안전성 확보를 위한 위험원 분석 기법간 상관관계에 대한 연구

A study on the Correlation Hazard Analysis for Signaling System Safety

한찬희*	이영수**	안 진**	조우식**
Han, Chan Hee	Lee, Young Soo	Ahn, Jin	Jo, Woo Sic

ABSTRACT

Computers are increasingly being introduced into safety and reliability critical systems. The safe and reliable operation of these systems cannot be taken for granted. Malfunctions of these systems can have potentially catastrophic consequences and they have already been involved in serious accidents.

Software fault prevention, fault tolerance, fault removal and fault forecasting are the techniques to be used, implemented and verified for embedded software in critical systems as the contributors to safety and reliability of the software. To use them when developing a software product, a relationship must be established between them and the development processes, the methods and techniques to be used to develop software, as well as with the different product architectures.

Railroad signaling system software is a safety-critical embedded software with realtime and high reliability requirements. The primary purpose of the safety management is to prevent the loss of lives or physical damages arising from potential hazards in the railroad signaling system.

This study provides a systematic approach to analysis of potential hazards for their management during the system life cycle to assure the identification and definition of the most appropriate hazards.

1. 서 론

안전성과 신뢰성이 요구되는 핵심 시스템에 컴퓨터의 도입은 이젠 당연시 되고 있다. 간단한 임베디드 시스템에서부터 항공장비, 원자력 발전소 제어시스템, 자동차의 브레이킹 시스템 그리고 열차제어 시스템에 이르기까지 일상생활과 밀접한 관계에 있는 많은 시스템에 마이크로프로세서가 핵심 부품으로 쓰이고 있으며, 이들 시스템은 수많은 사람들의 생명과 직결되고 있다. 또한 생명과 직결되지는 않으나 일상생활에 피해와 혼란을 초래하는 컴퓨터 시스템도 존재한다.

이러한 시스템의 안전하고 신뢰성 있는 운영은 저절로 보장될 수 없다. 이들 시스템의 오류 및 오작동은 재난에 맞먹는 결과를 유발할 수 있으며, 이미 심각한 사고가 발생한 경우도 있다. 소프트웨어가 우리의 일상생활 가운데 많은 영역에서 널리 활용되고 있으며 소프트웨어 오작동이 불편을 주기도 하지만, 심각한 경우에는 생명을 위협할 수도 있다는 점을 보여 주는 치명적인 고장 사례를 철도에서도 쉽게 찾아볼 수 있다. 열차 충돌, 열차 탈선, 출입문 오작동으로 인한 승객 추락 등을 들 수 있다.

* 경봉기술(주) 주임연구원 chanees@kbtech.co.kr (032)680-0895

** 경봉기술(주) 수석연구원

** 경봉기술(주) 수석연구원

** 경봉기술(주) 선임연구원

아러한 소프트웨어의 오류 및 오작동은 충분한 테스트를 거치고 인증되었다 하더라도, 몇 개월 또는 몇 년 뒤에 이상을 보이기도 한다.
안전성 필수 어플리케이션에 사용되는 핵심 소프트웨어나 임베디드 소프트웨어는 매우 크고 복잡하다. '어떤 장치에 내장된 임베디드 소프트웨어 1메가바이트는 100% 신뢰할 수 없다'라고 흔히들 말하고 있다. 이는 소프트웨어가 오류를 일으킬 가능성이 충분히 있다는 점을 수용해야 한다는 의미이다.
이러한 소프트웨어 제품 내부의 수많은 다양한 상호작용을 테스트한다는 것은 엄청난 일이며, 개발자와 고객 모두 그런 테스트를 하기 매우 어렵다. 그렇기 때문에 사고를 유발할 수 있는 소프트웨어의 원인들이 매우 다양하기에 이런 연관 관계를 직접적으로 모두 밝혀낼 가능성은 매우 희박하며 어려운 부분이다. 여기서 다음과 같은 의문점이 제기된다.

- 정밀한 설계와 광범위한 테스트 이외에 소프트웨어의 안전성을 보장할 수 있는 다른 방법은 없는 것일까?
- 오작동을 일으키는 소프트웨어가 스스로 진단하고 문제를 바로 잡거나 사고로 발전되기 전에 스스로 시스템을 중단시킬 수 없을까?

