

# A Unified Model of Dependability: Capturing Dependability in Context

**Victor Basili**, *University of Maryland and Fraunhofer Center for Experimental Software Engineering*

**Paolo Donzelli and Sima Asgari**, *University of Maryland*

**I**n contemporary societies, individuals and organizations increasingly depend on services delivered by sophisticated software-intensive systems to achieve personal and business goals. So, a system must have engineered and guaranteed dependability, regardless of continuous, rapid, and unpredictable technological and context changes.

The International Federation for Information Processing Working Group 10.4 ([www.dependability.org](http://www.dependability.org)) defines dependability as “the trustworthiness

of a computing system which allows reliance to be justifiably placed on the services it delivers.” “Reliance” is contextually subjective and depends on the particular stakeholders’ needs. Depending on the circumstances, different stakeholders will focus on different systems attributes, such as availability, performance, real-time response, and ability to avoid catastrophic failures and resist adverse conditions, as well as different levels of adherence to such attributes. Additionally, an attribute can mean

different things to different people; you often see different definitions given for the same attributes.<sup>1-3</sup> Put another way, dependability assumes a precise meaning only when applied to a specific context: the system and the stakeholders’ goals it must support. So, achieving and maintaining dependability in a quickly changing context requires you to firmly understand its meaning.

From this perspective, we introduce our Unified Model of Dependability, a modeling language for discussing and reasoning about dependability. By providing a structured framework for eliciting and organizing dependability needs, UMD helps stakeholders build dependability models that clearly identify the measurable, implementable properties individual systems need to be dependable for their users.

**Dependability is a key systems property that must be guaranteed regardless of continuous, rapid, and unpredictable technological and context changes. Our Unified Model of Dependability lets you reason about dependability and turn it into clearly defined, implementable system properties.**

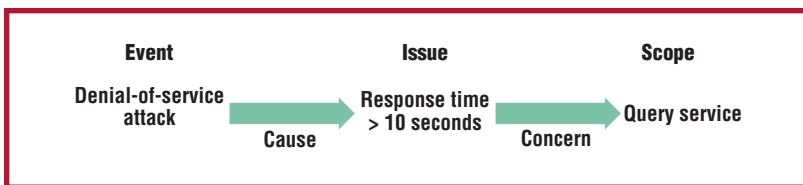
## WHY READ THIS ARTICLE?

Imagine for a moment that you have a complex system with numerous properties that must all be maintained within certain limits during the system's operation. When a specific negative event such as a denial-of-service attack occurs, what will its impact be on various properties such as system safety, availability, security, and other "-ilities"?

For most systems, such questions can be surprisingly difficult to analyze due to the diverse ways in which a single event can affect different system properties. In this ar-

ticle, Victor Basili, Paolo Donzelli, and Sima Asgari describe their Unified Model of Dependability approach to capturing and analyzing the impacts of diverse events on critical system properties. Their UMD method demonstrates the recurring theme that you can make -ilities more enduring and survivable over time by capturing and supporting a wider range of context information about those properties.

—Terry Bollinger, Jeffrey Voas, and Maarten Boasson, guest editors



**Figure 1. Unified Model of Dependability concepts and their relationships with an online application system example.**

### UMD

Traditional dependability modeling usually develops in a top-down fashion. For example, let's focus on a specific dependability attribute, *performance*, and assume we want to define what performance means for a specific service of a given system (for example, an online application's query service). First, we need to define performance—for example, "performance is a static or dynamic system's capability (response time, throughput) defined in terms of an acceptable range."<sup>4</sup> Then, on the basis of this definition, we can specify the desired behavior. So, our online application's query service could have this performance requirement: "The response time must be no greater than 10 seconds." In this way, we've obtained a model of the system's performance starting from a generic attribute definition. This simple model defines what we expect from the system: When the query service responds in more than 10 seconds, we can say there's a performance failure, or a lack of performance. However, the same failure could also represent a lack of other—perhaps even more relevant for the stakeholders—system properties. For example,

- If the online application supports an emergency response operator, a delayed response could result in a dangerous situation, so failure would be also a hazard, representing a lack of safety.

