Using Testbeds to Accelerate Technology Maturity and Transition: The SCRover Experience

Barry Boehm[§], Jesal Bhuta[§], David Garlan[†], Eric Gradman[§], LiGuo Huang[§], Alexander Lam[§], Ray Madachy[§], Nenad Medvidovic[§], Kenneth Meyer[*], Steven Meyers[§], Gustavo Perez[§], Kirk Reinholtz[*], Roshanak Roshandel[§], Nicolas Rouquette[*]

[§]*Computer Science Department*
University of Southern California
Los Angeles, CA 90089
USA
{boehm,jesal,gradman,liguohua, alexankl,madachy,neon,stevenme, gup,roshande}@usc.edu

[*]*Jet Propulsion Laboratory*
4800 Oak Grove Drive
Pasadena, CA 91109
USA
{kenny.meyer,kirk.reinholtz, nicolas.f.rouquette}@jpl.nasa.gov

[†]*School of Computer Science*
Carnegie Mellon University
Pittsburgh, PA 15213
USA
garlan@cs.cmu.edu

Abstract

This paper is an experience report on a first attempt to develop and apply a new form of software: a full-service testbed designed to evaluate alternative software dependability technologies, and to accelerate their maturation and transition into project use. The SCRover testbed includes not only the specifications, code, and hardware of a public safety robot, but also the package of instrumentation, scenario drivers, seeded defects, experimentation guidelines, and comparative effort and defect data needed to facilitate technology evaluation experiments.

The SCRover testbed's initial operational capability has been recently applied to evaluate two architecture definition languages (ADLs) and toolsets, Mae and AcmeStudio. The testbed evaluation showed (1) that the ADL-based toolsets were complementary and cost-effective to apply to mission-critical systems; (2) that the testbed was cost-effective to use by researchers; and (3) that collaboration in testbed use by researchers and the Jet Propulsion Laboratory (JPL) project users resulted in actions to accelerate technology maturity and transition into project use. The evaluation also identified a number of lessons learned for improving the SCRover testbed, and for development and application of future technology evaluation testbeds.

1. Introduction

The NASA High Dependability Computing Program (HDCP) is a major investment in new research and technology to improve the dependability of NASA mission software, and of software-intensive systems in general. The research program addresses new capabilities in such areas as lightweight formal methods, model checking, architecture analysis, human factors, code analysis, and testing The HDCP strategy for accelerating the usual 18-year transition time for software engineering technology [20] employs common-use technology evaluation testbeds representative of NASA mission software.

One such testbed is the SCRover robot testbed based on NASA JPL's Mission Data System (MDS). This paper summarizes the experiences to date of the collaborative effort between the University of Southern California (USC), the Jet Propulsion Laboratory (JPL), and Carnegie Mellon University (CMU) in developing and exercising this testbed. Section 2 describes the HDCP testbed approach, the MDS technology, and their relationships to the SCRover testbed. Section 3 describes the SCRover testbed and its constituent elements. Section 4 summarizes the approach and results of the initial technology evaluation, using the Mae architecture definition language (ADL) and the Mae set of analysis and execution monitoring tools, and documents some complementary results from a partial evaluation of CMU's AcmeStudio technology. Section 5 summarizes the implications for the use of Mae and AcmeStudio on JPL MDS software applications, showing how the testbed results are beginning to accelerate technology maturity and transition[1]. Section 6 presents the conclusions,

[1] *Note to reviewers:* A related paper [27], also submitted to this conference, provides a complementary view of this effort. While that paper focuses on the use of ADLs for modeling and analyzing specifications like SCRover's, this paper focuses on the use of testbeds like SCRover for speeding technology evaluation, maturation, and transition into project use.

including lessons learned from the testbed development and technology evaluations, and future plans for improving and experimenting with the testbed.

2. The NASA HDC testbed approach and relations to the SCRover testbed

2.1. The NASA HDC testbed approach

Risk considerations and economic considerations make it generally impractical to experimentally apply immature research artifacts to operational mission software. In response, the HDCP testbed approach has defined a series of four testbed stages that enable a highly cost-effective progression through increasingly challenging and realistic experimental applications of HDC technology. The four stages are:

1. Experimental scenarios roughly representative of NASA missions, developed or adapted by researchers to fit their technology capabilities.
2. Tailorable testbed suites with hardware, software, and specifications representative of NASA missions; provided to researchers along with representative mission scenarios, instrumentation, seeded defects, installation and experimentation guidelines, and baseline data for comparative evaluation of a technology's cost-effectiveness;
3. Application of more mature technologies to NASA operational mission software in a NASA simulation environment;
4. Application of matured technologies to NASA operational mission software in a NASA operational environment.

The use of these testbeds and complementary approaches such as Technology Maturation Teams enables NASA and HDCP to accelerate the maturation of an emerging HDC technology, using criteria such as the NASA Technology Readiness Levels [16,24]. Experimentation at each stage can be done with relatively low marginal expenditure of effort, based on the feedback received in earlier stages. Involving NASA mission personnel in defining representative mission scenarios and evaluating experimental results on a technology's cost-effectiveness will also enable mission personnel to adopt new technology earlier. Researchers will also be able to concurrently pipeline more advanced and more mature versions of their technology through the testbed stages. A major hypothesis to be tested by the HDCP is that the staged testbed approach will be able to compress the traditional 18 year interval from concept emergence to regular mission usage [20] to 5-7 years.