이러한 질문에 대한 답은 발생 가능한 잠재적인 위험원의 파악, 감소, 배제, 예방할 수 있도록 지원해 주는 기법이 있기에 "가능"이라 말할 수 있다.
소프트웨어 장애 예방, 장애 허용, 장애 제거 및 장애 예측은 소프트웨어 안전성 및 신뢰성 확보를 위한 방법으로, 핵심 시스템의 소프트웨어를 위한 설계/구현/검증에 활용되는 기법들이다. 언급된 방법 모두 매우 중요하지만, 핵심 소프트웨어 개발 시에 적절하고 체계적으로 활용되거나 구현되고 있지 않다.
이러한 방법들은 단일 프로젝트로 감당하기 어렵기에, 본 연구에서는 몇몇의 방법과 이에 속하는 기법에 대한 상호 보완성 및 적용성을 검토 연구하였다.

소프트웨어의 테스트는 소프트웨어를 실행하여 소프트웨어 장애를 제거하기 위한 것이다. 복잡한 소프트웨어인 경우에는 모든 테스트가 가능하지 않으며, 소프트웨어 완전성의 통계적 분석 같은 새로운 테스트 방법은 아직 미흡한 부분이 있다. 이러한 오류가 없음을 증명하는 테스트보다는 정성적, 정량적 분석이 더 바람직할 수 있다. 정성적, 정량적 분석은 소프트웨어 라이프사이클 초기 단계에서 활용할 경우 더 큰 효과를 볼 수 있다. 정성적, 정량적 분석의 특징을 설명하면 다음과 같다.

- 다양한 소프트웨어 개발 단계에서 체계적으로 수행함으로써, 소프트웨어의 안전성/신뢰성을 확보하는데 도움을 준다.
- 모순적인 아키텍처의 안전성/신뢰성 기준을 수용하기 위한 소프트웨어 설계의 조정을 지원한다.
- 소프트웨어 집약적 시스템의 소프트웨어에 적용되는 불충분한 스트레스 테스트를 보완한다.
- 시스템 안전성/신뢰성 구현과 이후의 안전성 및 신뢰성 평가를 지원한다.
- 유지 관리 및 운영 단계에서 요구되는 장애 제거 활동 수준을 감소시킨다.

본 논문에서는 정성적, 정량적 분석 기법인 FMEA(Failure Mode and Effect Analysis), HAZOP (Hazard and Operating), FTA(Fault Tree Analysis)와 같은 전통적인 시스템 안전성 분석기법을 적용하여, 이들 분석기법을 어떻게 적용하고 상호 보완적으로 활용할 수 있는지에 대해 기술한다.

2. 본론

2.1 소프트웨어 안전성

서론에서 설명한 바와 같이 제어와 관련된 핵심 소프트웨어의 안전성은 매우 중요하며, 이를 유지할 수 있도록 활동하는 것 또한 매우 중요하다. 그러나 소프트웨어의 안전성이라고 하며 대부분의 사람들은 쉽게 이해하지 못할 것이다.

소프트웨어의 안전성에 대한 개념을 이해하기 위해서는 시스템의 안전성에 대해 우선 이해해야 한다. 시스템 안전성은 다음과 같이 구분된다.

① 위해 요소의 파악
② 중요성 및 발생 가능성 측면에서의 위해 요소 평가(리스크 평가)
③ 위해 요소의 제거 또는 통제를 위한 장치 디자인
④ 시스템 디자인 및 구축 이후 최종적 리스크 평가

소프트웨어의 안전성에 대한 정의는 ISO9126에서는 다음과 같이 정의하고 있다.

"지정 사용 상황에서 사람, 비즈니스, 소프트웨어, 재산 또는 환경에 대하여 허용 수준의 위해 발생 리스크를 달성하는 소프트웨어 제품의 성능"

각종 문서의 소프트웨어의 안전성에 대한 정의에서는 Failure, Error, Fault, Hazard, Risk 등 확실하게 구분되어야 하는 용어들이 간과되기도 한다. 본 논문에서는 적용되는 용어의 통일성을 위해 다음과 같이 정의하였다.

