December 1997

A Case Study Of The Reliability Of Business Information Systems

Paul Bowen
University of Queensland

Jon Heales
Australian National University

Monthira Vongphakdi
The University of Queensland

A Case Study Of The Reliability Of Business Information Systems

Paul L. Bowen, Ph.D., CPA
Lecturer
Department of Commerce
University of Queensland
Brisbane, Queensland 4072, Australia
Phone: +61 7 3365 6584; Fax: +61 7 3365 6788
Internet: bowen@lorien.commerce.uq.edu.au

Jon Heales
Lecturer
Department of Commerce
Australian National University
Canberra, ACT 0200, Australia
Phone: +61 6 6249 5478; Fax: +61 6 6249 5005
Internet: Jon.Heales@anu.edu.au

Monthira T. Vongphakdi
Department of Commerce
The University of Queensland
Brisbane, Queensland 4072
Australia

The authors would like to thank Jim Hann, Fiona Rhode, Ralph Saunders, John Seymour, and Ron Weber for their assistance in making collection of the research data possible and for their comments on previous drafts of this paper.

Executive Summary
Software is playing an ever-increasing role in business processes as computer applications become more diverse and permeate more areas of organisations. As organisations use these information systems, not only for fundamental transaction processing but also to make operational, managerial, and strategic decisions, the reliability of these systems directly affects these organisations viability.

To date, most empirical investigations of software reliability have involved systems with well-defined, stable specifications or for safety-critical software. Many business-oriented software applications, however, are subject to frequent changes in requirements. End-users who use business-oriented systems and who specify the requirements of these systems are typically more people oriented and less technically oriented than users who specify the requirements of compilers, radar tracking systems, or medical equipment.

This research develops a software reliability model for business oriented software. The model portrays the total reliability of an information system as the product of the reliability of the mapping from end-user requirements to the specifications and the reliability of the mapping from the specifications to the software itself. A case study in a large government organisation was used to provide initial empirical evidence about the model. Results indicate that, in business environments, the reliability of the mapping from end-user requirements to the specifications is more problematic than the reliability of the mapping from the specifications to the software itself. In particular, the results show that as user requirements become more volatile, the reliability of the software decreases. Use of the software quality metrics illustrated in this paper can lead to better software management, lower maintenance costs, and improved software reliability.

1. Introduction

Software is playing an ever-increasing role in critical business processes (Musa et al., 1990:4; Lyu, 1989:80) as computer applications become more diverse and permeate more areas of organisations. The rapid evolution and the spread of information system technology increases the demand for reliability because imperfect or faulty software can have dramatic consequences on organisations. For instance, unreliable software can result in inaccurate or inconsistent updates of data. These data errors can cause erroneous management decisions and result in financial losses. Similarly, erroneous software that miscalculates customers' interest can threaten banks' business and their reputations. This paper reports the application of statistical reliability testing to business software reliability.

Software reliability measurements are often taken for two types of software. First, for software that has well-defined, stable specifications, whose continuous operation over an extended period of time is important. These systems must be extremely reliable because they are widely used by a large number of people and provide the basis of information systems infrastructure (Yamamoto and Yamagishi, 1994). Such software includes compilers and other system software (see, e.g., Khoshgoftaar, Szabo, and Woodcock, 1994; Wordsworth, 1991). Second, reliability measurements are made for safety-critical software. These systems cause major concerns for the software engineering community because their failure could be life-threatening or result in extensive damage (Parnas et al.,1990; Leverson, 1991). Such software includes applications used in military and civilian aircraft, in nuclear plants, and in medical devices (see, e.g., Rushby and von Henke, 1993; May, Hughes, and Lunn, 1995; Gerhart, Craigen, and Ralston, 1994a,b). For both systems infrastructure and safety-critical software, the requirements typically are well defined when the project is initiated and usually remain stable throughout the development and operational phases.

Software reliability is an important issue (Daniels and Hughes, 1985; Lyu, 1989; Rook, 1990), however, there is little literature which reports on the reliability of business applications. One reason for this deficiency relates to the cost and time involved in the reliability measuring process (Rook, 1988; Musa et al., 1990:4). Another reason is that the evolving nature of business environment changes the information needed (Oei et al., 1994). Software developers often find it difficult to respond to all change requests rapidly and accurately (Banker et al., 1994) because change requests cause software specifications to change. Consequently, software developers have difficulty ensuring that the software they produce retains high levels of reliability.