2.2. The JPL Mission Data System technology

In 1998, the NASA Jet Propulsion Laboratory (JPL) initiated a project called the Mission Data System (MDS) to develop a core system engineering methodology and software toolset for the next generation of deep space missions. Currently MDS technology is baselined to support system engineering and software development for the Mars Science Laboratory Project, scheduled to launch in the Fall of 2009.

The MDS goal has been to develop a set of closely matched tools and techniques to reduce development and debugging cost, promote reusability and increase reliability throughout a project's lifecycle. The principal MDS products include a system engineering methodology called the State Analysis Process, a software framework, a goal-based operational methodology, and a cost estimation model based on COCOMO II [2]. Each of these products is briefly described below:

2.2.1. State Analysis Engineering process. MDS provides a collaborative engineering methodology and tools for systems engineers to capture requirements in terms of familiar concepts: states, commands, measurements, estimators, controllers, and hardware devices. Requirements are captured in a database that can be checked for validity and completeness. The resulting requirements are organized into a state-oriented model of the system's behavior, which maps directly into the software framework (discussed below), eliminating errors in translation and reducing cycle time.

2.2.2. Framework. The MDS Framework consists of over 35 reusable packages for common functionality such as state-oriented control, event logging, time services, data management, visualization, units of measurement, state variables, and an interface to real or simulated hardware called the hardware proxy. The entire set is organized into a modular architecture supportive of state-oriented real-time control systems.

2.2.3. Operational tools. Operators of MDS-based systems specify activities in terms of "what" rather than "how," or, in MDS parlance, in goals rather than commands. Goal-driven operation provides a level of control that can vary from purely time-scripted to fully-autonomous operations. A goal is simply a constraint on the value of a state variable over a time interval. Goals are assembled into goal networks that prescribe timing and prerequisites (or preconditions) for goals. Goal networks are scripted in a Goal Elaboration Language (GEL) that provides an unambiguous expression of operational intent.

2.2.4. Cost estimation model. MDS defines a cost model that helps customers reliably estimate adaptation cost and schedule. The models are based on the COCOMO II cost modeling methodology and confirmed with objective metrics captured by ongoing MDS adaptation efforts. The current cost model parameter values are preliminary. However, as more projects develop MDS adaptations, these parameter values will be refined.

2.3. Relations to the SCRover testbed

Originally, the HDCP testbed approach involved provision of subsets of actual NASA mission software and specifications to researchers for experimental application of their technologies at testbed stages 1 and 2. But the heightened national concern with mission security occasioned by the events of September 11, 2001 caused most U.S. government mission applications software to be placed under the International Traffic in Arms Regulation (ITAR) distribution limitations, making them available only to U.S. citizens. As much of HDC research is being performed by mixed teams of U.S. citizens and foreign nationals, this caused a rethinking of HDCP testbed stages 1 and 2.

The resulting testbed strategy currently being pursued by HDCP is to create Stage 2 testbed suites representative of NASA missions that are not subject to ITAR constraints, and that can be tailored by mixed-nationality HDC research teams to provide both Stage 1 and Stage 2 testbed capabilities. The overall set of success criteria for these testbed suites includes having:

- Capabilities and usage scenarios representative of NASA missions;
- Application characteristics not subject to ITAR constraints;
- Full testbed support, including tailorable mission scenarios, instrumentation, seeded defects, installation and experimentation guidelines, and baseline data for comparative evaluations;
- Ease of distribution and use;
- Cost-effective development, operations, and maintenance.

The SCRover testbed is a collaborative effort by USC and JPL to develop a campus public safety robot performing mission scenarios representative of JPL planetary rover missions and using the JPL MDS Framework. The SCRover software is being developed by U.S. citizen graduate students with access to the JPL MDS internals, but it is being developed with open MDS interfaces not subject to ITAR constraints. The next section describes the elements of the SCRover testbed, and our experience to date in satisfying the testbed suite success criteria above.

3. Elements of the SCRover testbed

3.1. SCRover operational concept

The SCRover testbed provides an experimental framework that allows researchers to evaluate the efficiency of their HDC technology on a NASA-like project. The testbed contains software, supporting information such as documentation, metrics, instrumentation, seeded defects, and guidelines, a robotic platform (both real and simulated), and a development environment. To use the testbed, researchers start by applying their technology to the SCRover specification and code. Then, based on the evaluation criteria defined by them, appropriate instrumentation and seeded defects are applied to the project artifacts. These features will help gather the necessary data used to evaluate the performance of the technology.

The next step is to define the appropriate operational scenarios under which the technology will be evaluated. These operational scenarios are represented by goal networks that are transmitted to the system in the form of GEL files. The code is then executed.

After the execution of the system, the researchers use the data provided by the instrumentation to determine the percentage of seeded and unseeded defects of each type that were found. This enables an analysis of how well the technology performs in detecting, avoiding, or compensating for various classes of seeded and previously undiscovered defects, in comparison to alternative technologies. The data and the analysis are then stored in an experience base to be accessed by project managers interested in technology to increase the dependability of their delivered systems.