- If the online application is an e-commerce system, a delayed response could discourage potential customers from buying (the service is unavailable), so this misbehavior represents a lack of availability.
- The response delay could occur because of an internal fault but also because of an external event—for example, an *unintentional* external event, such as a hardware fault, or an *intentional* one, such as a denial-of-service attack. In this case, the performance failure becomes a lack of survivability (unintentional) or of security (intentional).

These simple examples show how you could interpret the same failure differently. Given this one-to-many relationship between failures and attributes, you can more efficiently and intuitively model dependability by assuming failure as a starting point (that is, think bottom-up and not only top-down). As Brian Randall points out, there's a clear need to go beyond terminology (that is, attributes definitions) and focus on relevant concepts (such as failure).<sup>5</sup>

UMD builds on this need. We designed it to focus on a *dependability issue* (that is, an undependable behavior, such as a failure) to help stakeholders build dependability models of individual systems. UMD (see Figure 1) lets stakeholders model dependability by defining an actual issue that shouldn't affect the system or a service (*scope*) along with a possible responsible external *event* that might cause it.

### Supporting elicitation

We provide stakeholder guidance by incorporating into UMD the models, definitions, and classifications adopted in the literature to discuss these concepts. For example (see Figure

2), we could categorize issues into the (not exclusive) concepts of failures and hazards to help stakeholders identify potential system misbehaviors.

We can further refine these concepts by introducing subclassifications for both failures and hazards—for example, by classifying them by type or other characteristics (such as failure availability impact in Figure 2). Depending on the specific needs, you could adopt different standards and classifications, for example, MIL-STD-882<sup>6</sup> for hazards and ISO/IEC 13236<sup>7</sup> for failures. Similarly, you could introduce ad hoc definitions and classifications for event and scope.

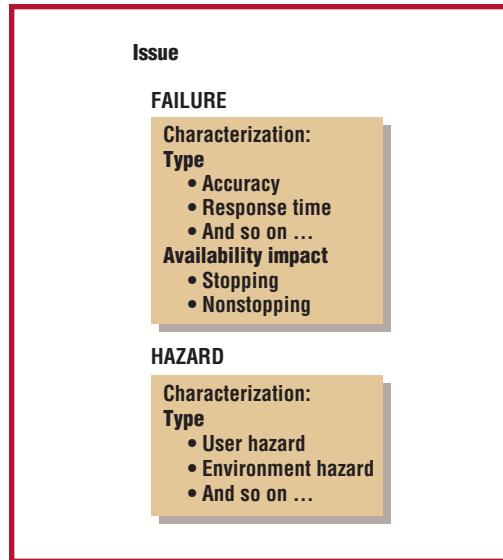
In this way, UMD guides stakeholders during the elicitation process. Specifically, stakeholders can use items already available or tailor, expand, and modify them according to their specific needs. For example, stakeholders can

- Use the definitions of failure types already present
- Use the same failure types but provide different definitions
- Introduce more specific failure types—for example, response time rather than performance

UMD’s bottom-up nature lets stakeholders take advantage of existing dependability knowledge (models and classifications). So, the UMD framework consists of

- *Invariant concepts*: These are stable for every UMD application (issue, scope, and event).
- *Semi-invariant concepts*: These are the structure of the concepts’ characterization (for example, mapping issue to failure, hazard, or both).
- *Customizable concepts*: These are the characterizations (for example, classes of failures, hazards, events, and so on). They depend on the specific context (project and stakeholders) and can be customized while applying UMD.

The somewhat arbitrary distinction between concepts represents the status of our dependability knowledge. We can hypothesize that as our knowledge increases and as classifications and models become recognized as standards, concepts will move from the frame-



**Figure 2. Stakeholder guidance for specifying issues.**

work’s lower to upper levels.

