

## Dependability

**John D. McGregor**, Clemson University and Luminary Software LLC, U.S.A.

### Abstract

We depend on software everyday in our professional and personal lives and people depend on the software we produce. By “depend” I mean that some essential activity could not be successful if the necessary software failed to perform. In this issue of Strategic Software Engineering, I will explore the importance of dependability to the strategic goals of the organization. I will consider a framework in which to make realistic architecture decisions about dependability.

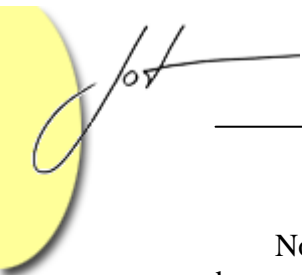
## 1 INTRODUCTION

Recently I had the honor of being invited to participate in a Dagstuhl seminar on software dependability engineering. The seminar provided the opportunity for researchers and industry representatives to come together and examine the state-of-the-art and the state-of-the-practice in building dependable software. This was particularly timely for me since I have been investigating how to build dependable software lately and was interested in exploring the issue with others.

I flew to and from the seminar in Germany in a Boeing aircraft that still has the pilot in control. I am not certain how I would have felt about flying to a seminar on dependability on a fly-by-wire aircraft! Particularly given that the standards to which avionics software is designed were last refreshed in 1992 [RTCA 92]. But, most of us depend on air travel and we accept that it is a dependable mode of transportation. Why? Because we have lots of evidence that it works and little evidence that it doesn't.

In fact, this acceptance is the essence of the standard definition of dependability. “Dependability is the trustworthiness of a computing system that allows reliance to be justifiably placed on the services it delivers.” [IFIP 90] As you can imagine, this definition allows a variety of interpretations. What is sufficient to establish trust in one environment is not sufficient in another.

“High dependability” is a term used by some to describe situations in which “health and well-being” are at stake. I view dependability on a continuum on one end of which is life critical systems and on the other are entertainment systems upon which nothing depends. For life critical systems clearly a high level of dependability is required. For the entertainment product little or no attention is given to how dependable the software is.



Not all systems require the high dependability of life critical systems but as software becomes more pervasive many of our everyday actions depend on software operating as anticipated. You need directions to your next meeting but the web site is down. After a power outage, your access to your office has been altered in the passcard system. No one will be harmed in either case but there will be a loss of productivity for the company depending on the software and loss of reputation for the companies who created the software that failed to operate properly.

## 2 THE EYE OF THE BEHOLDER

The IFIP definition of dependability opens the possibility of domain specific definitions. I am willing to depend on a word processor that has many more defects than the avionics of the Boeing 767. I gripe but I can put up with rebooting the system that runs my word processor but I get nervous when the pilot has to reboot the regional jet on which I am riding (that's happened to me twice now). Several of the participants at the Dagstuhl also take this context-dependent view [Basili 04].

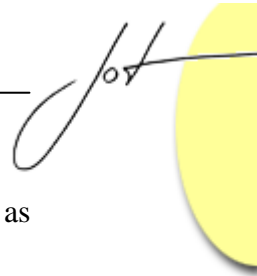
Avizienis et al [Avizienis 00] listed a set of quality attributes, shown in Table 1, that they posit as the components of dependability. This list reflects the fact that early use of the term “dependable” was limited to the life critical and safety critical segments of my continuum. This usually involved embedded systems in avionics and medical systems.

The increased automation of financial systems has expanded “dependability” to include qualities such as security. In addition to this expansion, different domains tend to have different priorities among the qualities. Some domains, such as e-commerce place a high priority on availability while domains such as e-government rank security higher than availability. Commercial organizations want satisfied customers and systems must be available for the customer to be satisfied. Government organizations handle very sensitive information and the loss of a little time is not as important as having the data be secure.

Dieter Rombach, of the Institute for Experimental Software Engineering (IESE) Fraunhofer, led a discussion at the Dagstuhl seminar on the definition of dependability. It became clear quickly that there was a wide range of definitions including my view that techniques for dependability engineering must allow the attributes to vary from one domain to another. We quickly agreed to disagree

## 3 EVIDENCE OF DEPENDABILITY

Mary Shaw, of Carnegie Mellon University, led a discussion about the types of evidence that are acceptable to the audiences who must be persuaded that software is dependable. In some cases the audience, such as government regulators, must be convinced of the dependability of the software prior to its first commercial use. In other cases the audience is the customer who commissioned the work or the buying public. They may want some



---

evidence prior to buying, such as a test data sheet, but they will continue to evaluate as they use the product.

The IFIP definition implies that two types of measurement are required. The software can be measured to determine its reliability, availability, and other attributes. The audience for specific software also need to be measured to determine what types of evidence are believable and what levels of that evidence are acceptable.

Our working group divided the types of evidence into hard and soft categories.

### **Hard evidence**

“Hard” evidence is the usual type of data that engineers might use to drive decisions. Formal proofs of correctness, use of certified code generators, statistical test results, and other tangible measurable criteria are examples. This evidence comes from explicit actions such as measuring the software during development or test executions. These types of evidence are expensive and time consuming to collect. The real problem though is it is still not clear exactly what and how to measure.