3.2. SCRover testbed architecture, specification, and code

Development within the MDS Framework to operate our robotic platform (Pioneer 2-AT) has focused on the Hardware Proxy, State Knowledge, State Determination, and State Control components of the framework's four component cycle as expressed on the left side of Figure 1.

In the following sections, we describe our efforts to enable MDS to communicate with the robot (Hardware Proxy) and our implementation of three top-level components (State Knowledge, State Control, and State Determination).

3.2.1. Behaviors. We have successfully implemented two separate high-level behaviors for the SCRover as a proof-of-concept, and to provide a baseline for our ongoing development of more complex behaviors. In Increment 1, we duplicated the functionality of JPL's MRE4 (Mars

Rover Example 4). This demonstration required the rover to turn 90 degrees and drive three meters. The simple scenario enabled us to establish the basic interoperability preconditions and protocols between the MDS Framework, the robot, and its simulator.

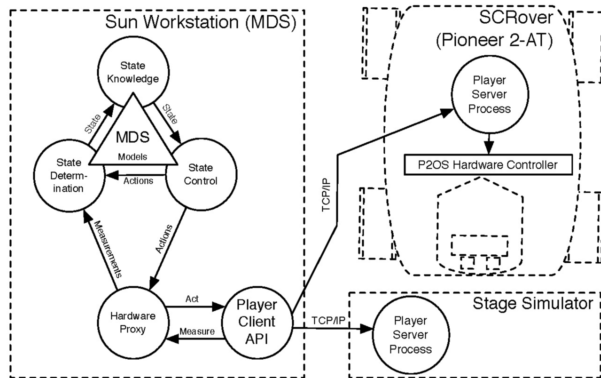


Figure 1 – MDS and SCRover high-level architecture

In Increment 2, we implemented reactive "wall-following" behavior. In this mode, the rover uses the laser rangefinder to determine the distance to the wall, drives forward while maintaining a fixed distance from that wall, and turns both inside and outside corners when it encounters them. An additional state in this behavior is that of the laser rangefinder's profile of obstacles (walls) in its surroundings. This scenario, involving both sensing and controlled locomotion (including reducing speed when approaching obstacles), provided an initial representative capability for technology evaluation.

3.2.2. State knowledge. State Knowledge is used to maintain the current state of the rover. For the two behaviors implemented, we adapted two State Knowledge components. One was called the PositionAndHeading state variable and holds the estimated position of the rover. The other is called the Obstacle State Variable and holds the estimated position of the nearest wall(s) in its frontal 180 degree view.

3.2.3. State control. The purpose of the State Controller is to collect the robot's current state from the State Knowledge components and to generate the proper commands for the robot to achieve the goal being executed. The commands generated then get submitted to the Hardware Proxy. For the two behaviors described, we built a controller that subscribed to the Obstacle State Variable and the PositionAndHeading State Variable. The controller would use this state information to generate the correct movement commands.

3.2.4. Hardware proxy. Our implementation of the Hardware Proxy in the MDS Framework is a stub for the Player rover API. Player [9] is developed at USC to communicate with the Pioneer family of robots. Player supports driving the robot's wheel motors, controlling the camera's pan/tilt unit, and querying a variety of on-board sensors. Access to these functions is provided through a client API, which communicates with a server process running on the rover itself. The client-server interaction can be conducted over a TCP/IP link, allowing us to execute the MDS Framework on a machine separate from the rover's on-board PC. Our MDS Hardware Proxy makes calls to a Player client shared library, various functions of which allow us to operate the drive motors, operate the camera, read the rover's position (maintained by the Player server process), and obtain a profile of the environment generated by the laser rangefinder. Figure 1 details this interaction.

3.2.5. State determination. Another component that we adapted is the State Determination Component. This component takes the sensor readings from the Hardware Proxy and uses this information to estimate current state of the robot. Once a state has been estimated, this information gets stored in the State Knowledge component. For the two behaviors described, we adapted two State Determination components. One component estimates the position and heading of the robot using the wheel sensors as its data while the other component estimates distance to the nearest wall with its laser rangefinder's values.

3.3. SCRover testbed support capabilities

To facilitate experiments using the SCRover testbed, the testbed provides several additional support capabilities. These capabilities include seeded defects, code instrumentation, scenario drivers, an instrumented development process, and experimentation guidelines.

3.3.1. Seeded defects. Suppose an experiment shows that in a given situation, the technology being evaluated finds 3 defects. How can we tell whether this is 100% of 3 defects or 3% of 100 defects? The best technique found to date is the seeded defect technique adapted from previous statistical techniques to software testing [18]. If we insert 10 representative defects into the software, and the technology being evaluated finds 6 of them, the maximum likelihood estimate is that the technology has found 60% of both the seeded and the unseeded defects. In general, if we insert I seeded defects, and the technology finds S seeded defects and U unseeded defects the maximum likelihood estimate of the total number T of unseeded defects is $T = I*(U/S)$.

Of course, this estimate is only as good as the assumption that the seeded defects are representative of the remaining defects [23]. We have tried to avoid the

known shortfall of people's inability to invent sets of representative defects by using as our pool of seeded defects the defects *actually* found in the specifications and code through peer reviews and a formal architecture review by JPL personnel. Researchers conducting their experiments simply modify a configuration file to insert selected defect(s) into the code without the need to recompile it. Once the configuration file is changed, the researcher can run the code with the defects and try to detect them with his/her technology.