Reliability of any software depends on how well the software functions meet user requirements, i.e., on its operational reliability (Littlewood, 1980). Software reliability depends on (1) the correlation between user requirements and software specifications and (2) the correlation between these specifications and the source code. If software is developed from specifications that correctly reflect user requirements, then software developers can use techniques such as structured programming (Linger, Mills, and Witt, 1979), formal specifications and correctness proofs (Reade and Fromme, 1990), and rigorous software testing (Shooman, 1989) to ensure that their source code is reliable (Bendell and Samson, 1985; Sommerville, 1992; Daniels and Hughes, 1979).

Various factors, however, affect how user requirements are transformed into software specifications. Obtaining user requirements involves interactions between users and IS professionals. The accuracy of these requirements depends on contextual conditions. Depending on the nature of the new system, its effect on users, and the relative power of users and system developers, the process of obtaining user requirements may vary from highly rational and co-operative to an acutely political and acrimonious process (Newman and Sabherwal, 1991).

To date, most statistical reliability measures reported have involved operating systems, compilers, or other software with relatively stable requirements. This study develops an explanatory model of the reliability of business application systems and statistically measures the reliability of business software.

2. Software Reliability Model

Software reliability measures quantify how well software meets the requirements of the users (Kokol et al., 1991:173; Musa et al., 1990:5; Siefert, 1991:86). Currently, software reliability is addressed in two ways: from a technical perspective and from an end-user perspective (Siefert, 1991).

The technical perspective deals with reliability from the developer's point of view (Musa et al., 1990;

Banker et al., 1994). From the technical perspective, software reliability means ensuring that coding errors do not occur. Coding errors occur because specifications were implemented incorrectly (Weber, 1988:668). This perspective assumes that specifications are complete and correct (Kokol et al., 1991). The technical perspective stresses the reliability of the mapping from specifications to implemented software. Developers measure software reliability by comparing its operation against the specifications and counting the faults or defects detected during testing.

The end-user perspective of software reliability considers how well the implemented software corresponds to actual current user requirements or expectations (Musa et al., 1990:5). If software fails to meet user expectations, dissatisfaction generally develops whether these failures result from coding errors or from specification deficiencies. Specification deficiencies also arise from omitted, misstated, poorly articulated, or assumed user requirements.

To integrate the technical and end-user perspectives, Kokol et al. (1991) suggested that, during the design phase, the specifications must be prepared according to the user requirements. In addition, software developers should implement all parts of the specifications and ensure that the implemented system represents a correct mapping from the specifications. Kokol et al. (1991) stated three propositions. First, during the requirement specification phase, the analysts must understand the user requirements and the environment in which software will operate. The software specifications must correspond to user requirements. Second, during the implementation phase, all parts of the specification should be correctly implemented. Third, it must be possible to validate the implemented system.

Using Kokol et al.'s perspective, let Ω represent the overall mapping from user requirements to the implemented software and $\mathbf{R}(\Omega)$ represent the reliability of this mapping. Also, let Ψ represent the mapping from user requirements to specifications and Φ represent the mapping from specifications to implemented software. The total reliability of software, $\mathbf{R}(\Omega)$, depends on the reliability of the mapping from user requirements to specifications, ($\mathbf{R}(\Psi)$), and the reliability of the mapping from specifications to implemented system, ($\mathbf{R}(\Phi)$) (see Figure 1). Too often, the reliability of the mapping from specifications to implemented software (reflected by $\mathbf{R}(\Phi)$) and the reliability of overall mapping from user needs to implemented software (reflected by $\mathbf{R}(\Omega)$) are considered independently.

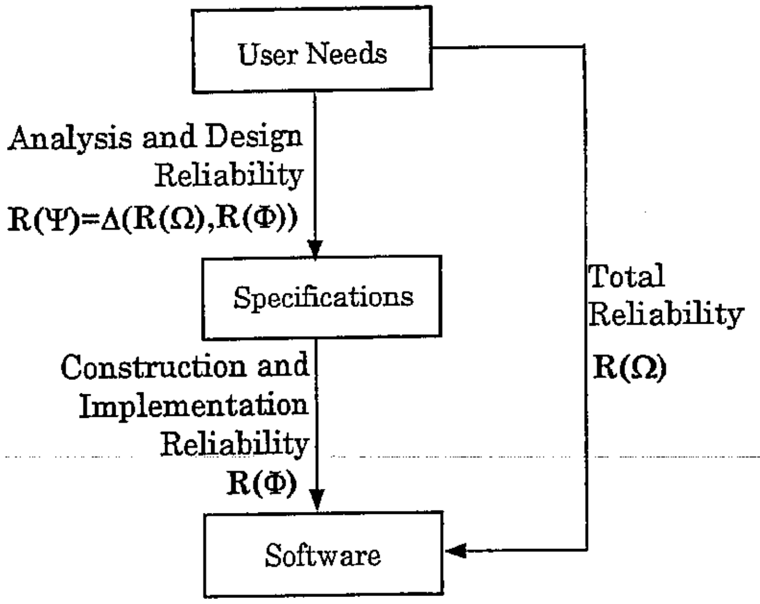
User Needs
Analysis and Design Reliability
R(Ψ)=Δ(R(Ω),R(Φ))
Specifications
Construction and Implementation Reliability
R(Φ)
Software
Total Reliability
R(Ω)

