

# Extending the Traditional Operational Profile Model

Mechelle Gittens  
University of Waterloo

Hanan Lutfiyya, Mike Bauer  
University of Western Ontario

Curtis Gittens  
SIASQA Corp.

## Introduction

Software reliability is the ability of a software system or component to perform its required functions under stated conditions for a specified period of time [stan91]. It is measured in the system testing phase of the software development life cycle with an approach that uses the *operational profile model* as defined by Musa to place the software of interest in a testing circumstance that is *typical* of the circumstance in which it is expected to operate [Mus03]. The operational profile is the set of operations or processes for a software application, and the probabilities of occurrence for those operations or processes [Mus99]. In [Rom03], Jorge Romeu of the Reliability Analysis Centre of the Department of Defense mentions the success of the profiling for the validation of hardware and theorizes that this is because hardware components often work in a specific homogeneous environment. This homogeneity may be due to enforced standards. In contrast, the software operating environment is often extremely heterogeneous [Rom03]. Consequently, hardware reliability has seen more success than software reliability. In this short paper we introduce an enhanced version of Musa's traditional operational profile model that addresses the heterogeneity of the software environment.

## The heterogeneity of software

The authors of [Whit00] note that usage is a central concern when testing hardware. The warranties for hardware items state that unintended use, which results in more than normal wear and tear, will render any warranties void. For example, the warranty for the GE Profile 21.7 Cu. Ft. Top Freezer Refrigerator states that it applies only for "single family domestic use" in Canada when the refrigerator has been "properly installed according to the instructions supplied by Camco" [man03]. If this product is plugged into a car battery, the warranty is voided, because its operational environment would have been incorrect. This example demonstrates how easily a hardware profile can be defined to take into account its operational environment. However, how would an operational profile of a word processor (for example) be defined to account for its operating environment [Whit00]?

There are many operations that such a program can execute as its functionality. These operations were developed and tested, and thus expected to behave in a particular manner. There are however several variances in the environment in which that program exists, or even in the configuration of the program itself that will cause what appeared at the development organization's site to be a correctly functioning program to go awry. A few potentially problematic complexities within

the operating environment of the word processor include: 1) The operating system refuses the word processor additional memory; 2) A document being requested by one user has been marked 'read-only' by another *privileged* user; 3) The backup feature for documents in use tries to write to a bad disk sector.

In all of these cases, knowledge of the operations and the probabilities of usage, which make up the traditional operational profile model, do not directly allow for the testing of such scenarios [Mus99]. It is our experience in industrial practice with a large commercial system that issues like the first scenario above, are contributed to by the operating environment in which the software exists [Git02]. The operating system enforces the limits on memory and makes decisions based on the requirements of the other applications in the operating environment [Voas00, Whit00]. The nature of the data structures used by the software and how those structures are configured also contribute to memory usage. Additionally, the amount and type of data being processed by the program is also of concern, as the data may require additional memory. Test cases that only deal with the operations and discount the data that is used by the operations are then clearly insufficient. These factors also apply to the second and third scenarios above. Therefore, the system operational profile for the software should also contain measured information about the typical data used with the software, as well as the used data structures.

As indicated in [Voas99, Voas00, Whit00], in addition to the most used features, the operational profile must also include information about the operating environment; information about other applications in the operating environment; and external data that is used by the application. Furthermore, based on [Whit00], information on the configuration of the application itself is also essential to a potentially pervasive operational profile specification.

## An extension of the operational profile model

Our research in [Git02] and the work of many of the previously mentioned authors, has shown that a new operational profile model must include several measurements of the configurations of the software, as well as measurements of the variety of inputs and operating environments. As a result we have developed the *Extended Operational Profile* (EOP) model. This new model is more representative of software operational environment than the traditional operational profile model. The EOP model is composed of three parts:

- The **processes** and the frequencies with they occur in a typi-

cal customer application typical of the current operational profile approach.

- The **structure** of the system on which the application is running, and the configuration or structure of the actual application that is running.
- The values of the inputs to the application from one or more customer. This is the **data** component of a profiled application.

It may be true that all of these components of profiling may not always appear to be relevant to all applications. For example, a single version of a word processor may only have one valid configuration. However the varying versions of that word processor, if it comes as part of a product line, as most modern software does, will invariably have varying configurations. As a result, the vendor of this product should be aware of the system operational profile for the existing incarnations of the product, as customers do not always behave as hoped, and may try to use a newer version in a manner in which they used a past version. If this behaviour is common enough, it is important.

### The contributions

The new EOP model provides statistical representation of supplementary production information for a software application in the categories of process, structure and data. The EOP allows an analyst to use measurements of the characteristics of the environment and configuration of a software application in use. These measurements supplement Musa's model, which captures the operations or functionality of an application and frequencies of occurrence for those operations. We have also derived a Internet based software tool that provides much needed automation to support the use of the EOP. The tool guides a user through the development of an extended operational profile for their organization's application.

The tool allows an analyst to compare the extended operational profile of an in-house test instance of an application with similar information for the application being used in the field by a customer. This is facilitated to determine if the test system captures the customer's use of the application. The demonstration of the new model using the tool is done with an industrial case study from a large scale production system. Finally we have also constructed a development process in which the EOP should be used, in order to guide the user within their own software development life cycle.

### Concluding remarks

There are several results from the new model. One of note for this short paper is the identification of characteristics of the customer's data structures that are different to, and require representation in, the test environment. We were able to link

these characteristics to specific customer questions. We also gained understanding of the statistical properties of data structures. Specifically, we realized that measurements of the properties of the data structures are discrete. This means that if one wishes to compare structural properties of a test instance and a customer instance, a robust method that makes no assumptions of data distribution must be used. The method that is used in the new model is the quantile-quantile plot.

## References

- [stan91] IEEE. *IEEE Standard 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology*, February 1991.
- [Mus99] John Musa. *Software Reliability Engineering*. McGraw-Hill, New York. 1999.
- [Mus03] John D. Musa. *More Reliable Software Faster and Cheaper (Software Reliability Engineering)*. Current August 2003. [members.aol.com/JohnDMusa/index.htm](http://members.aol.com/JohnDMusa/index.htm)
- [Wey00] Elaine J. Weyuker and Fillipos I. Vokolos. Experience with Performance Testing of Software Systems: Issues, and Approach, and Case Study. *IEEE Transactions on Software Engineering*, Vol. 26, No. 12. December 2000.
- [Rom03] Jorge Romeu. *A discussion of software reliability modelling*. Current August 20003. [rac.iitri.org/pdf/1ST\\_Q2000.pdf](http://rac.iitri.org/pdf/1ST_Q2000.pdf)
- [Whit00] James A. Whittaker and Jeffrey Voas. Toward a More Reliable Theory of Software Reliability. *IEEE Computer Vol. 33, No. 12*. December 2000.
- [man03] General Electric. *Use and Care Manual or Warranty - GE Profile Arctica 21.7 Cu. Ft. Top Freezer Refrigerator*. Current August 2003. [www.GEAppliances.com](http://www.GEAppliances.com)
- [Git02] Mechelle Gittens, Hanan Lutfiyya, David Godwin, Michael Bauer, Yong Woo Kim and Pramod Gupta. An Empirical Evaluation of System and Regression Testing. *Proceedings of CASCON 2002*. IBM 2002
- [Voas00] Jeffrey Voas. Will the Real Operational Profile Please Stand Up. *IEEE Software Vol. 17, No. 2*. March/April 2000
- [Voas99] Jeffrey Voas. Certifying Software for High-Assurance Environments. *IEEE Software Vol. 16, No. 4*. July/August 1999