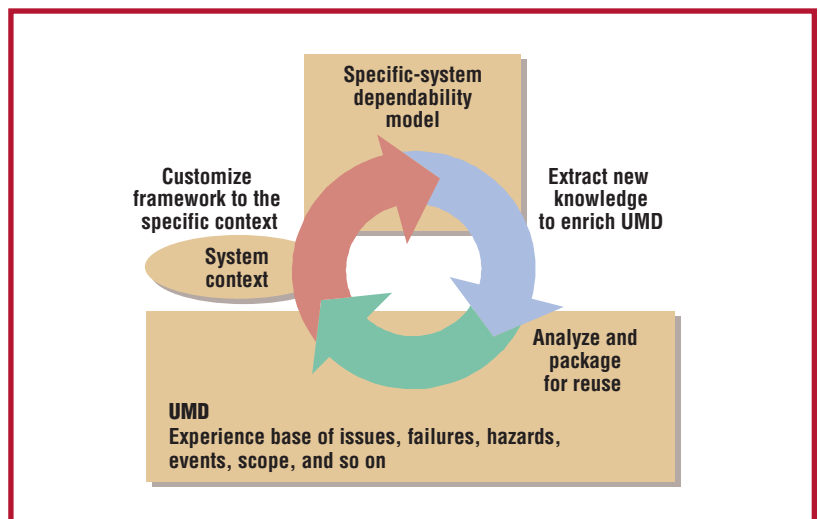
As Figure 3 shows, you can view UMD as an experience base that supports stakeholders in building a specific system’s dependability model. The knowledge embedded in the UMD experience base can be customized and provides guidance while eliciting specific context needs.

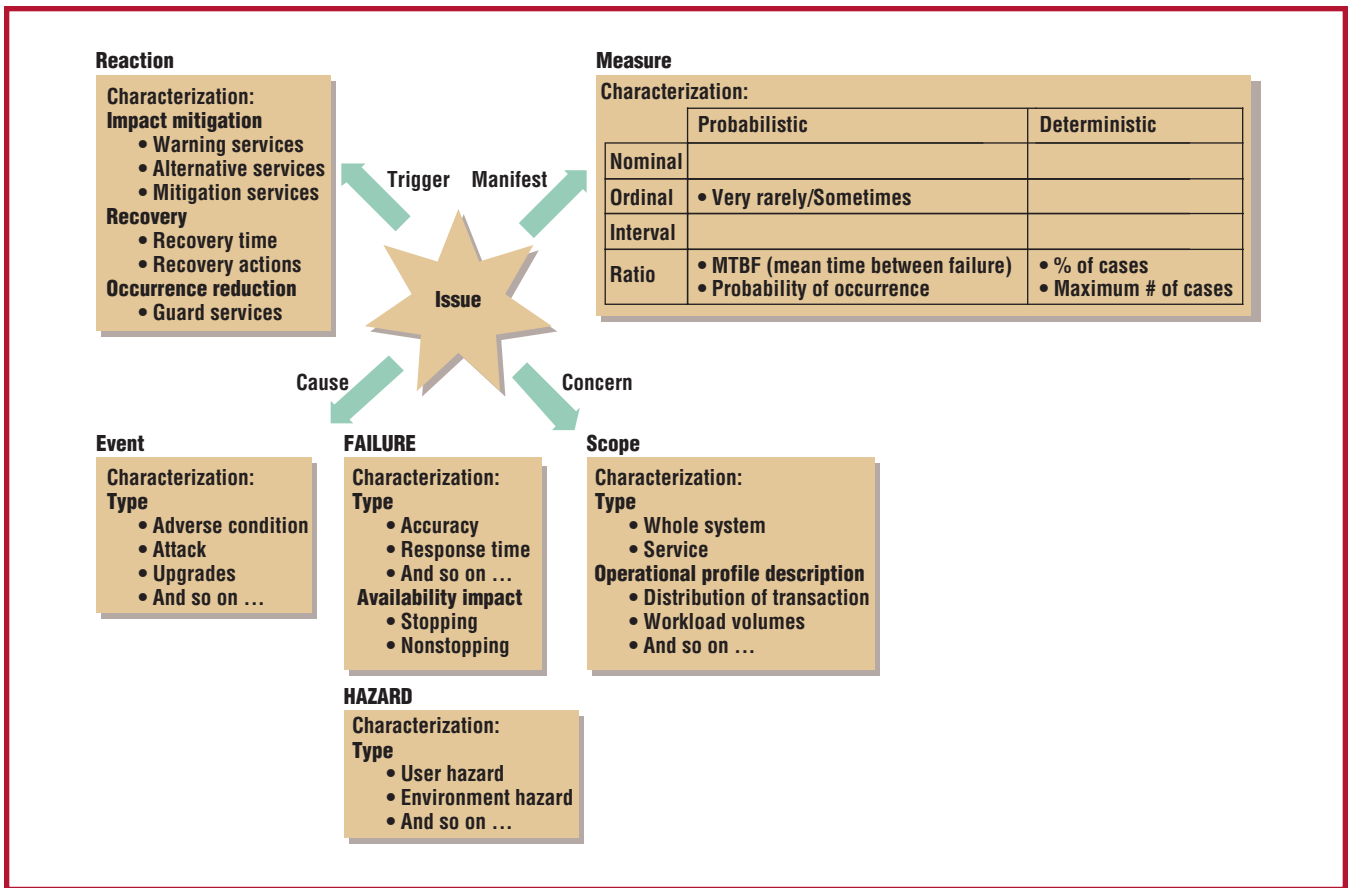
The new knowledge acquired while building the system dependability model can be analyzed and packaged for reuse in UMD.

