Presented at the 1st International Workshop on Aerospace Engineering (AEROSE 07)
Minneapolis, MN, May 21-22, 2007

Software Assurance for the Emerging Discipline of Model-Based Design

Jane M. C. Oh & Martin S. Feather
Jet Propulsion Laboratory, California Institute of Technology
jane.c.oh@jpl.nasa.gov & martin.s.feather@jpl.nasa.gov

Abstract

This paper addresses the novel challenges and opportunities that model-based design poses to software assurance. The novelty stems from the fact that the models are deliberately made to be machine manipulable, leading to a shift in emphasis and scope of human activities vs. machine-conducted activities. As a result, it requires a more closely coupled set of processes and techniques to achieve the desired level of product and process assurance. Opportunities arise from the machine manipulable nature of the artifacts, making them amenable to analytic methods and automated tools. The significant advantages of analytic methods over conventional assurance practices stem from their superior thoroughness, repeatability and applicability, especially when applied to models. In response, we are working towards an integrated software assurance process that combines advantages of top-down and bottom-up approaches to ensure the quality of the model and the quality of the software implementation.

1. Introduction

NASA and other space agencies rely on ever larger, more complex software systems to do more challenging space missions. This trend raises the stakes and increases the risks, and so drives the need for breakthrough solutions to reduce risks. For example, for the Mars Exploration Rovers, entry, descent, and landing were software controlled, and relatively sophisticated (for example, they included on-board algorithms to analyze data from the downward-viewing imager, ranger and inertial measurement unit to determine horizontal velocity so as to direct lateral thruster firings during descent [1]). Tom Gavin, the associate director of JPL, said "…that is a software driven system, and a lot of the project risk is in how robust that software is…" [2].

There is a critical need for assurance methods that will be applicable to the verification and validation of models. For example, Mars Polar Lander is thought to have failed when the software turned off its engines too soon. The flaw in the software-system design was introduced when the software requirements were derived from the system requirements, and was not caught in subsequent review and testing [3]. Had there been a system model, this flaw would have been possibly detected - the (retrospective) application of a model-based testing approach demonstrating this is described in [4].

This paper addresses the novel challenges and opportunities that model-based design poses to software assurance, defined as "the planned and systematic set of activities that ensure that software life cycle processes and products conform to requirements, standards, and procedures." [IEEE 610.12, IEEE Standard Glossary of Software Engineering Terminology] For NASA this includes the disciplines of Software Quality (functions of Software Quality Engineering, Software Quality Assurance, Software Quality Control), Software Safety, Software Reliability, Software Verification and Validation, and IV&V. [NASA-STD-8739.8]. The novelty stems from the fact that the models are deliberately made to be machine manipulable, leading to a shift in emphasis and scope of human activities vs. machine-conducted activities. As a result, it requires a more closely coupled set of processes and techniques to achieve the desired level of product and process assurance. Opportunities arise from the machine manipulable nature of the artifacts, making them amenable to "analytic verification" and automated tool. NASA (National Aeronautics and Space Administration) defines "analytic verification" as "a set of techniques and tools based on mathematical modeling and formal logic that are used to specify and verify requirements and designs for computer systems and software" [5]. Section 2 considers software assurance, Section 3 analytic verification.

2. Software Assurance Process

The objective of the software assurance process is to improve the quality of mission-critical software, and the productivity of the software development process, by detecting and removing errors and defects early in the life cycle. A list of traditional software assurance activities are shown in Table 1, along with the artifacts to be checked throughout the life cycle (i.e., Requirement, Design, Code, Test/Debug, and Operation) in practice at JPL.