Figure 1. *Proposed Software Reliability Model*[1]

[1] Adapted from Lehman's two leg software process model from Kokol et al. (1991).

The reliability of the mapping from requirements to specifications (reflected by $R(\Psi)$) has been difficult to measure because the quality of this mapping depends on two contextual factors: perceived threats to users and the relative power of IS and users. When users perceive the system to be a threat, they may resist system development by withholding information during requirements elicitation (Newman and Sabherwal, 1991; Keen 1981). Developers, on the other hand, may focus unduly on a new technique or environment to the detriment of delivering a system that meets user requirements in the most effective manner.

The proposed reliability model is based on Lehman's "two leg" software process model (see Figure 1) presented by Kokol et al. (1991). Lehman's "two leg" software process model summarises the software development process into two phases: specification phase and implementation phase. Requirements are transformed into specifications in the former phase and specifications are transformed into the implemented system in the latter phase.

The Kokol et al.(1991) propositions imply that the reliability of software depends on the correctness of the mapping from user requirements to specifications *and* on the correctness of the mapping from specifications to the implemented software. That is, with reliability ranging from 0 to 1:

$$R(\Omega) = R(\Psi)*R(\Phi) \qquad \text{[Equation 1]}$$

Assessing the reliability of software, therefore, involves investigating both the specification phase and the implementation phase. During the specification phase, developers must ensure that the specifications correctly reflect users needs. During the implementation phase they must ensure that the software is implemented according to the specifications.

From Figure 1 and Equation 1, $R(\Omega)$ represents the total reliability of the software. $R(\Omega)$ depends on the reliability of the mapping from user needs to the specifications $R(\Psi)$ *and* the reliability of the mapping from specifications to the software ($R(\Phi)$). That is, for the software to be reliable, user needs must be transformed accurately and completely into the software specifications and these specifications must, in turn, reflected accurately and completely by the resulting software.

End-users focus their concern on how well the software meets their needs. That is, end-users concentrate on overall software reliability, $R(\Omega)$, and, generally, have little interest in whether flaws were caused by analysis and design problems, $R(\Psi)$, or construction and implementation problems, $R(\Phi)$. Management seeks to improve the overall reliability of the software and ensure that its reliability remains high over time. Because management needs to know where to focus software quality effort, in addition to overall software reliability, $R(\Omega)$, they are also interested in the behavior and mix of the reliability components, $R(\Psi)$ and $R(\Phi)$.

3. Hypotheses

Owing to the dynamic behaviour of their environments, organisations are confronting rapidly changing information needs. As information becomes integrated more into business processes, producing reliable information systems becomes more and more crucial (Oei et al., 1994:213). In business environments, users often find it difficult to initially define their software requirements because they, especially novice users, have an insufficient understanding of the entire system and its interfaces with other systems. Therefore, user requirements are subject to change as the users' understanding of their own functions and of related functions develops. For those businesses that evolve rapidly, defining software requirements becomes even more difficult because software requirements need to be continually adjusted to suit the business circumstances. When user requirements change, the reliability of mapping from user requirements to specifications, $R(\Psi)$, suffers. Frequent business changes make it difficult to ensure that the specifications accurately reflect user requirements. The reliability of mapping from specifications to the implemented system, $R(\Phi)$, may not be affected, however, because software was implemented according to the original requirements. Hence:

Hypothesis 1: In business environments, the reliability of the mapping from specifications to implemented software, $R(\Phi)$, is higher than the reliability of the mapping from user requirements to specifications, $R(\Psi)$,.

If user requirements change frequently, maintaining software reliability becomes even more difficult. Systems which undergo frequent modifications may have higher error rates because each modification represents an opportunity for new errors to be introduced (Banker et al., 1994). When software undergoes frequent changes, developers have less opportunity and less motivation to test those changes thoroughly. Consequently, the reliability of the software suffers. Therefore, as an application experiences frequent change requests, its overall reliability decreases. That is:

Hypothesis 2a: Increases in the volatility of user requirements are negatively associated with overall software reliability, $R(\Omega)$.