### Measuring dependability

UMD lets stakeholders specify the issues they don’t want to occur. However, this doesn’t suffice; we need an operational definition of dependability. For example, Jean-Claude Laprie says, “The extent to which a system possesses the attributes of dependability should be interpreted in a relative, probabilistic sense, due to the unavoidable occurrence of failures.”<sup>8</sup> UMD provides measurement models via the

**Figure 3. The updating process from version 11.23 to version 11.25, showing how to add a new column in the Bank table.**





**Figure 4. UMD's structure.**

invariant concept *measure* (see Figure 4). In this way, stakeholders can choose the measurement style most appropriate for describing an underlying issue's acceptable manifestation levels. For example, Figure 4 shows the following measurement classes:

- *Ordinal and probabilistic*—for example, an ordinal scale such as “very rarely,” “rarely,” and “sometimes”
- *Ratio and probabilistic*, such as MTBF (mean time between failure) and probability of occurrence (in the next time unit or transaction)
- *Ratio and deterministic*, such as number of occurrences (in a given time frame)

### Improving dependability

UMD lets stakeholders provide ideas for improving dependability through the invariant concept of *reaction*. In this way, stakeholders can suggest *reactive* and *proactive* services the system should provide to become more dependable. The stakeholder adds reactive services triggered by an issue to be warned of the situation or to try to reduce an issue's consequences. The stakeholder adds proactive services to the system to further reduce the probability of an issue's occurrence, provide alternative ways of

performing the same tasks, or allow a quicker recovery (for example, automatic data backup). We propose the following classification for reaction (see Figure 4):

- *Warning services* warn users about what happened or is happening (for example, “if response time exceeds 10 seconds, warn the user about the delay”).
- *Alternative services* help users perform their tasks regardless of the issue.
- *Mitigation services* reduce the issue's impact on users (for example, “if response time exceeds 10 seconds, suggest a better time to try again”).
- *Recovery behavior* is the time required to recover from the issue (for example, expressed as MTTR (Mean Time to Recover) and the kind of required intervention (for example, user or technician).
- *Occurrence reduction* guards against the issue—that is, to reduce the probability of occurrence (for example, preventing saturation or trashing by rejecting incoming requests). Stakeholders can extend this idea to capture any suggestion they might have to prevent the issue from happening (such as modifying existing services, design changes, and so on).