3.3.2. Code instrumentation. The SCRover testbed provides guidelines to the researchers on how to instrument the code for collecting the statistics they wish to track. For the first set of analysis performed with the SCRover testbed, the development team implemented an instrumentation class on top of one of the features offered by the MDS Framework that allows programmers/ researchers to report events that occur in the code. The instrumentation class generates an output file containing a list of the events that occurred in the system and when each one of them happened. This file can then be analyzed by a researcher.

3.3.3. Scenario drivers. A mission/scenario is specified in MDS by using the Goal Elaboration Language (GEL). At the beginning of a mission, a scientist passes the GEL file to the rover and the rover executes the mission as stated in the file. Researchers who wish to create their own scenarios with the SCRover system may create their own GEL files. To create a goal for the rover to execute, researchers fill in the goal statement located in the GEL file with the appropriate values and the interval during which the goal should be achieved. Guidelines on how to create a GEL file are included in the SCRover testbed. Researchers can also execute the provided scenario drivers by simply executing the right command with the right GEL file. Currently, the SCRover system offers two GEL files for researchers to execute: the Increment 1 (MRE4) and Increment 2 (wall-following) scenarios described in section 3.2.1.

3.3.4. Instrumented development process. The SCRover team used a well-instrumented version of the Win-Win Spiral model called Model-Based (System) Architecting and Software Engineering (MBASE) [3,22] for system and software development. MBASE involves the concurrent development of the system's operational concept, prototypes, requirements, architecture, and life-cycle plans, plus a feasibility rationale ensuring that the artifact definitions are compatible, achievable, and satisfactory to the system's success-critical stakeholders. MBASE shares many aspects with the Rational Unified Process (RUP) [14], including the use of the Unified Modeling Language (UML) [4] and the spiral model anchor point milestones [1]. The SCRover team was experienced in its use.

While executing the development strategy, the team was able to collect data about the development process using various instrumentation techniques. In addition to the aforementioned defects found in the SCRover artifacts, the SCRover team kept track of its effort spent on the project. The effort data covered all the tasks performed by the SCRover team which includes writing each MBASE document, the system engineering aspects of the project, tool support, the defect reviews, coding, and testing. In addition, the developers used a tool called Hackystat developed by NSF-HDCP researcher Philip Johnson to collect the effort spent in coding the system [13].

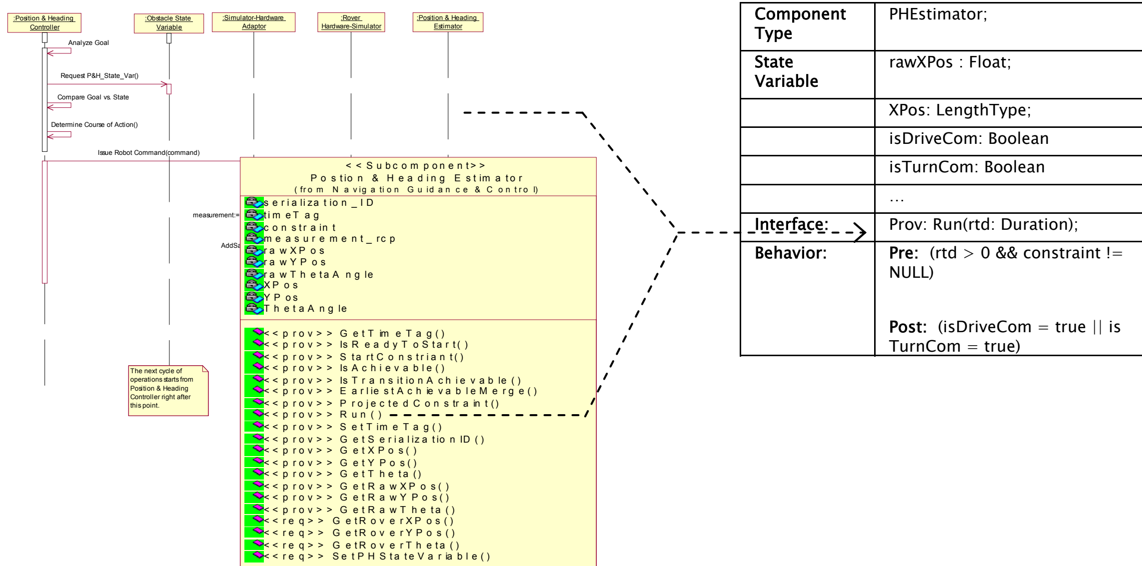
3.3.5. Experimentation guidelines. Additional experimentation guidelines are being developed by the Fraunhofer Center at the University of Maryland to provide guidance on designing sound experimental evaluations, on which experimental technique is best for a given situation, and on most appropriate statistical data analysis techniques.