Maintaining a reliable mapping from user requirements to specifications, $R(\Psi)$, becomes difficult as user requirements change frequently. Because the volatility of user requirements directly affects the reliability of mapping from user requirements to specifications, $R(\Psi)$, decreases in overall software reliability are more likely to be caused by $R(\Psi)$ rather than $R(\Phi)$. Therefore:

Hypothesis 2b: Decreases in overall software reliability, $R(\Omega)$, associated with increases in the volatility of user requirements are caused primarily by problems with Ψ, i.e.,$R(\Omega)/R(\Phi)$ decreases as the volatility of user requirements increases.

4. Research Method

To study the reliability of business application software, a single case design with multiple embedded units of analysis was used[2]. This single case followed Yin (1993) suggestion in that "a case study research is appropriate when the researcher desires to (a) define the topic broadly, (b) cover contextual conditions and not just the phenomenon of the study, and (c) rely on multiple sources of evidence". The research design is a multiple embedded design because three application systems were examined.

A metropolitan government agency agreed to participate in this research. Three application systems that differ in terms of the volatility of user requirements were examined. These applications were selected based on the recommendation of the EDP application manager. Application I was developed in a HP/Sybase[3] environment. Applications II and III were developed in a ICL/IDMS[4] environment.

Data collection methods included reviewing specification documents and user manuals, performing statistical tests on selected applications, and interviewing end-users. End-user interviews addressed their perceptions of the reliability of each application they use on a regular basis. A questionnaire, measuring other factors affecting software reliability, was administered at the end of the interviews with the end users. Separate questionnaires were also distributed to the analysts and to the application manager.

5. Constructs Measurement

Statistical software testing, also referred to as statistical usage testing, software functional testing, or random testing, was used to calculate the reliability measures. As much as possible given the organizational constraints associated with the case study, the data sampling and collection procedures followed those recommended for Cleanroom Software Engineering (Dyer, 1992; Linger, 1994; Walton, Poore, and Trammell, 1995). The reliability measurements correspond to the measure proposed by Cheung (1980).

---

2 The types of case research designs are (1) single case design with a single unit of analysis, (2) multiple case design with a single unit of analysis, (3) single case design with multiple (embedded) units of analysis, and (4) multiple case design with multiple (embedded) units of analysis.

3 Hewlett-Packard 9000 series equipment was specially installed in 1988/89 for Application I. Application programs are written in C and embedded SQL under the Sybase Relational Database Management System.

4 An ICL Super Dual 2958 mainframe was installed in 1984 and later upgraded to a 3980 model in 1986. An Integrated Database Management System (IDMS) based on the network data model is used under the Virtual Machine Engine (VME) operating system. Application programs are mainly written in Cobol.

5.1 *Reliability of the mapping from specifications to software $R(\Phi)$.*

Statistical software testing was applied to measure the technical reliability, **R(Φ)**, of each software application. One of the researchers studied specifications documents and user manuals to gain an understanding of the applications. Because the size of some applications, especially the second application, was too large and too complex to test completely, representative sections of the systems were selected for statistical testing. Because statistical testing should reflect the expected usage pattern of the software, monthly transaction reports of the production systems were examined and used to determine the distribution of transactions to test.

In this research, reliability was defined to be the probability of a software system's operational success. To measure the technical reliability, each field on each screen was tested according to specification documents[5], i.e., a sample of possible values was tested to see whether the software behaviour matched the specifications. For example, data values of a particular field may need to be master file checked. Test cases for this field included both valid and invalid codes. Correct software responses were counted as a 'successful' test and 'unsuccessful' otherwise. The estimated reliability of the mapping from specifications to implemented software was calculated as

$$\hat{R}(\Omega) = S/N$$

where ***S*** is the total number of 'successful' test values and ***N*** is the total number of test values.

5.2 *End User Reliability of the software $R(\Omega)$*

Interviews with end-users were used to measure the overall reliability of the software. Each end-user interview started with an open-ended question about the functionality of the application under study. Users were asked to demonstrate how they usually operate the application, what they liked or disliked about the system, what was working or not working, etc. For each screen that they encountered, users were asked to assess each data field on the screen. Three end-users for application I and II were interviewed. Five users for application III were interviewed, however, two interviews were discarded because these users were reluctant to divulge their opinion about the system. Each interview lasted from half an hour to two and a half hours.