## The UMD Tool

We developed a Web-based tool that implements UMD, organized around two tables:

- The Scope frame (see Figure 5a) lets the stakeholder identify all system services for which dependability could be a concern. For the system and each identified service, a stakeholder must provide an identifier (name) and a brief description.
- The Issue frame (see Figure 5b) lets users specify their dependability needs by selecting and defining potential issues, their tolerable manifestations and scope, possible triggering events, and desired system reactions.

## Applying UMD

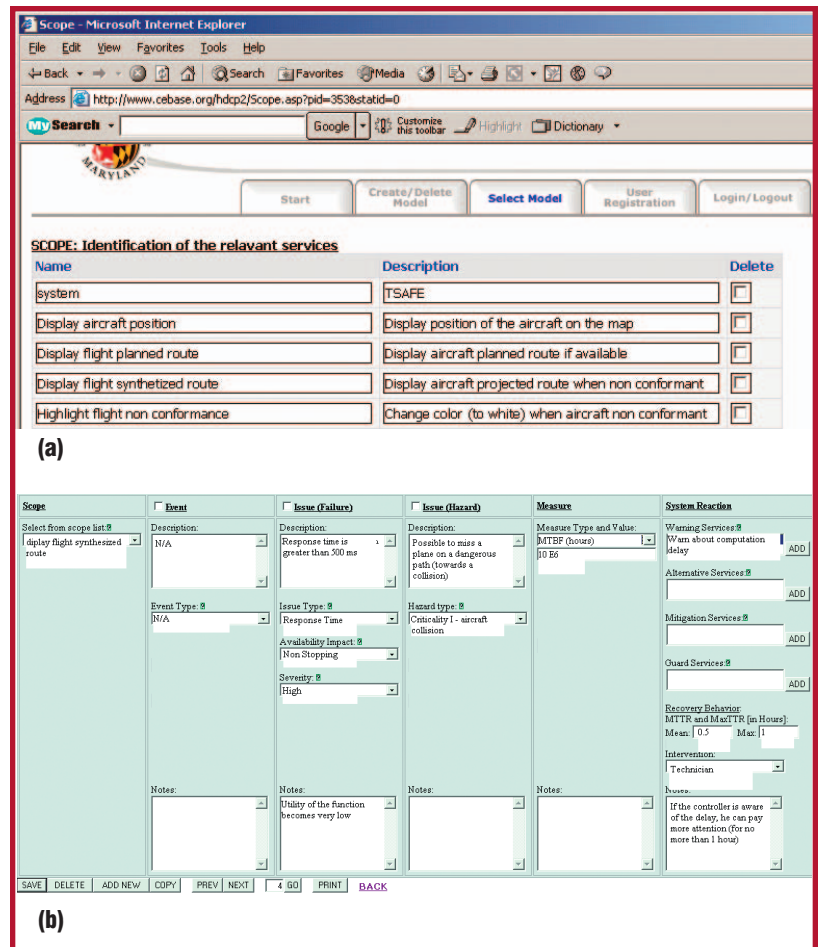
Within the NASA High Dependability Computing Program, we performed a feasibility analysis of the UMD concept. As part of a larger technology evaluation program, we aimed to develop a dependability model of the Tactical Separation Assisted Flight Environment testbed.<sup>9</sup> TSAFE is a software system designed to aid air traffic controllers in detecting and resolving short-term conflicts between aircraft. We derived the adopted testbed from the TSAFE version that Gregory Dennis developed.<sup>10</sup>

TSAFE provides air traffic controllers with a graphical representation of the conditions (position, planned route, and forecasted synthesized route) and status (conformance or non-conformance to a planned route) of selected flights in a defined geographical area. It aims to detect conflicts between three and seven minutes in the future and issue avoidance maneuvers accordingly.