Software Assurance Activities	Software Assurance Artifacts
Software Development Process and Monitoring	Software Development Plan, Project Management Plan, Software Quality Assurance Plan, Software Configuration Management Plan, Software Risk Management Plan
Subcontractor Controls	Requests for Proposals, Statements-Of-Work, Procurement Requisitions, Pre-award Vendor Software Surveys
Software Requirements Analysis	Level 1, 1, 2, 3, 4, and 5 Requirements Documentation, Functional Specification, Interface Control Documentation, Bi-Directional Traceability, Operational Specs
Software Reviews	Code Walkthrough, Unit Test Walkthrough, Test Case Generation, Code Run-time Monitoring, Code Static Analysis
Formal Reviews	Conceptual Design Reviews, Preliminary Design Reviews, Architecture Reviews, Detailed Design Reviews, Test Readiness Reviews, Assembly and Operation Readiness Reviews
Management Reviews	Fever Chart, Risk Assessment, Project Insight and Oversight
Peer Reviews	Software Reuses, Inheritance Documentation, Test Plans and Procedures, Test Results Verification,
Software Delivery Reviews	Software Review Certification Records, Release Description Document, Acceptance Test Plan/Procedures/Report, End Item Data Packages
Software Safety Hazard Analysis	Safety-critical Components Assessment, Safety/Hazard Analysis Report
Software FMECA And Fault Tree Analysis	Software Failure Modes Effect and Criticality Analysis, Software Fault Tree Analysis
Software Configuration Management	Engineering Change Requests, Change Control Report, Configuration Management Report, Build List
Software Problem/Failure Reporting	Problem/Failure Report, Root Cause Analysis Report, Action Items Report, Test Traceability Matrix
Verification and Validation	Unit Test, Integration Test, System Test, User Acceptance Test, Subsystem Validation, System Validation, Hardware Review Certification Records

Table 1. Software Assurance Activities

JPL tailored IBMer Fagan's original process [6] of software inspections to conform to its software development environment in 1987 [7]. Inspections are widely regarded as one of the most cost-effective techniques in the assurance arsenal. We believe inspections have the potential to be equally effective for model-based design and thus we apply current software assurance techniques such as software inspections [8] and peer reviews [9] to those models generated for flight software projects. A strategy is to combine advantages of both traditional assurance activities (e.g., peer reviews, milestone reviews, etc.) and analytic methods to ensure the quality of the model (e.g., the measures of correctness, completeness, and conformance) and the quality of the software implementation (e.g. the measures of defects, failures, and efficiency). This integrated software assurance process is depicted in Figure 1.

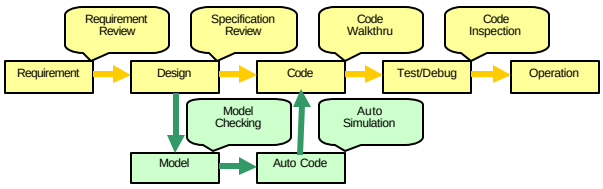

Figure 1. Software Assurance Process

Those traditional assurance activities listed in Table 1 will be changed in the move to dealing with model based artifacts [10]. There will be fewer *code* walkthroughs, and instead more *model* walkthroughs and analyses using several emerging techniques such as runtime monitoring (during simulation of the model), static analysis (adapted to models), and model-checking (by using SPIN). As a result, it will be necessary to ensure the software community is ready for model-based design and the increasingly challenging missions' software systems developed with this paradigm. In this context, model-based design offers both a challenge and an opportunity to assurance: the challenge stems from the explosion in

detail and complexity, with which current software assurance practices will be hard-pressed to keep pace; the opportunity stems from the increased suitability to analytic methods of the artifacts (models and simulations) that pervade the model-based design paradigm.

Within JPL and NASA, the emergence of model-based design will reformulate the design process, and the nature of software assurance must change accordingly. Those assurance activities that will be changed in the move to model-based are shown in Table 2. Using model-based design combined with auto-coding, there will be less time required to check the 25 artifacts (listed in Table 2) out of the 60 software assurance artifacts (shown in Table 1).