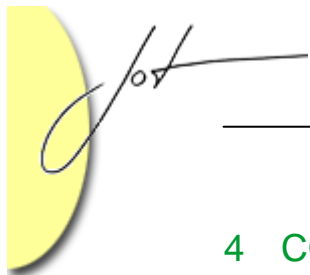
Although much work has been done on specifying software modules, there are still many issues surrounding what constitutes a good specification. One issue is exactly what the specification should include. For example, in some applications timing is critical. The specifications of components used in these applications need to provide some type of information about the timing of operations so that the timing of the composition of two components can be estimated. In other situations, information on timing is of no value but information on capacity, for example, is critical.

### **Soft evidence**

Soft evidence is less tangible, but no less valid, evidence of the trustworthiness of the software. The reputation of the vendor providing the software is one type of evidence. The maturity of the process by which the software is constructed can elicit trust in some. Using specific techniques, such as state machine modeling, may provide a level of trust in some clients.

Software architects use soft evidence in creating the architecture for dependable systems. In many cases, such as estimates of performance, the information starts out as quantitative but it is used in a “soft” manner. That is, the architect uses judgement to determine whether a proposed architecture transformation will enhance the high priority architecture qualities while degrading only low priority ones. These qualities are often measured on different scales and can not be directly combined by quantitative means.

Our current work uses qualitative modeling to represent and combine soft evidence about dependability. We use techniques developed in the 1980s’ to construct a qualitative physics as the basis of our technique [Faltings 96]. We build a qualitative model of the quality attributes and their relationships. The model can be used to predict the changes in attributes as the architecture is modified.



## 4 CONSTRUCTING DEPENDABLE SOFTWARE

At the Dagstuhl seminar Rance Cleaveland, executive director of the Fraunhofer Maryland, Inc., led a discussion about the construction of dependable systems. One issue, on which there was widespread agreement, was that dependability must be explicitly built into the product. It is just as unlikely that a system can be re-engineered late in the lifecycle to be more dependable as it is that we can re-engineer for performance.

I use the Attribute Driven Design technique developed by the Software Architecture Technology initiative at the Software Engineering Institute to develop software [Bachmann 00]. This technique uses “quality attributes” such as performance and maintainability to guide the development of the software architecture.

Dependability is usually defined as a combination of widely recognized attributes. As mentioned previously, Avizienis et al [Avizienis 00] give a list of quality attributes that defines their view of dependability. Writers in other areas have different lists.

Our investigation showed that the literature within a domain often had close agreement on what constitutes “dependable” but the definition in one domain of literature varied considerably from another domain [McGregor 07]. We also found that when different disciplines included the same attribute in their lists often the attribute had a very different priority in one domain from the other. The lists in Table 1 were derived from a survey of the published literature. The right three columns illustrate the differences among three domains.

Attributes	[Avizienis 00]	E-commerce	Middleware	Grid
AVAILABILITY	1	1	1	2
RELIABILITY	2	2	3	3
SAFETY	3		4	
CONFIDENTIALITY	4			
INTEGRITY	5			
MAINTAINABILITY	6			
SECURITY		3	2	1

Table 1 - Dependability Attributes

The most convincing, in my opinion, piece of evidence that software is dependable is the quality of the development process used to construct it. That quality is best documented by a chain of development techniques that use intentional techniques that take high quality inputs and achieve high quality outputs that possess the specific qualities that constitute dependability. One of my current interests is in creating techniques that result in an architecture that will contribute to that chain of dependability.



---

## 5 CONCLUSION

Producing software products upon which customers can, and do, depend is of strategic importance to an organization. Dependability is an attribute that is an increasingly important product quality as more users integrate our software into their essential business processes. To achieve the required level of this attribute, an organization must instill trust in their customers. The architect designing software to be dependable determines the types of evidence likely to instill trust in the software by the stakeholders.

Software that is dependable is likely to become an essential part of an organization's business process. When that happens, the organization that produced the software comes to be a trusted partner of the user of the software. This represents a strategic opportunity for the organization in the form of future versions and maintenance contracts for the current product, and ready acceptance of other products from the same company.

## ACKNOWLEDGEMENTS

Thanks to Tacksoo Im for comments that helped improve this paper and for his participation in the research underlying this paper.

## REFERENCES

- [Avizienis 00] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. *Fundamental Concepts of Dependability*
- [Bachmann 00] Felix Bachmann, Len Bass, Gary Chastek, Patrick Donohoe, and Fabio Peruzzi. The Architecture Based Design Method. Technical Report CMU/SEI-2000-TR-001, Software Engineering Institute, Pittsburgh, PA, 2000.
- [Basili 04] Victor Basili, Paolo Donzelli, and Sima Asgari. *A Unified Model of Dependability: Capturing Dependability in Context*. IEEE Software, V. 21 n. 6, pp. 19 – 25, November 2004.
- [Faltings 96] Boi Faltings and Peter Struss. *Recent Advances in Qualitative Physics*, The MIT Press, 1996.
- [IFIP 90] IFIP WG 10.4. *Dependability: Basic Concepts and Terminology*. IFIP Working Group on Dependable Computing and Fault Tolerance, October 1990.
- [McGregor 07] John D. McGregor and Tacksoo Im. *A Qualitative Approach to Dependability Engineering*, Proceedings of Dahstuhl Seminar #07031, January 2007.
- [RTCA 92] Radio Technical Commission for Aeronautics. DO-178B, 1992.

## About the author

Dr. John D. McGregor is an associate professor of computer science at Clemson University and a partner in Luminary Software, a software engineering consulting firm. His research interests include software product lines and component-base software engineering. His latest book is *A Practical Guide to Testing Object-Oriented Software* (Addison-Wesley 2001). Contact him at [johnmc@lumsoft.com](mailto:johnmc@lumsoft.com).