Common Cause Failure Problems in Ultra-High Reliability Systems
- A View Point on Common Cause Internal Effects and Statistical Principles -
(초신뢰성 시스템에서의 공통원인 실패문제.
공통원인의 내부적 효과 및 통계학적 원리의 관점에서)

P. Park* K.H. Ko* C.S. Kim* H.K. Kim* H.S. Oh**
(박범, 고광호, 김철수, 김한경, 오현승)

This study involves a Common Cause Failure (CCF) problem on the ultra-high reliability required system development such as war game operations, nuclear power control, air traffic control, space shuttle missions, and large scale network communication systems. The system situation problems are defined according to CCF, reliability and system fault identifications for the development cost verifications in the multi-version redundant software system. Then, CCF analysis of redundant system, system principles and statistical dependence are also described. This validation of the CCF in the human software interaction system will notify software engineers to conceive what really is CCF contribution factors, not only the internal but the external ones.

I. INTRODUCTION

1. Overview

This research introduces the principle and a statistical domain of common cause failure in human-software interactions. This study is concerned with common cause failures during ultra-high reliability required software system development. This includes the situation problems, definitions, and statistical characteristics in the common cause failure of human-software interaction. It also concerns reliability between the human, who is presumed responsible for overseeing the software system, usage of the software system and software development.

In these days common cause failure studies[1][2][3] in the human-system area have been receiving wide attention especially in the software systems area. This is because the assumption of statistically independent

* ATM M&A Software Section
** Han Nam University
failure of redundant systems is easily violated in real human-software interaction processing systems. Since the software components are not independent of each other in regard to failure behavior, software redundancy does not improve reliability except in multi-version software development. Multi-version software system development is often requested to improve of reliability, especially in ultra-high reliability systems such as nuclear power control, air traffic control, space shuttle missions, war games, and large scale network communication systems.

There are three main components of this system situation in human-software interaction; the human as a software engineer, software as an operator system, and the hardware system as a software development workstation. It is important to analyze the characteristics and the environment of each subsystem.

The major common cause failures verified in the situation can contribute strongly to internal common cause failure effects in a multi-version software development project. The human reliability function, the principle of the common cause and its effect, and the probabilistic concept of common cause are presented in this work.

2. Common Cause Failure, Fault, and Reliability

Common cause failure is defined as “the simultaneous failure of more than one component [4].” Here, a failure of a component or subsystem is said to be a propagating failure when the failure changes the programming conditions, environments or requirements in such a way as to cause the failure of other components of software development. It is said to be a common-cause failure if more than one component fails due to a single cause (usually assumed to be external to the programming conditions of the human-software information processing system). Such common causes may be from the human domain attributable to cognitive processor, or to psychological behavior, or to physiological capacity, or to external disruption by man-made or natural events [5].

In hardware reliability theory where multiple components fail due to a single cause, a common-cause failure is said to have occurred. This can easily be extended to software components. A straightforward method to incorporate these common cause failures is given in Dhillon [1]. Let \( r \) be defined as the fraction of component failures that are common cause. Each component failure intensity \( \lambda \) is the sum of an independent fail-
ure intensity $(1-r)\lambda$ and a common cause failure intensity $r\lambda$.

A failure of a human-software interaction system occurs when that system does not perform its service/execution in the manner specified, whether because it is unable to perform the service/execution at all, or because the results and the external state are not in accordance with the specifications. Failure is “a departure of the external results of program operation from program requirements on a run [2].” A departure is the occurrence of a discrepancy between the desired output result stated in the requirement specifications for the specific run and the actual output result. Therefore, it represents a defect in a transformation. The output result is the set of values of output variable with a program execution. A discrepancy is defined as “the difference between the actual value of an output variable with an execution and the value expected by the requirement specifications [2].” The time of a failure is the time at which the discrepancy first occurs. The type of failure is defined as the conjunction of both run type or input state and discrepancy. The allocation of causes to human or components in human-software interaction systems is a purely pragmatic question regarding the stop rule applied for analysis after the fact.

Fault is defined as a defective, missing, or extra instruction or set of related instructions that is the cause of one or more actual or potential failure types. There cannot be multiple faults causing a failure. The entire set of defective instructions that is causing the failure is considered to be the fault. The requirement that the instructions be related is specified so that the count of the number of faults cannot be changed arbitrarily by a regrouping of instructions. The characteristics of a fault described in [2] are: (i) it is the cause of deviation from a standard; (ii) it is found on the causal path by tracing backwards from this effect; (iii) it is accepted as a familiar and therefore reasonable explanation; (iv) a cure is known.