Because users' perceptions are the focus of the end-user reliability, if a particular field does not satisfy the users' expectations in some way then that field is a failure. For example, a particular field may be positioned wrongly on the screen, or it may display inaccurate information. The estimated value of **R(Ω)** was calculated as

$$\hat{R}(\Omega) = S'/N'$$

where S' is the total number of successful data fields and N' is the total number of data fields evaluated by the user[6].

5.3 *Reliability of the mapping from user needs to specifications $R(\Psi)$*

$\hat{R}(\Psi)$ can be derived from $\hat{R}(\Omega)$ and $\hat{R}(\Phi)$. That is,

$$\hat{R}(\Psi) = \hat{R}(\Omega) / \hat{R}(\Phi)$$

5 The specification document for the third application was not available because the application was developed in conjunction with other governmental bodies. To understand the functionality of this application, the researcher was briefed by the responsible programmer and a training officer. Each briefing took one and a half hours. Consequently, statistical testing was based on this understanding and the user manual.

6 The reliability of business software includes the ability of software to detect and prevent erroneous data from being entered and stored in the system. When erroneous data are entered, the reliability of the entire system suffers because of the inaccurate and inconsistent data the system will produce. Therefore, the quality of input controls that govern the data entering the system play an important role in determining the reliability of an information system. In this research, the measurement of technical reliability of the application systems is based on the reliability of input controls.

5.4 *Volatility of Requirements*

Volatility of requirements was measured by using questions with 7-point scales. Volatility of requirements refers to the frequency of changes to user requirements. The applications used in this study were selected to differ in terms of the volatility of the requirements. Before the study commenced, the research was discussed with the application manager who then suggested which applications would fit this requirement. Three questions in the questionnaire were used to measure the extent of the volatility of requirements.

6. Research Results

Application I is a computer aided despatching system that responds to police emergency phone calls. The system has two major functions: first, it records information about reported incidents and second, it allocates resources[7] to these incidents. The application was developed by the organisation's IS department in conjunction with two governmental bodies and a telecommunications company. The application went live in September 1990.

Among the three applications, application I is the most critical and the most sophisticated system. All data entry and resource allocations are performed centrally by trained users working in one department. Due to the complexity of this application, end-users must attend a six week specialised training before they can use the system. Only trained users can access application I.

Application II is the largest of the three applications. The system integrates information from four disjoint databases[8] and records information about offences. The application was developed by a team of IS staff and user representatives. Application II was completed and went live in November 1994, however, program modifications were still being made to the system at the time of this study.

All data entry is performed centrally by data entry clerks who were trained to use the application. This training course requires four weeks. Other end-users with access to this application can only perform queries.

Application III is used to record traffic incidents. The system serves as an electronic copy of the original paper-based forms. The requirements of application III were dictated by another government office that maintains traffic accident statistics. After the initial implementation, the format of source document was reviewed and some major changes were made to improve the quality of information recorded. The majority of the organisation's staff uses this application. All users with access to the application can enter, modify, and query data.

Development of application III was initiated and went live in 1991. During development, the system's analyst in conjunction with the business analyst performed the requirements analysis and wrote the specifications.

6.1 *Software Reliability Measurements*

The values for the reliability measures used to test the research hypotheses are presented in the following sub-sections. Section 5.2.1 reports the reliability measure of the mapping from specifications to implemented software ($R(\Phi)$). Section 5.2.2 reports the reliability measure of the overall mapping from user requirements to implemented software $R(\Omega)$. The derived reliability measures of $R(\Psi)$ are presented in section 5.2.3.

The technical reliability, $R(\Phi)$, of each of the three systems was measured by using test cases[9]. For each system, the estimated reliability of mapping from specifications to implemented software,

[7] Resources include patrol vehicles, patrol officers, and other uniformed officers.

[8] At the time during which the study was undertaken, the participating organisation did not have a corporate data model, therefore, all databases are separate and disjoint.

[9] Due to the size of the applications (especially applications II and III) and the amount of time available, screens were randomly selected for testing.

$\hat{R}(\Phi)$, was determined by using population proportions[10]. Results from these tests are presented in Table 1.

Table 1

Results from software testing, $\hat{R}(\Phi)$.

Application	Total fields tested	Unsuccessful fields	Ratio	$\hat{R}(\Phi)$
I	268	6	262/268	0.9776
II	242	6	236/242	0.9752
III	220	7	213/220	0.9682

Overall software reliability, R(Ω) was measured using end user evaluations. The users were asked to assess and comment on the reliability of fields on the screens they use[11] without considering the response time of the system. Each interview took from half an hour to two hours. Table 2 summarises results from the end-user evaluations.