Software Assurance Activities	Software Assurance Artifacts
Software Development Process and Monitoring	Software Quality Assurance Plan, Software Configuration Management Plan, Software Risk Management Plan
Subcontractor Controls	Statements-Of-Work
Software Requirements Analysis	Bi-Directional Traceability
Software Reviews	Code Walkthrough, Unit Test Walkthrough, Test Case Generation, Code Run-time Monitoring, Code Static Analysis
Formal Reviews	Detailed Design Reviews
Management Reviews	Risk Assessment
Peer Reviews	Software Reuses, Test Results Verification
Software Delivery Reviews	Software Review Certification Records, End Item Data Packages
Software Safety Hazard Analysis	Safety-critical Components Assessment
Software FMECA And Fault Tree Analysis	Software Fault Tree Analysis
Software Configuration Management	Configuration Management Report, Build List
Software Problem/Failure Reporting	Action Items Report, Test Traceability Matrix
Verification and Validation	Unit Test, Integration Test, System Test, User Acceptance Test

Table 2. Changed Activities

3. Analytical Method

The significant advantages of analytic methods over conventional assurance practices stem from their superior thoroughness, repeatability and applicability, especially for models (e.g., [11, 12]). However, despite considerable JPL and NASA effort invested in them [13], and successful pilot applications on NASA spacecraft software (e.g., [14, 15, 16]), the uptake of formal methods in JPL's software processes has been miniscule.

We believe the primary impediment has been the effort needed to construct the formal models on which to perform the analyses. In conventional software development, these models take time to develop, and require personnel skilled in both the system being scrutinized and the mathematical underpinnings of the analysis methodology. The result is an approach that is costly, hard to find people to perform, and always struggles to remain up-to-date with the rapidly changing nature of flight project developments. Even for so-called "lightweight formal methods" reported in [16], where it is argued that the "benefit more than outweighs the extra cost of maintaining several representations", infusing these into mainstream practice is challenging. Model-based design circumvents this impediment because the "models" of model-based design can serve as the formal models for analysis! Benefits will be greater yet if the formal models are easily reusable.

This research aims to achieve early and more thorough detection (and resolution) of problems. By coupling analytic methods with automated tools, we aim to increase the utility of those techniques: an increased fraction of software related defects may be detected earlier in the development process (i.e., well in advance of testing). As illustration, two examples of UML statecharts are provided in Figures 2 and 3. The former has an unreachable state and the latter has an undesired state.

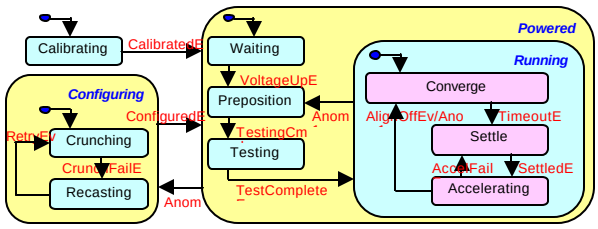

Figure 2. Check the Unreachable States

The state-machine shown in Figure 2 does not satisfy the "reachability" property. The semantics of this UML Statechart is such that when the Anom event is generated, the transition from the Running state to the Preposition state takes preference over the transition from the Powered state to the Configuring state.

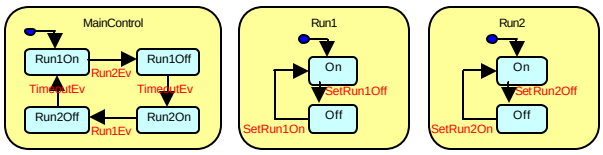

Figure 3. Check the Undesired States

The UML statechart shown in Figure 3 consists of three concurrently running state-machines (MainControl, Run1, and Run2). The MainControl state-machine is responsible for directing control by sending internal command events to the Run1 state-machine and the Run2 state-machine. There are 3 external events (e.g., Run1Ev, Run2Ev, and TimeoutEv). An undesired state was found (Run1 = On && Run2 = On). In the correct model, the MainControl state-machine must send out a command after it receives an acknowledgement that the other state-machine has transitioned to Off (Run1OffAck and Run2OffAck).

When formal models exist, we need to reconsider the purposes of human review/inspection. For simple syntactic properties (e.g., detection of statecharts lacking an initial transition), we may instead utilize simple automation check the statecharts. Depending on the modeling tool(s) used, such checking may already be built in to the tool used to help in the construction of models, or if not, can easily be provided in the form of simple scripts that check the output of such tools.