Human error, which is the direct causal factor internally, consists of any significant deviation from a previously established, required or expected standard of human performance, that results in unwanted or undesirable time delay, difficulty, problem, trouble, incident, malfunction, or failure. In another way, it is described as the failure to carry out a specified task (or the performance of a forbidden action, or improper performance of a task) that could lead to dis-

---

1) The external state of a system is the result of a conceptual abstraction function applied to its internal state. The internal state of a system is the aggregation of the external states of all its components [1].
ruption of scheduled operations or result in
damage to property and component. Errors
can arise from many causes, but most of
them can be grouped in one of four catego-
ries [2]: communications, knowledge, in-
complete analysis, or transcription. In real
situations where arguments of precisely
what is or is not a human error are less im-
portant than what can be done to prevent
them, the operational definition may be
restricted to those errors (i) which occur
within a particular set of activities, (ii)
which are of some significance or criticality
to the primary operation under considera-
tion, (iii) which involve a human action of
commission or omission, and (iv) about
which there is some feasible course of action
which can be taken to correct or prevent
their reoccurrence.

**Human reliability** is defined as “the prob-
ability of accomplishing a job or task suc-
cessfully by humans at any required stage in
a system operation within a specified mini-
mum time limit [6].” Here, human-software
reliability can be defined as the probability
of successful performance with human-soft-
ware task ability and reliable systems at any
required stage in an operation of the human-
software interaction system within a speci-
fied duration of time.

Common-cause failure in system interaction
and statistical theory are discussed in [3], [7],
[8], [10].

3. Software Engineer and Programming
   Task

Will the software engineer solve a given prob-
lem? How well will he or she be able to perform
that task, and how will this system be well-
adapted to achieve the intended goal? The an-
wers depend on the following critical factors: the
nature of the task, the availability of the needed
expertise, and the ability to analyze and perform
the task in such a way that a computer program,
using limited levels of reasoning, can work out
what has to be done.

The conditions will tend to rule out certain ap-
plications from the start; the software engineer
should be able to perform the task, know how he
or she performs the task, be able to explain how
to perform the task, have the time to explain how
to perform the task, and be motivated to cooper-
ate in the enterprise.

Even if the above conditions are met, there may
be features of the task that limit the extent that
skills can be mechanized. This occurs, for instance,
if the task involves complex sensory-motor skills
beyond the scope of current technology in
robotics, computer vision, telecommunication net-
work and switching system, and high technologi-
cal software operation; also if the task involves
common cause reasoning or arbitrary amounts of everyday knowledge.

To be effective would also require an enormous amount of knowledge about the world: knowledge of objects and their properties, software engineers (or teams) and their motivations, physical and psychological causality. The fact is that only the most rudimentary notions about how to impact this kind of common cause, knowledge to computer software work exist. So any task that is not sufficiently self-contained to be encapsulated in a finite set of particular facts and general rules is definitely beyond the state of the art.

II. MULTI-VERSION REDUNDANT SOFTWARE SYSTEMS AND THE COMMON CAUSE EFFECT IN HUMAN-SOFTWARE INTERACTION

1. Multi-Version Redundant Software System and Common Cause Effects

A failure in programming task can occur during any phase of software development. Potential failures can sometimes be found by software engineers as the result of design review, and code proof reading. A software failure is a departure of operation from specified requirements in setting up or modifying a program. The common cause effect, as a reliability component of the common cause failure system, serially connected with other system components, in human-software interaction, is affected by internal common cause human domain errors.

The component structure of AND-OR rules in a $k$ out of $n$ component structure assumes that failures of different components are independent of each other [2]. This means that there can be no failures that result from the same cause [3]. Reliability is often increased in hardware systems by providing redundant components. In software systems, the situation is different. In hardware systems, the causes of failure associated with physically individual but functionally identical units in hardware component systems, are frequently independent. This phenomenon does not occur in software systems because multi-copies of a program are identical not only in function but also in the faults that can cause failures.

However, there is a possible exception in the situation of software development if multi-version software components are developed by different teams. There is some possibility that many faults introduced may be independent of each other with redundant software components developed by separate teams following the same specific require-
ments. This is described by Knight et al. [10] who points out experiments in multi-version programming which seem to indicate that failures in different versions are clearly not completely independent of each other. They do not appear to be all common either. Multi-version programming may well improve the reliability level, but not to the extent totally independent components would. Having totally independent components may be cost effective for critical modules of systems with ultra-high reliability requirements, such as nuclear power plants, air traffic control systems, space shuttle missions, war games, and large scale telecommunication network system.