TSAFE already had available a set of functional requirements defining the system. However, it lacked precisely stated dependability requirements, although dependability is a main concern given the application domain (Dennis's work focused on other aspects). So, we used UMD to produce a dependability model.

## Data gathering

For the case study, a small group of computer science researchers and students acted as stakeholders (air traffic controllers) after a short introduction to TSAFE. This initial case study aimed to evaluate the suggested approach's feasibility rather than to identify TSAFE's correct dependability requirements. All



**Figure 5. UMD tool tables: (a) Scope and (b) Issue (not related to an external event).**

acting stakeholders interacted with the UMD tool through an analyst. This helped to better evaluate the tool's capabilities and represent real-life situations in which stakeholders might be unfamiliar with automatic tools.

We applied UMD in two main steps:

- *Scope definition.* By analyzing the already available functional requirements, all stakeholders, working together and supported by the analyst, selected the TSAFE main services that they believed to be relevant for dependability. The scope table shows the resulting four services (see Figure 5a).
- *Model building.* Each stakeholder, supported by the analyst and guided by the structure tool provided, filled as many tables as necessary to define her or his dependability needs (see Figure 5b).

While applying UMD, stakeholders used the available characterizations and, whenever necessary, extended them with their own definitions. Figure 6 provides some details about the failure characterization obtained for TSAFE (reconciled among the different stakeholders).

## FAILURE

### Characterization:

#### Type

- **Accuracy.** Data (flight position, trajectory, and so on) aren't displayed with the required accuracy.
- **Throughput.** System or service fails to handle the desired number of flights.
- **Response time.** System or service fails to respond within the desired time.
- *And so on...*

#### Availability impact

- **Stopping.** System or service becomes unfit for use.
- **Nonstopping.** System or service is still usable.

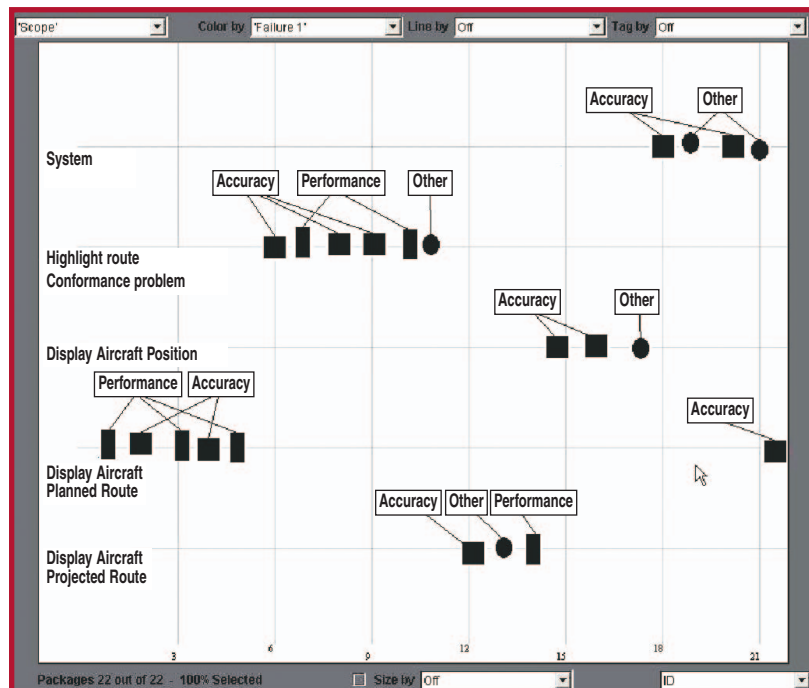
#### Severity

- **High severity.** Major impact on the system's utility for the operator.
- **Low severity.** Minor impact on the system's utility for the operator.