4. Initial technology evaluation: Mae and AcmeStudio

4.1. Mae technology summary

The development team used UML to specify use-cases, class diagrams, and sequence diagrams that combine to describe SCRover's functionality visually and graphically. Further refinement of these diagrams into implementation-level specification helps the developers in building the "right" system. However, UML's lack of a precise semantic underpinning prevents reliable detection of inconsistencies, mismatches, and other classes of defects. Beside the basic UML syntax checking provided by the Rational Rose tool [19] used by the project, the only mechanism to detect such errors was peer-review of the UML diagrams. These steps, although useful, are not sufficient in ensuring correctness of the specification.

USC's Mae technology serves as an intermediate step between the UML diagrams and the implemented system. Mae is an extensible architectural evolution environment developed on top of xADL 2.0 [6] that provides functionality for capturing, evolving, and analyzing functional architectural specification [21]. A set of XML extensions were developed to model specific characteristics of MDS architectures. Consequently, Mae-MDS models of SCRover capture all functional properties of MDS style architectures [27].



Component Type	PHEstimator;
State Variable	rawXPos : Float;
	XPos: LengthType;
	isDriveCom: Boolean
	isTurnCom: Boolean
	…
Interface:	Prov: Run(rtd: Duration);
Behavior:	**Pre:** (rtd > 0 && constraint != NULL) **Post:** (isDriveCom = true || is TurnCom = true)

Figure 2. Position and Heading component's UML class and sequence diagrams and corresponding partial specification as captured by Mae.

4.2. Experimental application of Mae technology to SCRover

The Mae-MDS specification of SCRover architecture was built by refining the existing UML diagrams. In particular, the UML class and sequence diagrams were used in determining the architectural configuration of SCRover in terms of components and connectors, and their interaction. The components' specifications were then further refined to specify components' interface types (ports), associated interfaces (signatures), and the pre- and post-conditions that describe their static behaviors. The combination of this information, along with the domain knowledge of the MDS architectural style [7] was used in building SCRover models that can be analyzed in Mae.

The example depicted in Figure 2 shows the class and sequence diagrams of the Position and Heading Estimator component, and its corresponding partial Mae-MDS model. The Mae-MDS model is obtained by refining the UML's diagrams. The original class diagrams specify component interfaces but fail to model corresponding operations. Without detailed specification of component's operation, ambiguity arises in implementing the system. Additionally, since UML diagrams cannot be automatically analyzed, inconsistencies in the specification can hamper their usefulness. The UML sequence and class diagrams and specified component's operations associated with the interfaces were used to formally identify input/output parameters and conditions that must be satisfied prior to and after the corresponding interface is invoked. These pre- and post-conditions are modeled in first-order logic in Mae and thus can be further analyzed by the Mae tools.

The analysis provided by Mae revealed several inconsistencies. These inconsistencies correspond to mismatches in the interface and behavioral specification of components' services [17].

The particular classes of defects detected by Mae were especially important in the context of the seeded defects. As part of our design and implementation process, we identified a set of defects in the requirements and UML specifications. These defects were classified under a categorization schema similar to Orthogonal Defect Classification [5] and their severity was identified. We then reviewed the defects and planted them into the Mae-MDS specification of SCRover where possible, to identify and track the class of defects that Mae analysis can detect.

4.3. Experimental results

As part of the standard peer-review process of our UML design documentation a set of 38 defects were identified and classified. The classification identifies each defect and assigns one of the predefined defect types of interface, object/class/function, method/logic/algorithm, ambiguity, data value, and other to each defect.

The nature of the above 38 defects varied from English language problems and typographical errors, to sophisticated errors that could potentially cause harmful behaviors; some of them were architectural in nature while others were conceptual. A subset of architectural defects concerned functional behaviors that Mae-MDS

models capture, while others relate to other issues not currently captured by our models. Re-seeding these defects into the Mae-MDS models helped us identify and further classify the defects that MDS adaptation of Mae can detect. It also reveals the types of defects that Mae cannot detect, which is valuable in identifying complementary technologies necessary to detect additional classes of architectural defects.

Out of the 38 identified defects, we were able to seed 24 (63%) of them back into the Mae-MDS specification. The remaining 14 are conceptual defects that do not directly translate to functional specification of the system or its behavioral properties. Examples of this type of defects that Mae models do not capture is "*Inaccurate purpose for a given component X*", or "*Class Y should be split into classes Y1 and Y2*".

The result of Mae analysis on the models containing the 24 seeded defects is as follows:

- Mae analysis revealed 15 errors (62%).
- Out of 9 defects not detected by Mae, 7 did not directly impact the execution of the system. Examples include specification of component's provided interface or state variable (attributes) that were never used or required elsewhere.
- Mae was unable to detect 2 critical defects in the specification. These defects would result in harmful interactions that undermine system's operation. Particularly, these defects concerned stylistic constraints of MDS architectural style. An example of this type of defects is "*Communication direction between X and Y must be reversed*".
- Mae analysis revealed 6 additional defects that were previously undetected by the review process. These defects primarily concerned the inconsistency in the specification. Specifically, Mae detected inconsistent specification of interfaces and behaviors among interacting components, resulting in possibly harmful interactions in the system.