Table 2

Results from End User Evaluation, $\hat{R}(\Omega)$.

Application	Results			Total	$\hat{R}(\Omega)$
	User 1	User 2	User 3	$\hat{R}(\Omega)$	
I	70/77	51/55	53/56	174/188	0.9255
II	105/116	121/134	121/134	347/384	0.9036
III	127/139	85/87	121/124	333/350	0.9514

The reliability of mapping from user requirements to specifications, R(Ψ), for each system, is derived using measures of the three reliability estimators, i.e., $\hat{R}(\Phi)$, $\hat{R}(\Omega)$, and $\hat{R}(\Psi)$ can be derived from and.. Eighty percent confidence intervals[12,13] were calculated for each system. The results are summarised in Table 3.

---

[10] Proportion is the relative frequency of the occurrence of an event (Freund, 1979: 297). In this case, an event is a 'failure'. Throughout the statistical analysis it was assumed that the probability of success has a constant value p to satisfy the conditions underlying the binomial distribution.

[11] Please note that it is not necessarily that all users (of the same application) use the same screens. Hence, there are differences in the total number of fields evaluated among users.

[12] The confidence interval estimate is a specified range of numbers within which a population mean should lie (Zikmund, 1994: 404).

[13] To calculate the confidence interval for $\hat{R}(\Psi)$, $n = \frac{n_1+n_2}{2}$ where n_1 is the total number of test cases and n_2 is the total of fields evaluated by users.

Table 3
Derived values for $R(\Psi)$

Application	$\hat{R}(\Psi)$	80% Confidence Interval of $\hat{R}(\Psi)$	$\hat{R}(\Phi)$	80% Confidence Interval of $\hat{R}(\Phi)$	$\hat{R}(\Omega)$	80% Confidence Interval of $\hat{R}(\Omega)$
I	0.9467	[0.9276, 0.9658]	0.9776	[0.9661, 0.9891]	0.9255	[0.9010, 0.9500]
II	0.9266	[0.9077 0.9455]	0.9752	[0.9624, 0.9880]	0.9036	[0.8843, 0.9229]
III	0.9826	[0.9727, 0.9925]	0.9682	[0.9530, 0.9834]	0.9514	[0.9367, 0.9661]

Hypothesis 1 suggested that in a business environment, the reliability of the mapping from specifications to implemented software, $R(\Phi)$, is higher than the reliability of the mapping from user requirements to specifications, $R(\Psi)$. At an 80% confidence level, the values of $R(\Psi)$ and $R(\Phi)$ for applications I and II provide statistically significant support for Hypothesis 1. Application III, however, did not produce the same result. This conflicting outcome may be caused by the fact that application III's user requirements were the least volatile among the three applications. Because user requirements were clear initially and remained stable during the development, $R(\Psi)$ is high for Application III. An interview with an end user who involved in developing application III supported this supposition[14].

6.2 *Analysis of Questionnaire Responses*

Results from the three questionnaires were aggregated and extensive analyses of the data were carried out. A total of seventeen responses were obtained[15]. Multiple regression, correlation, and simple statistics were used to analyse the questionnaire responses. Because of the small sample size, factor and cluster analyses were not performed.

Regression analysis was used to test hypotheses 2a and 2b. A covariate, *system*, was included in both models. *System*, with values of 1, 2, or 3, differentiates the applications. The application with the highest $\hat{R}(\Phi)$ was valued 1 and the application with the lowest $\hat{R}(\Phi)$ was valued 3.

6.3 *Software Reliability and User Requirements Volatility*

Hypothesis 2a suggested that an inverse relationship exists between the overall software reliability, $R(\Omega)$, and the volatility of user requirements, i.e., as user requirements become more volatile, the overall software reliability decreases. The results of an ANOVA indicates a significant relationship between these two variables with a p-value < 0.05, one-tail test (see Table 4).

14 During the interview, users who were involved with the development process were asked about the volatility of user requirements. For application III, one user commented that the requirements of the application was changing in terms of the reporting aspect. The actual requirements of the application as dictated by the government statistician office, however, were comparatively stable.

15 Three different questionaires were filled in by 9 users, 4 analysts, and an application manager. One analyst was involved with the development of two applications, therefore separate responses were made for each application. Similarly, the application manager filled out a questionaire for each project.