The common cause effect is a system component that is not well recognized. It is serially-connected with the human-software system, operating system, and hardware system. Common cause failure effect can be defined as the consequences a common cause failure mode has on the operations, function or status of an item/task. Failure effects are classified as local effect, next higher level and end effect [11]. It is also defined by I. A. Watson [3] as inability of multiple, first-in-line items to perform as required in a defined critical time period, due to a single underlying defect or physical phenomena, such that the end effect is judged to be a loss of one or more systems. Human aspects of internal common causes are also analyzed in common cause effect at human-software interactions of multi-version software development [5]. Because this common cause effect affects the productivity and the development cost of a software project, removal of common causes is very important in improving reliability in an ultra-high reliability software project.

2. Situation Problems with Common Cause Effects

The mission of a specific software development project is to set up system components of human-software interaction. Each configuration is composed of a Software engineering WorkStation (SWS), a Central Operating Processor (COP) whose computer assigns and controls all the works at the local working stations, and a Multi-Version Software (MVS) development load. One approach to software design research using such a system that tends to be expensive, is to install two independent versions of MVS developed by two completely separate software development teams/engineers. The common cause effect affected by internal common cause human domain errors is determined using redundant components in this
case as in Figure 1.

![Event Diagram of A Multi-Version Redundant Software System in Human-Software Interaction.](image)

In the given example, the system reliability is

\[(0.99)(0.95)[1 - (1 - (0.98)R_c^* )^2]R_c^*\]

If there is no common-cause error effect, then \(r = 0\) and \(R_c\) becomes 0.809 (\(\lambda_c = 0.0662\) failure/cpu hr.). However, the chances are that \(r\) is relatively large, that is, similar common errors are made by each team. If \(r = 0.5\), then \(R_c = 0.922(\lambda_c = 0.0254\) failure/cpu hr.). In the case of \(r = 0\), the development cost for the MVS software will be about \$580,000 \((\$290,000\) for each copy of the software). Similarly, if \(r = 0.5\) the software development cost will be about \$1,150,000. An additional \$250,000 will be incurred for the second unit of MVS hardware. The total cost will be \$830,000 or \$1,400,000, depending on the value of \(r\) in the Musa’s study [2].

III. COMMON CAUSE FAILURES IN HUMAN SOFTWARE INTERACTIONS

Common cause failures, which were overlooked 20 years ago, have been receiving wide attention especially in the software systems area. This is because the assumption of statistically-independent failures of redundant systems is easily violated in the case of human-software interaction. common-cause failures concern the possibility that system or mission failure involving multi system component failure may occur due to a common cause, i.e. the loss during some critical period of multiple, redundant systems, component functions, due to an underlying common control mechanism, fault or phenomena.

A related study for common-cause failure initially, a small research group on Rare Event was to investigate and organize the program of work based on the findings of the task force in the following areas:

- rare event data collection and analysis;
- common mode failure analysis;
- human error analysis and quantification;
- statistics and decision theories applicable to rare events;
- interdisciplinary communication and tutorial programs on rare events programs and their solution.

I. A. Watson described that the analysis of common cause may be complicated because of various considerations including [3]:

i) recognition of many possible causes of common cause failure and their identification;
ii) selection of models to be used in the quantification of system reliability;
iii) the availability of historical data;
iv) the comparative rarity of common cause failure.

There is a different situation in a human-software system compared with a hardware system. A failure in one Transaction Processor (TP) software component would also occur in another since they are identical [2]. Both copies contain identical faults. Since the software components are not independent of each other in regard to failure behavior, software redundancy does not improve reliability. This is a commonly occurring and very important point for software components. A common cause failure in the software development processing system is any instance where multiple components malfunction due to a single cause.

At first, in modeling common cause failures in software development, it is desirable to introduce the initiating events physically. An initiating occurrence is to be regarded as an external event such as a flood, earthquake, power outage, or fire which can cause the failure of several components simultaneously, due to the environmental stresses occasioned by its occurrence.

Also simultaneously, another common cause failure of several components occurs when one component has several functions, so that its failure prevents each of these individual functions.

Another possible common cause is the existence of standby components which are called into use when specified components have failed. The conditional waiting time until a failure in the standby component is observed is different from the waiting time until failure if it were in non-standby usage.