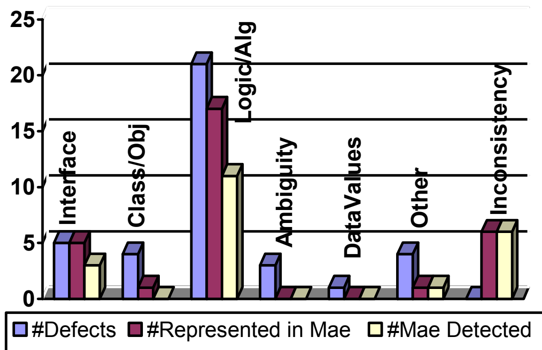


Figure 3 – Mae defect detection yield by type

Figure 3 summarizes the original number of defects (left column) against the subset that can be captured in Mae-MDS models (middle column), and those detected by Mae (right column).

Incorporating defect seeding analysis to these results also demonstrates that, since Mae detected 15 of 38 seeded defects as well as 6 unseeded defects, the maximum likelihood estimate of the total number of remaining defects is T = 38*(6/15) = 15.

Since Mae found 6 unseeded defects, this leaves an estimate of 9 remaining defects. As a rough estimate of where to look for these defects, we can posit that their distribution is similar to the distribution of defects not found by Mae. This is often but not always true, as with other defect-proneness metrics such as module complexity metrics [25,26]. Table 1 shows the results.

Table 1. Seeded defect estimate of remaining defect distribution

Defect Class	Total	Interface	Class/ Object	Logic/ Algorithm	Ambiguity	Data Values	Other
Unfound Seeded Defects	23	2	4	10	3	1	3
Remaining Unseeded Defects	9	0.8	1.6	3.9	1.2	0.4	1.2

The availability of the testbed support capabilities made the effort to perform the translation from UML to Mae-MDS and the Mae tool runs relatively low. The total effort was roughly 160 hours of which about 50 hours was spent on adapting the tool to model MDS architectures, 80 hours was spent on building Mae-MDS models out of UML models and models while the remaining 30 hours was spent on building the model, using the tool, and performing the analyses.

The testbed evaluation also showed that Mae's analysis could be extended in two main directions that would result in further detection of architectural defects. First, Mae could perform stylistic constraint analysis that checks for specific defects related to MDS architectural style. This would result in detecting some of the interface and Logic/Algorithmic errors that were left undetected by Mae's current analysis utility. Additionally, Mae could perform protocol matching to ensure proper dynamic behaviors of components. Thus, the testbed usage resulted in insights and plans for maturing and extending Mae's defect detection capabilities.

4.4. Early results from AcmeStudio

An early opportunity to use the SCRover testbed to obtain comparative data on specification tool capabilities come with its recent experimental usage with the CMU Acme ADL [8] and AcmeStudio tool suite [28]. Among

their other capabilities, Acme is particularly good at representing characteristics of architectural styles, and AcmeStudio is particularly good at verifying whether a system's architectural specifications are in appropriate compliance with the relationships and constraints imposed by the architectural style. These aspects of Acme and AcmeStudio were experimentally applied to an extended set of SCRover UML specifications covering additional equipment such as batteries and optical camera, but excluding the seeded defects.

The major results of the experiment are:

- AcmeStudio was able to find 3 previously detected interface defects and 8 previously undetected defects involving compliance of the SCRover architectural specifications with the MDS architectural style.
- Although the full capabilities of AcmeStudio were not exercised, there were some defects found by Mae that would not be found by AcmeStudio and vice-versa.
- As with Mae, a number of ambiguities were found in translating the UML specs into Acme that represented potential defects that would be avoided by using Acme.
- The SCRover UML specs provided were not a good match to the state-oriented architecture used by the MDS. This could be improved by more extensive use of the UML State Machine constructs, which can be used at the system level and not just for modeling object lifetimes as in [4].
- The effort of roughly 120 hours required to perform the UML-Acme translation and AcmeStudio analysis was at a reasonable low level similar to that for Mae. Of those 120 hours, 80 hours was spent developing the architectural style, independent of the SCRover development, 30 hours was spent transforming the SCRover UML documentation to an architectural model in that style, and 10 hours was spent tailoring the environment, modeling the system, and conducting the analysis.
- The AcmeStudio researchers identified several improvements in the SCRover testbed package that could have reduced the experimental effort, such as the organization of and access capabilities for the testbed artifacts. These improvements are being made to the testbed package.
- A review of the results by Mae and AcmeStudio researchers indicated that combining their representations and tool capabilities was both feasible and advantageous. Explorations are now underway on the best ways to combine them.
- The objectives of creating an exportable and externally usable SCRover testbed were reasonably well met on this first attempt with valuable feedback on how to improve subsequent external usage.

5. Implications for JPL project use of Mae, AcmeStudio, and SCRover testbed results

The results of the Mae and AcmeStudio experiments with the SCRover testbed have been of considerable interest to JPL MDS personnel, who had been experimenting individually with their capabilities. The complementarity of their defect identification and avoidance capabilities, the relatively low level of effort in developing and analyzing the specifications, and the prospect of combining the two toolsets opened up new prospects for using ADLs to supplement MDS's current state-oriented architectural approach. Potential benefits include stronger defect avoidance, detection, and diagnosis; stronger compositional modeling of MDS components and connectors; and an overall strong return-on-investment (ROI) potential of software architecture modeling and analysis compared to that of traditional but expensive engineering review processes. The modeling effort required of software and system engineers has been convincingly low, in comparison to the added effort required for later manual defect detection and resolution.