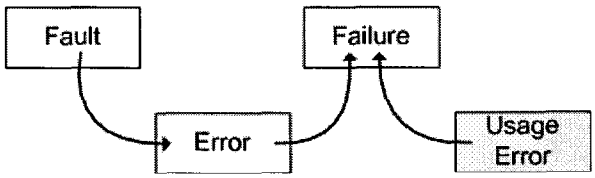

그림 1. 용어 차이점

Failure

'필요 기능을 수행할 수 있는 능력이 종식된 것' [ECSS],[IEC61508]
'소프트웨어에서의 failure는 오류의 발현이다.' [EN50128]

Error

error는 두 가지 의미를 가지고 있다. [IEEE-612]는 "소프트웨어 장애를 유발하는 사람의 작용"으로 정의하지만, 이 의미는 "실수(mistake)"라는 것으로 생각하면 될 것이다. 대부분의 표준문서에서는 error를 "계산, 관찰, 측정 값 또는 조건과 지정, 또는 이론적으로 정확한 값 또는 조건 사이의 차이"로 정의하고 있다.[IEC61508], [EN50128]

Fault

특정 운영 조건에서 이상으로 이어질 수 있는, 시스템의 불완전 또는 결함으로 정의된다. fault는 "시스템에 일시적 또는 영구적으로 하드웨어나 소프트웨어의 오류를 유발하는 원인"이다. [EN50128], [DO178B]

2.2 소프트웨어 장애(fault) 제거

위에서 정의한 바와 같이 소프트웨어의 고장(failure)은 장애(fault)에서부터 시작하게 된다. 그렇다면 고장이나 오류(error)가 발생하기 전인 장애(fault)를 제거하는 방법이 시간적, 비용적으로 가장 효율적이고 효과적이라 할 수 있다. 그렇다면 다음과 같은 질문이 나오게 된다.

- 소프트웨어 장애(fault)를 제거하는 기법은 무엇을, 어떻게 분석해야 하는가?
- 여러 기법의 비교에 어떤 기준을 적용해야 하는가?

위의 첫 번째 질문은 다음과 같이 두 가지 세부 질문으로 나눌 수 있다.

- 소프트웨어 장애(fault)를 제거하는 기법은 무엇을, 어떻게 분석해야 하는가?
  - ↳ 소프트웨어 장애 제거 기법은 무엇을 분석해야 하는가?
  - ↳ 이 분석을 어떻게 수행해야 하는가?

소프트웨어 장애(fault)는 오류 및 고장의 원인이므로, 소프트웨어 장애(fault)를 제거하면 시스템의 안전성이 확보된다. 특히 본 연구에서는 장애(fault)가 발생하여 차후에 소프트웨어 이상을 유발하여 치명적인 결과를 가져올 수 있는, 개발 단계의 소프트웨어 장애(fault) 제거에 중점을 두었다.

우선 소프트웨어에서 피해야 할 핵심 소프트웨어 장애 모드가 무엇인지 파악하는 것이다.
장애 모드를 파악한 다음에는 이의 발생을 피하기 위하여 그 원인을 제거해야 한다. 원인 제거는 다음과 같은 단계를 거친다.

- 장애 감지 : 장애를 발견하거나 발견하도록 설계된 단계, 즉 장애가 발생할 것임을 결정하는 단계
- 장애 분리 : 장애의 출처 또는 위치를 파악하기 위한 단계, 즉 장애 원인의 위치와 특성을 결정하는 단계
- 장애 복구 : 장애를 제거하거나 이상 발생을 피하기 위한 단계, 즉 장애를 제거하거나 이의 통제 또는 허용을 위한 수단을 제공하는 것.

장애 제거기법은 이상의 모든 단계를 포괄해야 하며, 모든 라이프사이클 전 과정에서 적용되어 제거되어야 하나, 설계 단계에서 중점적으로 이루어지고 테스트 단계에서 이를 확인하게 된다.

2.3 소프트웨어 장애제거를 위한 분석기법

이미 언급한 바와 같이 소프트웨어 안전성과 관련하여 이 논문에서는 소프트웨어의 장애 제거에 관해서 다룬다. 소프트웨어의 오류 및 장애를 제거하기 위한 다양한 기법들을 정리하면 다음과 같다.