Several factors of common cause failures in general systems are as follows [1]:

ii) Equipment failure resulting from some unforeseen external event: fires, floods, earthquakes, tornadoes.
iii) Design deficiencies: During the design phase of the system some faults may
have been overlooked. For example, the interdependence between electrical and mechanical items of a redundant system may have been overlooked during the design phase of a system.

iv) Operation and maintenance errors: Occurrence of these errors may be due to improper maintenance, carelessness, improper calibration, the same person performing maintenance on all redundant units repeating the same mistake on all of them, etc.

v) Multiple items purchased from the same manufacturer: all these items may have the same manufacturing defects.

vi) Common external power source to redundant units.

vii) Functional deficiency: misunderstanding of process variable behavior, inadequacy of designed protective action, inappropriate instrumentation, etc.

1. Common Cause Failure Analysis of Redundant Systems

There is a method for incorporation common cause failure in a redundant network analysis of the human-software processing system [1]. Dhillon described the CCF on the time-dependent reliability models involved with (i) reliability evaluation of a redundancy with CCF, (ii) reliability analysis of a repairable two-unit redundant system, (iii) availability analysis of a two-nonidentical-unit system with CCF. There is an assumption that network units are identical and independent, and also that the same portion of common cause failures is associated with other redundant network components.

It is assumed in the common cause failure model that

\[ r = \text{fraction of component or system failures that are common-causes} \]

\[ \lambda_u = \lambda_i + \lambda_c \]

where

\[ \lambda_u : \text{the component constant failure rate} \]

\[ \lambda_i : \text{the component independent constant failure rate} \]

\[ \lambda_c : \text{the component or system constant common cause failure rate.} \]

Since

\[ r = \frac{\lambda_c}{\lambda_u} \]

then

\[ \lambda_c = r \lambda_u. \]

By arranging these equations, we get

\[ \lambda_i = (1 - r) \lambda_u. \]

2. Parallel Component Network Systems

A series-parallel component network
system model in multi-version software is illustrated in Fig. 2. This is actually a modified parallel network system to incorporate common cause failures. The parallel portion of the network represents \( n \) independent, failure components and the single component in series is a hypothetical component representing system common cause failures \([2]\). The failure of the hypothetical common cause failure component will cause system failure.

The human-software reliability, \( R_{hs} \), of the human-software processing network given in Fig. 2 is

\[
R_{hs} = [1 - (1 - R_s)^n]R_c
\]

where

- \( R_s \) : the independent failure mode reliability of a component;
- \( R_c \) : the common cause failure mode system reliability;
- \( n \) : the number of identical components.

The time dependent reliability of the \( i \)th independent component with constant failure rate is

\[
R_i(t) = e^{-\lambda t}.
\]

Similarly, the hypothetical common cause failure component reliability is

\[
R_c = e^{-\lambda_t}.
\]

By substituting variables in all of these equations

\[
R_{hs}(t) = [1 - (1 - e^{-(1 - \alpha)\lambda c})^n]e^{-\lambda c}.
\]

To calculate the mean time to failure (MTTF), \( R_{hs}(t) \) is integrated over the time interval \([0, \infty]\)

\[
MTTF = \int_0^\infty R_{hs}(t) = \left[ \sum_{i=1}^{n} (-1)^{i+1}\binom{n}{i} \right] / [\lambda c + n\lambda c (i - n)].
\]

The common cause principle and statistical dependence are described in the next section.
IV. COMMON CAUSE PRINCIPLE
AND STATISTICAL
DEPENDENCE

Wesley Salmon [9] and Bas C. Van Fraassen [7] have successively refined and elaborated Reichenbach’s principle of the common cause, as part of a wideranging inquiry into statistical inference and explanation. In this section, the probabilistic concept of common cause, that is, the principle of the common cause, is derived.

Reichenbach’s common cause principle says roughly that if there is a positive correlation between simultaneous, spatially separate events, then there is a third event in their common past which explains for their frequent joint occurrence.

This is an empirical statement. It reminds one somewhat of certain traditional principles of metaphysics, such as that every event should have a cause. A scientific theory concerning those correlated events is not complete unless it exhibits, or implies that there is, such a common cause as a tactical maxim for scientific inquiry.

Extreme Bayesianism is the position that a rational person's epistemic state can be represented faithfully and without loss by means of a probability function; that any probability function at all can so represent some rational person; and that rational change of epistemic state consists in conditioning of that personal probability on the total evidence received.