JPL MDS personnel and CMU and USC researchers are now exploring collaborative approaches to combine their ADL-based capabilities and apply them to MDS in ways that could push the ROI even further. For example, with *a priori* data about component-level criticality and susceptibility to failures, extensions of static analysis that propagate static constraints to the topology of the software architecture along component/connector paths could form the basis for evaluating the vulnerability of the system due to coupled interactions among threads.

Even more promising is the possibility of performing this analysis continuously at runtime to maintain a level of self-awareness about criticality (because the goals on the system imply a number of state analysis elements) and vulnerability (because of past experience with failures), and use the resulting information to make better repair strategies. This would include leveraging available execution mechanisms and operating system features to partition and isolate critical components from couplings and interdependencies with potentially harmful components.

Some further open issues remain about scalability and applicability of the technologies to more complex robot configurations and mission scenarios. These are being addressed in the definition and development of the Increment 3 of SCRover testbed capability. Based on

discussions on architectural analysis priorities with the JPL Mars Science Laboratory project users of MDS, the Increment 3 capabilities will include obstacle avoidance, compensation for actuator drift, target encounter, payload capabilities, and multi-goal conflict resolution.

6. Conclusions, lessons learned, and future plans

1. Cost-effectiveness of Mae and AcmeStudio tools. Even in initial exploratory evaluations across somewhat different SCRover testbed configurations and limited mission scenarios, both Mae and Acme studio were cost-effective with respect to UML and peer-reviews in avoiding, detecting, and diagnosing mission-critical specification defects. Explorations are underway to extend the comparative evaluation to other specification technologies such a MIT's Alloy [12], USC's dynamic analysis tools [10], Stanford University's Maude high-performance reflective language and system [15], and University of Oregon's iSIM simulation tool [11]. Also, since the results were obtained on a relatively simple rover configuration and mission, efforts are underway to develop a significantly more extensive Increment 3 SCRover testbed.
2. Cost-Effectiveness of MDS and Player-Stage Frameworks. There is a non-trivial investment required in learning the frameworks and getting them to compile, run, and interoperate, but a significant acceleration in productivity thereafter. For example, it took two person-months to get the very simple Increment 1 MRE4 capability to work with SCRover, and only one person-month to develop the considerably more complex Increment 2 wall-following capability. Having the Player/Stage framework enabled us to implement the SCRover MRE4 capability in only 800 lines of code (LOC). This is a reduction of more than 80% over the 5000 LOC implemented by JPL for their version of the MRE4. The MDS Event Logging Function was also a significant timesaver in developing and applying the SCRover testbed instrumentation package.
3. Capabilities and limitations of seeded defect techniques. The seeded defect approach was effective in identifying the degree to which Mae could identify defects of various classes. However, after estimating 9 likely remaining defects, we found that AcmeStudio alone discovered 8 remaining defects, 5 of which were in categories (style usage, completeness) not in our defect categorization scheme. Thus it appears that the seeded defect technique's maximum likelihood estimate is better considered as a lower-bound estimate of the defects remaining in the categories constituting the current universe of defect sources. As an analogy, since the seeded defect technique derives from the use of fish tagging to estimate the total number of fish in a body of water, the technique can only estimate the number of fish catchable by the type of net used in catching tagged and untagged fish. There may be a number of smaller but significant fish (i.e., defects) swimming around undetected.
4. Testbed technology coverage: The SCRover testbed also includes requirements, code and test cases, but our initial experiments have focused on evaluation of architecture description language analysis tools, with some use of the ADL specifications for runtime assertion checking. Future plans for Increment 3 and beyond include testbed support for evaluating dependability technologies focused on requirements, code, or testing, and for evaluating combinations of technologies.
5. Testbed support scalability: The current SCRover testbed was able to provide a fairly low entry barrier for the Mae and AcmeStudio researchers, but only with a nontrivial amount of support by SCRover developers. Plans for Increment 3 include considerably more support by automated aids for researcher tailoring of mission scenarios, instrumentation, and experimentation. Examples are a command language and GUI for configuring and executing scenarios with various combinations of seeded defects, instrumentation, and data analysis.
6. Broad Participation and Teambuilding. Both for testbed technology and HDC technology adoption, user-supplier teambuilding is at least as important as technology excellence. This is particularly true when multiple stakeholders need to rapidly adapt to unforeseeable changes, which happened frequently during the SCRover testbed development and experimental use. The number and diversity of contributing authors of this paper is a good example of this teambuilding strategy in action.
7. Testbed ability to accelerate technology maturity and transition: The ability to evaluate alternative ADL-based specification technologies on the common SCRover testbed enabled both technology researchers and project personnel to identify previously unrecognized technology complementarities and opportunities to combine the technologies to achieve significant project dependability benefits. As discussed in section 5, JPL project personnel and USC and CMU researchers have come together to explore and expedite these technology opportunities. This provides encouraging evidence that the testbed approach can cost-effectively accelerate software engineering technology maturity and transition.

