

Foundational Theory of Software Component Reliability

Dick Hamlet
Portland State University
hamlet@cs.pdx.edu

Dave Mason Denise Woit
Ryerson Polytechnic
{dmason,dwoit}@scs.ryerson.ca

Promise of Software Components

In engineering design, the idea of aggregating standardized components to create a complex system has allowed engineers to create better systems more easily. Components are described in a handbook, where each has a “data sheet” entry. Its data sheet describes what a component does, and equally important, it gives constraints that allow the system designer to decide if the component is “good enough” for the application. For mechanical components, these constraints concern, for example, the life expectancy of the component.

The success of the component-construction paradigm in mechanical and electrical engineering has led to calls for its adoption in software design. Software is embedded in systems with mechanical and electrical components, systems designed using component techniques from these other branches of engineering. The system designer of an embedded system would like the software “component” to have a data sheet.

Analogies between disciplines are only suggestive. No one knows whether software components can realize the benefits of the analogous mechanical and electrical ones. Certainly the embedded-system designer who today looks for technical data sheets on software parts will be disappointed. Without the solid information of a data sheet, software “components” are no bargain. To buy off-the-shelf software of unknown quality is only to trade the difficult task of assessing your own work, for the more difficult task of assessing someone else’s.

Component Data Sheets

We propose an underlying theory for the technical quality information to appear on a software-component data sheet. The other side of the components’ coin – the technical description of what a component is supposed to do – is of equal or greater importance, but we do not address it. (However, most of the current research in reusability goes into component specification, not component quality.)

To be useful in system design, component data sheets must contain technical information sufficient for the system designer to make reliability calculations; and, it must be possible for the component developer to collect this information at a reasonable cost. Subjective assessments derived from the software development process do not qualify as data-sheet information, because they amount to no more than assertions that the developer is trying to do a good job. For example, stating that the developer uses a careful inspection process is no help in calculating reliability. The reputation of the developer is not an inconsequential factor in selecting a com-

ponent; but, its role should be to convince the designer that technical data can be believed.

Two measures of software quality that qualify for a data sheet are (1) that the component has been proved correct, or (2) that it has been randomly tested using an accurate user profile. Both of these require a formal specification of what the component is supposed to do (the part of the data sheet that we do not consider). A statement that the component has been proved correct is a guarantee that it will perform according to its specification. A statement that (say) its reliability is better than $1 - 10^{-4}$ per execution with an upper confidence bound of 99%, is a statistical assertion that there is no more than 1 chance in 100 that it will not perform according to the specification in 10,000 trials. Correctness proofs (1) can be thought of as the special case of a profile-independent reliability of 1.0, with 100% confidence. The reader of a data sheet might doubt that the developer has actually established such precise technical claims. But this is a question that can be answered scientifically, and if the developer has lied there are legal remedies.

When the quality information on a component’s data sheet is statistical, it must be obtained by random testing. The fundamental problem of assessing component quality is that any standard profile from which test inputs are drawn will *not* match the profile that the component will experience when placed in a system.

We solve the profile problem with data-sheet mappings based on a subdomain decomposition of the component input domain. A system designer can use these maps to predict the system reliability before implementation. Our procedure is not necessarily practical (although we hope to show that it is practical enough to be often useful); but, we maintain that it is a fundamental theory of component reliability, on which any practical work must be grounded.

Data-sheet Mappings

A component developer provides a data sheet giving a set of subdomains, and two mappings of an input profile. A profile is assumed to be expressed as a weighting over the subdomains. Let the subdomains S_i partition the component input domain $D = S_1 \cup S_2 \cup \dots \cup S_n$. A profile P is then a vector of weights to be assigned to each S_i :

$$P = \langle h_1, h_2, \dots, h_n \rangle .$$

The data-sheet mappings are:

Reliability mapping. Carries a profile vector to a real value $R \in [0, 1]$, the probability that the component will not

fail on an input drawn according to this profile. The developer measures the failure rates f_i within each subdomain using random testing. Then

$$R = \sum_{i=1}^n h_i f_i.$$

Profile-transformation mapping. Carries an input profile vector to an output profile, the latter expressed as a weighting vector over an arbitrary set of subdomains U_1, U_2, \dots, U_m (unrelated to the subdomains on the data sheet). The weightings of the output profile

$$Q = \langle k_1, k_2, \dots, k_m \rangle$$

on these subdomains is the sum of the contributions from each input subdomain,

$$k_j = \sum_{i=1}^n h_i \frac{|\{z \in S_i | c(z) \in U_j\}|}{|S_i|},$$

where c is the function computed by the component. The information on the data sheet and the ability to execute the component to calculate c , are sufficient to estimate the k_i given U_i , by making a random selection from each S_i .

The data-sheet mappings allow a system designer to calculate using an arbitrary profile (given in terms of a vector of subdomain weights) for a component. The profile-transformation describes how a profile is altered across a component, and the reliability mapping gives a component's independent contribution to the system reliability.

System Design and Reliability

The example of a system design and reliability calculation using two software components A and B in sequence will illustrate our ideas. The components must be functional, that is, must not communicate through global state. A is given the system input, and invokes B , passing its output as B 's input; B 's output is the system output. A must *invoke* rather than *use* B , in the sense of Parnas [buzz]. The system developer has a profile vector for the system, and the data sheets for A and B . Using the system profile (which is input to A), A 's reliability mapping can be used to calculate R_A , the reliability of A alone. (The subdomains of the system are projected onto the subdomains of A 's data sheet.) A 's profile-transformation mapping can be used with the subdomains from B 's data sheet as the U_i , to calculate the profile B will see. Then B 's reliability mapping can be used to calculate reliability R_B for B alone. The system reliability is finally calculated as $R_A R_B$.

Summary and Future Work

Briefly, the theory we are developing includes:

- Reliability calculation for system-design structures other than component sequence, in particular, conditionals and loops. Thus we have a complete algebra for system design.
- Treatment of "system glue," the explicit system code between and around components. We divide this code into component-like fragments that are functional and successively invoke each other.
- Consideration of component independence.

The ideas on which the theory draws have appeared in references [RST], [HASE], and [SRE].

Of the remaining open questions, perhaps the two most important are:

- How should component subdomains be chosen? In particular, subdomains that are too large may hide failures from the component developer's tests.
- How can software systems employ components so that the reliability of the whole is better than that of its parts? For safety-critical systems, a design technique must be found analogous to parallel redundant subsystems in mechanical engineering. Multiversion programming (MVP) is the only known technique, but in it the effect of coincident failures cannot be quantitatively predicted.

The theory we have developed is "microscopic" rather than "macroscopic." We show how to calculate system properties from the data sheets of components in full detail. If this procedure is to be used in practice, it must be supported by tools, for example to automate the transformation of profiles and domains through sequences of components.

References

- [Buzz] David Parnas, On a 'buzzword': hierarchical structure, *Proc. IFIP Congress '74*, North Holland, 1974, pp 336-339.
- [HASE] Denise Voit and Dave Mason, Software component independence, *Proc. 3rd IEEE High-Assurance Systems Engineering Symposium (HASE '98)*, Nov., 1998.
- [RST] Dick Hamlet, Software Component Dependability, a Subdomain-based Theory, Technical Report RSTR-96-999-01, Reliable Software Technologies, Sterling, VA, September, 1996.
- [SRE] Denise Voit and Dave Mason, Software system reliability from component reliability, *Proc. of 1998 Workshop on Software Reliability Engineering (SRE '98)*, July, 1998.