Human review/inspection of models is well suited to checking *some* of the validity of the models. But, for properties that stem from the behaviors implied by models (e.g., unreachability of states, analogous to dead code in classical software), human inspection is not particularly well suited. Likewise, when behaviors involve multiple, cooperating statecharts, human inspection is also challenged – in both of these cases model checking is well known to be an appropriate technique.

4. Conclusions

Two factors drive the need for this research. First, within JPL and NASA, the emergence of model-based design will reformulate the design process, and the nature of software assurance must change accordingly. Second, the scale and complexity of mission-critical software is increasing as NASA's missions become more challenging.

The results of this research will have applicability that spans NASA's mission suite: the stringent reliability needs of NASA's human exploration activities (Crew Exploration Vehicle, CEV and Constellation) of ESMD (Exploration System Mission Directorate) relate to model based approaches (e.g., Simulation Based Acquisition), and thus dovetail with this research focus. The use of model based design as a means to accelerate the design and development process is expected to continue in JPL's mission set.

5. Acknowledgements

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration and funded through the internal Research and Technology Development program.

6. References

[1] Y. Cheng, J. Goguen, A. Johnson, C. Leger, L. Matthies, M. San Martin and R. Willson, "The Mars Exploration Rovers Descent Image Motion Estimation System", *IEEE Intelligent Systems*, 19(3), May-June 2004, pp. 13-21.

[2] P. Regan and S. Hamilton, "NASA's Mission Reliable", *Computer*, IEEE, Jan 2004, pp. 59-68.

[3] JPL, Mars Polar Lander/Deep Space 2 Loss – JPL Special Review Board Report, D-18709.

[4] M. Blackburn, R. Busser, Nauman, A. Knickerbocker, and R. Kasunder, "Mars Polar Lander Fault Identification Using Model-based Testing", *26th NASA Goddard Software Engineering Workshop*, 2002, pp. 128-135.

[5] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems, Volume I: Planning and Technology Insertion", *NASA/TP-98-208193*, 1998.

[6] M.E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", *IBM Systems Journal*, 15(3), 1976, pp. 182-211.

[7] J.C. Kelly, J.S. Sherif, and J. Hops, “An Analysis of Defect Densities Found During Software Inspections”, *Journal of Systems and Software.* 17(2), February 1992, pp. 150-166.

[8] R.A. Radice, *High Quality, Low Cost Software Inspections*, Paradoxicon Publishing, 2002.

[9] K. Wiegers, *Peer Reviews in Software: A Practical Guide*, Addison-Wesley, 2002.

[10] M.S. Feather, L.M. Fesq, M.D. Ingham, and S.L. Klein, “Planning for V&V of the Mars Science Laboratory Rover Software”, *IEEE Aerospace Conference*, March 2004.

[11] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen, and P. McKenzie, *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer Verlag, 2001.

[12] G. Holzmann, *The SPIN Model Checker, Primer and Reference Manual*, Addison Wesley, 2003.

[13] J.C. Kelly, "Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems, Volume **I**: A Practitioner's Companion", *NASA-GB-001-97*, 1997.

[14] D. Dvorak, “Challenging Encapsulation in the Design of High-Risk Control Systems”, *Conf. Object-Oriented Programming, Systems, Languages, and Applications*, Seattle, 2002.

[15] F. Schneider, S. Easterbrook, J. Callahan, G. Holzmann, W. Reinholtz, A. Ko, and M. Shahabuddin, “A Process for Verifying and Validating Requirements for Fault Tolerant Systems Using Model Checking”, *Transactions on Software Engineering*, February 1999.

[16] S. Easterbrook, R. Lutz, R. Covington, R. J. Kelly, Y. Ampo, D. Hamilton, “Experiences using lightweight formal methods for requirements modeling”, *IEEE Transactions on Software Engineering*, 24(1), January 1998, pp. 4-14.