7. Acknowledgements

This work was supported by NASA-HDCP contracts to CMU, JPL, and USC. It also benefited from significant support by USC's A. Winsor Brown, Scott Chen, Keun Lee, Shauna Madrigal, Gaurav Sukhatme, and Denis Wolf; by JPL's Dan Dvorak, Alex Moncada, Bob Rasmussen, George Rinker, Al Sacks, Marcel Schoppers, Brian Vickers, David Wagner, and Chengxing Zhai; and by CMU's Michael Evangelist, Bradley Schmerl, and Dehua Zhang. We also wish to acknowledge the cooperation and continuous help with xADL tools provided by the developers of xADL: Eric Dashofy, Andre van der Hoek, and Richard Taylor.

8. References

[1] B. Boehm, "Anchoring the Software Process", *IEEE Software*, July 1996, pp. 73-82.

[2] B. Boehm, C. Abts, A.W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000.

[3] B. Boehm and D. Port, "Balancing Discipline and Flexibility with The Spiral Model and MBASE", *Crosstalk*, December 2001, pp. 23-28 (http://www.stsc.hill.at.mil/crosstalk)

[4] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1999.

[5] R.Chillarege, I.S.Bhandari, J.K.Chaar, M.J.Halliday, D.S.Moebus, B.K.Ray, and M-Y.Wong, "Orthogonal Defect Classification- A Concept for In-Process Measurements", *IEEE Transactions on Software Engineering*, 18(11), 1992.

[6] E.M. Dashofy, A. van der Hoek, and R.N. Taylor, "An Infrastructure for the Rapid Development of XML-based Architecture Description Languages", *In Proceedings of the 24th International Conference on Software Engineering (ICSE2002)*, Orlando, FL, May 2002, pp 266-276.

[7] D. Dvorak, R. Rasmussen, G. Reeves, and A. Sacks, "Software Architecture Themes In JPL's Mission Data System" *In Proceedings of the AIAA Space Technology Conference and Exposition*, Albuquerque, NM, September, 1999.

[8] D. Garlan, R. Monroe, and D. Wile, " Acme: Architectural Description of Component-Based Systems ", , Foundations of Component-Based Systems, Gary T. Leavens and Murali Sitaraman (eds), Cambridge University Press, 2000, pp. 47-68.

[9] B. Gerkey, R. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems", *In Proceedings of the 11th International Conference on Advanced Robotics*, Portugal, June 2003.

[10] A. Helmy, D. Estrin, "Simulation-based `STRESS' Testing Case Study: A Multicast Routing Protocol", *IEEE Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Canada, 1998.

[11] iSIM Context Simulator, URL: *http://www.cs.uoregon.edu/~jprideau/iSIM/isim.html*

[12] D. Jackson, "Alloy: A Lightweight Object Model Notation", Technical Report 797, MIT Laboratory for Computer Science, Cambridge, MA, February 2000.

[13] P. Johnson, Project Hackystat: "Accelerating adoption of empirically guided software development through non-disruptive, developer-centric, in-process data collection and analysis", Department of Information and Computer Sciences, University of Hawaii, 2001.

[14] P. Kruchten, The Rational Unified Process (2nd ed.), Addison Wesley, 2001.

[15] P. Lincoln, M. Clavel, F. Durán, S. Eker, et. al, "Towards Maude 2.0", In the 3rd International Workshop on Rewriting Logic and its Applications (WRLA'00) - Electronic Notes in Theoretical Computer Science, 2000.

[16] J. Mankins, "Technology Readiness Levels", NASA Office of Space Access and Technology, April 1995.

[17] N. Medvidovic, D.S. Rosenblum, and R.N. Taylor, "A Language and Environment for Architecture- Based Software Development and Evolution", *In Proceedings of the 1999 International Conference on Software Engineering*, Los Angeles, CA, May 1999, pp.44-53.

[18] H. Mills, "On The Statistical Validation of Computer Programs", IBM Federal Systems Division Report 72-6015, 1972.

[19] Rational Rose Tool, URL: *http://www.rational.com/products/rose/index.jsp*

[20] S. Redwine and W. Riddle, "Software Technology Maturation", *In Proceedings of the 8th International Conference on Software Engineering (ICSE1985)*, August 1985.

[21] R. Roshandel, A. van der Hoek, M. Mikic-Rakic, and N. Medvidovic, "Mae - A System Model and Environment for Managing Architectural Evolution", *ACM Transactions on Software Engineering and Methodology* (In review) 2002.

[22] USC Center for Software Engineering, "Guidelines for Model-Based (System) Architecting and Software Engineering", 2003 (http://sunset.usc.edu/research/MBASE).

[23] J. Voas and G. McGraw, *Software Fault Injection*, Wiley, 1998.

[24] T. George and R. Powers, "Closing the TRL Gap", *Aerospace America*, August 2003, pp. 24-26.

[25] T. McCabe, "A Complexity Measure" *IEEE Trans. Sw. Engr., SE-2(4)*, pp-308-320

[26] M. Halstead, *Elements of Software Science*, Elesvier, 1997

[27] R. Roshandel, B. Schmerl, N. Medvidovic, D. Garlan, and D. Zhang, "Using Multiple Views to Model and Analyze Software Architecture: An Experiment Report", *USC Technical Report Number USC-CSE-2003-508*, 2003.

[28] B. Schmerl and D. Garlan, "Exploiting Architectural Design Knowledge to Support Self-repairing Systems", The 14th International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, 2002.