Table 4

The Effects of User Requirements Volatility on $R(\Omega)$

Source	DF	Type III SS	Mean Square	F	Pr > F	Parameter Estimate
Model	2	0.00486992	0.00243296	6.22	0.0428	
Error	3	0.00117440	0.00039147			
Volatility	1	0.00467111	0:00467111	11.93	0.0204	-0.042522581
System	1	0.00324212	0.00324212	8.28	0.0318	-0.038941935

R-Square = 0.805701

Hypothesis 2b suggested that the R(Ψ) decreases as user requirements becomes more volatile. The results of an ANOVA test support hypothesis 2b with a p-value < 0.05, one-tail test (see Table 5).

Table 5

The Effects of User Requirements Volatility on $R(\Psi)$

Source	DF	Type III SS	Mean Square	F	Pr > F	Parameter Estimate
Model	2	0.00462809	0.00231404	5.97	0.0450	
Error	3	0.00116201	0.00038734			
Volatility	1	0.00459845	0.00459845	11.87	0.0206	-0.042190569
System	1	0.00252481	0.00252481	6.52	0.0419	-0.034365068

R-Square = 0.799311

7. Discussion

Results from the previous section suggest that in a business environment where user requirements change, the reliability of the mapping from specification to implemented software, R(Φ) is higher than the reliability of mapping from user requirements to specifications, R(Ψ) (Hypothesis 1). As user requirements become more volatile, the reliability of software decreases (Hypothesis 2a). This decrease in software reliability appears to be associated with problems with maintaining a reliable mapping from user requirements to specifications (Hypothesis 2b). A plausible reason for this deterioration may be the lack of a continuing user requirements validation process (Flynn & Warhurst, 1994; Nosek & Schwartz,1988), whereby the users judge whether their needs have been transformed accurately into a consistent list of requirements.

Organizations need software quality metrics to implement effective software management, i.e., organizations cannot manage what they cannot measure (Moller and Paulish, 1993:231). As part of their total quality management, they can use software metrics such as illustrated in this paper to improve reliability, reduce costs, and increase their software maturity level (Myers, 1994). Organizations can use statistical software testing as a first step toward more rigorous software development and maintenance methods such as Cleanroom Software Engineering (Dyer, 1992; Linger, 1994).

8. Contributions and Limitations

This research provided several contributions. First, the study used statistical software testing to measure the reliability of business applications. Second, the study developed a software reliability model that incorporates both the technical and end-user definitions of software reliability. Although the presented reliability model (Figure 1), adapted from the Lehman's "two-legged" software process model, and the two definitions of software reliability are not new, there is no known prior research that combined the two views. Third, this research provided empirical evidence of factors affecting the transformation of user needs to software specifications. Fourth, because organisations operate in different environments, case studies with multiple sources of evidence allow the researchers to take into account the context in which the study was conducted.

Although three applications were investigated, the study was based on three applications in a single organisation. Studies of business systems in other organisations or of different systems within this organisation could yield different results.

Because statistical testing requires substantial amounts of effort, a large sample of applications was not possible. A larger sample of applications may improve the statistical results.

The applications selected may have been subjected to selection bias in that those applications selected by the participating organisation may not be a representative of business software applications at this organisation or at other organisations. Test cases used in software testing were based on the researcher's understanding of the applications and may have biased the software reliability measures. In addition, responses from users for $\mathbf{R}(\Omega)$ may be biased because some of the users interviewed were recommended by the analysts.

References

Banker R., Datar, S., Kemerer, C., Zweig, D. "Software Reliability in a Maintenance Environment", Working paper, Carnegie Mellon University, 1994.

Bendell A. and Samson W. "Software Quality and Reliability" in "The State of the Art Report", Volume 13:2, Pergamon Infotech, England, 1985, pp. 17-26.

Cheung, R.C. "A User-Oriented Software Reliability Model," IEEE Transactions on Software Engineering, Vol SE-6, No 2, March 1980, pp. 118-125.

Daniels, B. and Hughes, M. "A literature survey of Computer software reliability", Quoted in Bendell and Samson, 1985.

Dyer, M. "The Cleanroom Approach to Quality Software Development," New York: John Wiley & Sons, 1992.

Flynn, D. and Warhurst, R. "An Empirical Study of the Validation Process within Requirements Determination", *Information Systems Journal*, Issue 4, 1994, pp. 185-212.

Freund, J. *Modern Elementary Statistics*. 5th ed. London: Prentice/Hall International, 1979.

Gerhart, S., Craigen, D., and Ralston, T. "Experience with Formal Methods in Critical Systems," IEEE Software, Vol 11, No 1, January 1994a, pp. 21-29.

Gerhart, S., Craigen, D., and Ralston, T. "Regulatory Case Studies," IEEE Software, Vol 11, No 1, January 1994b, pp. 30-40.

