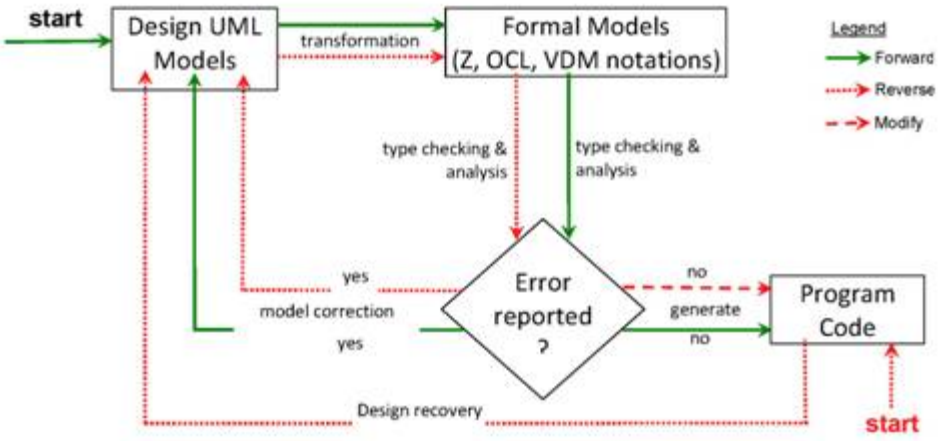


Figure 6. Reverse and Forward Engineering with Formalism

IV. RESULT

The University of North Dakota (UND) – UAS Risk Mitigation Project was awarded a contract to develop a proof-of-concept air truth system, which monitors the operation of UAVs in the US National Airspace (NAS). The project began with minimal requirements; however, the timeframe for delivery was very uncompromising. This resulted in the rapid development of a prototype to assist in exploring and developing additional requirements. A feature of prototypes is that they are often poorly documented. To resolve this, concurrent definition and documentation of system requirements were performed as the prototype evolved. This was enhanced with the design of graphical software models. In model-driven engineering, the purposes and uses of graphical software models are multifaceted. They represent the structural design of the system, and the flow of data, and communication and control between the various systems and subsystems. Its use is not only suited for astute stakeholders but also non-technical stakeholders such as customers – to convey how their requirements are being met.

In order to advance the initial work done on the UAS project, with respect to the model-driven software development methodology, the DO-178C process, as presented in the workflow of Fig. 4 is now being applied to further system development. This effort involves re-engineering some aspects of the prior work. A process model was developed to implement this re-engineering and accompanying forward-engineering activities. This process model is presented in Fig.6, which also illustrates the use of formal specification techniques for validating the re- and forward-engineering activities. The "Design UML Models" activity of Fig.6 is synonymous to the "Conduct Low-Level Design 5.2.2" of Fig. 4, and the "Formal Models" activity of Fig. 6 is synonymous to the "Verify Low-Level Design 6.6" activity of Fig.4.
This work resulted in the identification of multiple errors in the initial design models by means of the formal specification technique (FST) [8] that was applied at the verification phase of the model-driven software development methodology employed on the project. In a safety-critical environment, these software design errors could result in catastrophic failure of the operating system. On this project the Z formal specification notation [9] was employed, because of the developers' familiarity with the notation, but other FST could have been employed.

CONCLUSION

This report documents the work in defining a set of workflow process models for model-driven software development that is based on the RTCA DO-178C Software Consideration in Airborne Systems and Equipment Certification specification. This work is primarily focused on the development of safety-critical systems that are now ubiquitous in daily life. Such systems appear in many fields, such as the medical, transportation, and home-care, products and services that are used in private. Professional, individual, and group spheres worldwide. The reliability of these systems is paramount to not only the users of these systems, but also to the future application of software systems in new domains. The software crisis continues to be the motivation for many research projects that seek to make advances in software development. The increasing need for more reliable software systems will continue to grow and the need for more reliable software systems keeps pace.

This research effort successfully developed a set of workflow models of DO-178C processes that were integrated into a model-driven software development methodology, specifically for airborne systems, but applicable to the broader domain of safety-critical systems. Aspects of this methodology were applied in reverse engineering and forward engineering the display system for the UND UAS Risk Mitigation Project. The outcome of this effort resulted in the identification of errors in the implementation of the code which would have resulted in undesirable operational performance of the system.
This research effort is ongoing, with the methodology being applied in another domain that involves the problem of optimization of the allocation of parking sloths to vehicles with restrictions on the size and configuration of the vehicles, the available services at each parking slots, and the location of the parking slots. Another area of further research is the automation of some of the defined workflows, in order to make the development methodology more reliable.

REFERENCES

[1] H. Weber. The Software Factory Challenge (ed.). *Fraunhofer Institute for Software Engineering and Systems Engineering (ISST)*, IOS Press, Amsterdam, Netherlands, 1997.

[2] N. G. Leveson, C. Turner, "An Investigation of the Therac-25 Accidents", IEEE Computer, IEEE Computer Society, Vol. 26, No. 7, July 1993, pp. 18-41.

[3] J. L. Lions, "ARIANE 5, Flight 501 Failure", Report by the Inquiry Board, Paris France, July 1996.

[4] Bureau d'Enquêtes et d'Analyses, "Final Report On the accident on 1st June 2009 to the Airbus A330-203 registered F-GZCP operated by Air France flight AF 447 Rio de Janeiro – Paris, Paris France, July 2012.

[5] RTCA. "Software Considerations in Airborne Systems and Equipment Certification." DO-178C. 2011.

[6] ISO/IEC 19501, Information Technology - Open Distributed Processing,: Unified Modeling Language (UML) Version 1.4.2 (2005).

[7] E. S. Grant, V. K. Jackson, S. A. Chachar. "Towards a Formal Approach to Validating and Verifying Functional Design for Complex Safety Critical Systems". Proc. 2nd Annual International Conference on Software Engineering & Applications (SEA 2011), Hotel Fort Canning, Singapore, Singapore. April 2011.

[8] France, R.B., Bruel, J., Larrondo-Petrie, M.M.: An Integrated Object-Oriented and Formal Modeling Environment. In Proceedings of JOOP. 25--34. (1997).

[9] ISO/IEC 13568, Information Technology: Z Formal Specification Notation - Syntax, Type System and Semantics. 1st. ed. ISO/IEC 2002).

★★★