**Figure 6. The failure characterization obtained for Tsare.**

To demonstrate the kind of data collected, we describe a table filled in by a single stakeholder—specifically, for an issue not related to an external event (see Figure 5b). The stakeholder signals a potential failure for the service “display flight synthesized route,” when the response time is greater than 500 ms. This is a response time, non-stopping, high-severity failure, given the high impact on the service’s utility. For the stakeholder, this failure is also a hazard, given that he thinks he could miss spotting a plane on a dangerous path. To be more confident in the system, the stakeholder then asks to introduce a warning service that will advise if the computational time becomes greater than 500 ms. This will alert the operator to the need for additional attention. Finally, the stakeholder sees this as a mis-

**Figure 7. Data graphical analysis.**



sion-critical failure (transformed by the analysts into an MTBF of 10 E6) and asks for a technician to perform the recovery within one hour. If this failure condition lasts for more than an hour, the stakeholder feels he won't be able to properly perform his duties because of the need to maintain a higher than usual level of attention.

## Data analysis

Once the tables were completed, the analyst presented the stakeholder with the results to date. The analyst performed both graphical and computational analyses. The UMD tool incorporates the Visual Query Interface tool,<sup>11</sup> which allows quick data visualization, navigation, and assessment. Figure 7 shows the distribution of the identified issues around the main services, highlighting (with different colors and shapes) the corresponding characteristics. Additionally, the MS Access database the UMD tool produces supports different types of computations. In particular, the analyst could combine the measures expressing the tolerable manifestation for each of the identified issues to provide aggregated values of dependability. For instance, he computed the aggregated probability of occurrence of all the accuracy failures, all the accuracy failures that were also stopping failures, and accuracy failures for a specific service. Similarly, with the tolerable manifestation of the stopping failures defined and knowing the corresponding desired recovery time, he could compute the desired availability (again, for the whole system or a specific service).

Based on the analysis, the stakeholder was able to make changes and the analyst pointed out possible risk areas. For example, the issue distribution (see Figure 7) showed that a service hadn't been taken into account, so the analyst asked the stakeholder to confirm such a choice. Also, the computational analysis showed that some failures were dominant, representing a bottleneck for the whole system's dependability; in some cases, the stakeholder wanted to revisit his choices. The stakeholder wasn't satisfied with the system's computed availability, so he revised some of the choices made while filling in the tables. This assessment led to further refinement of the tables' entries and completion of new ones. The iteration ended when both the analyst and the stakeholder felt confident about the results.


## About the Authors

### Data reconciliation

At this point, the analyst had to reconcile the different stakeholders' emerging needs. Example reconciliations included

- When two or more stakeholders had filled in tables concerning the same service but identified different classes of failures
- When stakeholders asked the system to behave in incompatible ways (for example, asking the system to simultaneously stop and provide an alternative service)
- When combined stakeholder requests couldn't be addressed with available resources

**T**he case study results increased our confidence in UMD's ability to act as a modeling language to discuss, gather, represent, and make measurable dependability needs. UMD provided valuable support in building a precise dependability model of TSAFE, making explicit and combining different stakeholders' needs. Additionally, the approach's flexibility also lets you model system properties that usually aren't considered strictly related to dependability, such as usability.

Our future work will develop in two main directions. First, we want to perform further empirical assessments, in particular to understand how to better employ UMD. We might combine UMD with other tools and approaches, such as the Win-Win model,<sup>12</sup> to support negotiation, and goal-based requirements engineering techniques,<sup>13</sup> to support requirements elicitation. Then, we want to extend the support that UMD provides stakeholders by encompassing more experience-based capabilities. For example, once a stakeholder has identified a potential issue for a specific service, we'd like it to suggest the most appropriate measurement models as well as the most common system reactions and possible external events. 

### Acknowledgments

We acknowledge support from the NASA High Dependability Computing Program under cooperative agreement NCC-2-1298. We thank the researchers on the HDCP project for their insights and suggestions and Jennifer Dix for proofreading this article.