Keen , P. "Information systems and Organisational Change", Communications of the ACM, Volume 24, January 1989, pp. 24-33.

Khoshgoftaar, T.M., Szabo, R.M., and Woodcock, T.G. "An Empirical Study of Program Quality During Testing and Maintenance," Software Quality Journal, Vol 3, 1994, pp. 137-151.

Kokol P., Zumer V., and Stiglic B. "New Evaluation Framework for Assessing Reliability of Engineering Software Sytems Desgin Paradigms", in Brebbia C. and Ferrante A. *Reliability and Robustness of Engineering Software II*, 1991.

Leveson, N. "Software Safety in Embedded Computer Systems", *Communications of the ACM*, Volume 34, No. 2, February 1991, pp. 34-46.

Linger, R.C. "Cleanroom Process Model," IEEE Software, Vol 11, No 2, March 1994, pp.50-58.

Linger, R., Mills, H., and Witt, B. "Structured Programming: Theory and Practice", Phillipines: Addison Wesley, 1979.

Littlewood, B. "Theories of Software Reliability: How Good Are They and How Can They Be Improved?" IEEE Transactions on Software Engineering, Vol SE-6, No 5, September 1980, pp. 489-500.

Lyu, M. "Research and Development Issues in Software Reliability Engineering", *International Symposium on Software Reliability Engineering*, IEEE Computer Society Press, California, 1989, pp. 80-87.

May, J., Hughes, G., and Lunn, A.D. "Reliability Estimation From Appropriate Testing of Plant Protection Software," Software Engineering Journal, Vol 10, No 6, November 1995, pp. 206-218.

Musa J., Iaanino A., and Okumoto K. *Software Reliability : Professional Edition*. New York: McGraw-Hill, 1990.

Moller, K.H. and Paulish, D.J. "Software Metrics: A Practitioner's Guide to Improved Product Development," London: Chapman & Hall, 1993.

Myers, W. "Hard Data Will Lead Managers to Quality," IEEE Software, Vol 11, No 2, March 1994, pp. 100-101.

Newman M. and Sabherwal, R. "Information System Development: Four Process Scenarios With Case Studies", *Journal of Information Systems*, Spring 1991, pp. 84-101.

Nosek, J. and Schwartz, R. "User Validation of Information System Requirements: Some Empirical Results", *IEEE Transactions on Software Engineering*, Volume 14, No. 9, September 1988, pp. 1372-1375.

Oei, J., Proper, H., and Falkenberg, E. "Evolving information systems: meeting the ever-changing environment", *Information Systems Journal*, Issue 4, 1994, pp. 213-233.

Parnas, D., Van Schouwen J., and Kwan S. "Evaluation of Safety-Critical Software", *Communications of the ACM*, Volume 33, No. 6, June 1990, pp. 636-648.

Reade, C. and Fromme, P. "Formal Methods for Reliability" in Rook, P. *Software Reliability Handbook*, 1990, pp. 51-82.

Rook, P. *Software Reliability Handbook*. England: Elsevier Applied Science, 1990.

Rushby, J.M., von Henke, F. "Formal Verification of Algorithms for Critical Systems," IEEE Transactions on Software Engineering, Vol 19, No 1, January 1993, pp. 13-23.

Shooman, M. *Software Engineering - Design/ Reliability/ Management*. USA: McGraw-Hill, 1989.

Siefert, D. "Back To The Future", *International Symposium on Software Reliability Engineering*, IEEE, California: Computer Society Press, 1989.

Sommerville, I. *Software Engineering*, 3rd ed. England: Addison-Wesley Publishing, 1989.
Walton, G.H., Poore, J.H., and Trammell, C.J. "Statistical Testing of Software Based on a Usage Model," Software Practice and Experience, Vol 25, No 1, January 1995, pp. 97-108.
Weber, R. *EDP Auditing.* 2nd ed. NewYork: McGraw-Hill Book Company, 1988.
Wordsworth, J.B. "The CICS Application Programming Interface Definition," Proceedings of the Fifth Annual Z User Meeting, Springer-Verlag, 1991, pp. 285-294.
Yamamoto M., Aizawa M., and Yamagishi H. "High-Reliability Operating System ACOS-4/XVP", *NEC Research & Development*, Volume 35, No. 1, pp. 89-95.
Yin, R. *Applications of Case Study Research*, Applied Social Research Methods Series, California: Sage Publications, 1993.
Zikmund, W. *Business Research Methods: International Edition*, 4th Ed., Fort Worth: The Dryden Press, 1994.