If Reichenbach’s principle can be explained as an empirical proposition, there are many probability functions that do not give it a high value. If one manages secondly his garden of beliefs in such a way that, whenever he has a certain degree of belief that two events are positively correlated, he gives at least that degree of belief to the proposition that they have a common cause, then either he gives probability one to that empirical proposition (the common cause principle) or else his belief change does not follow the pattern of conditioning on the total evidence, As Salmon has rightly emphasized, the principle of the common cause will appear as a powerful argument for scientific realism when it comes in any of these rational inference related forms [9].

1. The Principle of the Common Cause

Two events, A and B, are called statistically independent if \( P(AB) = P(A)P(B) \). When the equality is replaced by the greater-than relation we may call them positively correlated. A third event C, using the condi-
tional probability $P(-/C)$ may have a relationship with either of these notions:

if (1) $P(AB) > P(A)P(B)$

then there is an event $C$ such that

(2) $P(AB/C) = P(A/C)P(B/C)$

(3) $P(AB/C) = P(A/\bar{C})P(B/\bar{C})$

(4) $P(A/C) > P(A/\bar{C})$

(5) $P(B/C) > P(B/\bar{C})$

With the time element, the $AB$ is an event which happens at a given time if and only if both $A$ and $B$ happen at that time. Suppose that put $A_t$ for the (individual, non-generic) event is the occurrence of (generic) event $A$ at time $t$. Suppose that has always occurred $C$ in the intersection of the past cones of the occurrences of the $A$ and $B$. There are two relatively independent questions which may be raised. The first question is whether there is always an event $C$ at a preceding time such that the above probabilistic relations hold. The second one is whether if $C$ satisfies the stated conditions, it follows that $C$ accounts for the correlation. (Can it reasonably be termed the cause?)

2. Statistical Dependence

The following relationships are important on examining statistical dependence:

(6) $P(AB) > P(A)P(B)$ : 

$A$ and $B$ are positively correlated ;

(7) $P(A/B) > P(A)$ : 

$A$ has a positive dependence on $B$ ;

(8) $P(A/B) > P(A/\bar{B})$ :

$B$ is positively relevant to $A$ ;

(9) $P(A/BC) = P(A/C)$ :

$C$ screens off $B$ from $A$.

In each case, if the probability function $P$ is replaced by the conditional probability $P_x

= P(-/X)$, then the same terminology can be used with adding the rider relative to $X$. One can say easily how cognate terms such as independent, negatively relevant, and the like are used. Symmetric term, $A$ and $B$ are, is appropriate because the relationship is so clearly symmetric in $A$ and $B$. It is important that there is no need to memorize the terms in (6) – (8), and their cognates, because the ones which are easily confused are actually equivalent (provided all the probabilities involved are well-defined). To get their this precise, let the letter, $R$, range over

positive linear relations among numbers, defined by the properties [7]:

If $0 \leq x, y \leq 1$, and $0 < b$

then

(1) $xRy$ iff $bxRby$

(II) $xRy$ iff $(b+x)R(b+y)$

where $=, <, >, \leq, \geq$ are all positive linear relations.

LEMMA. If $R$ is a positive linear relation
and \( P(X), P(\overline{BX}) \) are positive, then the following are mutually equivalent:

\[
\begin{align*}
(A) \quad & P(AB/X)P(A/X)P(B/X) \\
(B) \quad & P(A/BX)P(A/X) \\
(C) \quad & P(A/BX)P(A/\overline{B}X)
\end{align*}
\]

Using this Lemma, there are restatements on the properties of the common cause in Reichenbach’s principle in follows:

(10) If \( A \) and \( B \) are positively correlated, then there is an event \( C \) such that
   
   (A) \( A \) and \( B \) are independent relative to \( C \) and also relative to \( \overline{C} \),
   
   (B) \( C \) is positively relevant both to \( A \) and to \( B \).

(11) If \( B \) is positively relevant to \( A \) then there is an event \( C \) such that

   (A) \( C, \) and \( \overline{C}, \) screens off \( B \) from \( A \),
   
   (B) Both \( A \) and \( B \) have a positive dependence on \( C \).

V. CONCLUSION

In this study common cause failure problem was defined and verified statistically on the ultra-high reliability required system development. The severity of CCF problem in the software system according to cost verification is well described. Then, this preview will notify software engineers to conceive the real CCF contribution factors.

References