**Victor R. Basili** is a professor of computer science at the University of Maryland, College Park. He served as Executive Director of the Fraunhofer Center for Experimental Software Engineering-Maryland and was a founder and principal in the Software Engineering Laboratory at NASA Goddard Space Flight Center. He works on measuring, evaluating, and improving the software development process and product. He received his PhD in computer science from the University of Texas. He is a fellow of the IEEE and ACM. Contact him at Dept. of Computer Science and Inst. for Advanced Computer Studies, Univ. of Maryland, College Park, MD 20742; basili@cs.umd.edu; www.cs.umd.edu/~basili.

**Paolo Donzelli** is a visiting senior research scientist with the Computer Science Department at the University of Maryland, College Park. He's on sabbatical from the Office of the Prime Minister in Italy, where he's a director with the Department of Innovation and Technology. His research interests include software process improvement, requirements engineering, and dependability modeling and validation. He received a PhD in computer science from the University of Rome, TorVergata, Italy. Contact him at the Computer Science Dept., Univ. of Maryland, College Park, MD 20742; donzelli@cs.umd.edu.



**Sima Asgari** is a research associate in the Computer Science Department at the University of Maryland, College Park. Her research interests include the combination of theoretical and empirical software engineering, human factors, and software dependability. She received her PhD in computer science from the Tokyo Institute of Technology. Contact her at the Computer Science Dept., Univ. of Maryland, College Park, MD 20742; sima@cs.umd.edu.

### References

1. B. Boehm et al., *The Nature of Information System Dependability: A Stakeholder/Value Approach*, tech. report, Computer Science Dept., Univ. of Southern California, 2003.
2. B. Bruegge and A.H. Dutoit, *Object-Oriented Software Engineering*, Prentice Hall, 2004.
3. I. Rus et al., "Empirical Evaluation Techniques and Methods Used for Achieving and Assessing High Dependability," *Proc. Workshop Dependability Benchmarking*, 2002, www.ece.cmu.edu/~koopman/dsnwdb/wdb02\_rus.pdf.
4. B. Melhart and S. White, "Issues in Defining, Analyzing, Refining, and Specifying System Dependability Requirements," *Proc. IEEE Conf. Eng. of Computer Based Systems*, IEEE CS Press, 2000, pp. 334-340.
5. B. Randell, "Dependability—A Unifying Concept," *Proc. Workshop Computer Security, Dependability and Assurance: From Needs to Solutions*, IEEE CS Press, 1998, pp. 16-25.
6. MIL-STD-882, *System Safety Program Requirements*, US Dept. of Defense, 1993.
7. ISO/IEC 13236, *Information Technology: Quality of Service: Framework*, ISO/IEC, 1998.
8. J.-C. Laprie, *Dependability: Basic Concepts and Terminology, Dependable Computing and Fault Tolerance*, Springer-Verlag, 1992.
9. S. Asgari et al., "Empirical-Based Estimation of the Effect on Software Dependability of a Technique for Architecture Conformance Verification," *Proc. Int'l Conf. Software Eng. 2004 Workshop Architecting Dependable Systems*, LNCS 3,069, Springer-Verlag, 2004.
10. G. Dennis, *TSAFE: Building a Trusted Computing Base for Air Traffic Control Software*, masters thesis, Computer Science Dept., Massachusetts Inst. Technology, 2003.
11. N. Jog and B. Shneiderman, "Starfield Information Visualization with Interactive Smooth Zooming," *Proc. IFIP 2.6 Visual Databases Systems*, Chapman & Hall, 1995, pp. 3-14.
12. B. Boehm et al., "Using the Win-Win Spiral Model: A Case Study," *Computer*, July 1998, pp. 33-44.
13. L.K. Chung et al., *Non-Functional Requirements in Software Engineering*, Kluwer, 2000.

For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/publications/dlib](http://www.computer.org/publications/dlib).