기법	설명	특징
알고리즘 분석 (Algorithm analysis)	알고리즘의 정확성, 적절성, 안정성을 점검하고, 알고리즘이 모든 정확성, 타이밍, 사이징 기준에 부합하는지 점검한다. 알고리즘 분석은 알고리즘, 등식, 수학공식 또는 표현의 올바른 구현 여부를 시스템 또는 소프트웨어 기준에 대비하여 확인한다.	반복성, 가용성, 신뢰성을 만족시킬 수 있음.
공통원인 이상 분석 (Common cause failure analysis)	동시에 예비 부분에서 동일한 이상이 발생함으로써, 예비 시스템 구비의 혜택을 훼손시킬 수 있는 잠재 이상 파악	
데이터 흐름분석 (Data flow analysis)	프로그램 실행 시에 프로그램 변수가 초기화, 변형, 또는 조회될 때 프로그램 변수의 행동 점검	
ETA (Event tree analysis)	개시 이벤트 이후 시스템에서 발생하는 이벤트 순서를 도식적으로 모델링하여, 그에 따른 결과가 얼마나 심각한지 보여주는 방법	정량적 분석기법 귀납적 방법 상향성 접근방법
FMEA (Failure mode effect analysis)	시스템의 잠재적 장애 모드, 이들 장애의 영향, 그리고 그 중요성을 체계적으로 파악하는 절차이다.	정성적 분석기법 귀납적 방법 상향성 접근방법
FTA (Fault tree analysis)	독자적으로 또는 다른것과 함께 시스템의 장애상태로 이어지게 하는 원인의 파악을 위한 체계적인 접근 방법	정량적 분석기법 연역적 방법 하향성 접근방법
HAZOP (Hazard and operability analysis)	컴퓨터 시스템 컴포넌트 섹션과 컴퓨터 시스템의 운영 상태를 체계적으로 검사함으로써, 제어 시스템에서 잠재적 위해 상황을 유발할 수 있는 이상 모드를 파악하는 방법	정성적 분석기법 Guide word를 통한 전문가 난상토론
PHA	시스템 정보가 적은 상태에서 수행되며, 어느정도	귀납적 방법

(preliminary hazard analysis)	의 위험이 존재하는지 정성적으로 평가하는 방법	상향성 접근방법

2.4. 조건에 따른 분석기법

본 연구에서는 소프트웨어의 안전성을 확보하기 위해서는 잠재되어 있는 장애 발생 원인을 제거할 수 있는 방법에 중점을 두었으며, 그 방법으로는 다양한 기법을 적용함으로서 최대한 제거하고자 하였다. 2.3에서 제시된 기법들 이외에도 체계적이고 확률적인 기법이 있으나 적용하기에 매우 복잡하고 기법을 이해하기 어려운 기법은 연구에서 제외하였다.

열차제어시스템과 같이 운행에 핵심적인 시스템의 소프트웨어는 인명의 피해와 매우 밀접한 관계를 가지고 있다. 이러한 소프트웨어의 오류 및 장애를 제거하기 위해서는 1~2가지의 기법만으로는 미흡한 결과가 도출되기도 하였으며, 여러 가지의 기법을 적용할 경우에도 상호 연관성을 가지지 못하고 독립적으로 적용될 경우 이 또한 만족스럽지 못한 결과가 도출되었고 결과에 대한 테스트와 검증 그리고 관리 또한 어려웠다.

본 연구에서는 위와 같은 결과의 재발을 방지하기 위해 기법간의 조건에 따른 상호 연계성과 보완성에 대해 실제 개발되고 있는 소프트웨어의 안전성 활동을 바탕으로 심도 깊은 연구 결과를 제시하였다.

다음은 여러 가지 조건에 따라 기법들을 분류한 내용이다.

2.4.1 수리적 방법에 따른 분류

정성적 분석 (Qualitative analysis)	What if analysis
	고장모드 및 영향 분석 FMEA
	HAZOP
정량적 분석 (Quantitative analysis)	부품계수법 (Part count approach)
	신뢰도 블록 다이어그램 분석 (Reliability block diagram analysis)
	치명도 분석 CA
	사상수목분석 ETA
	결함수목분석 FTA

2.4.2 논리적 추론과정에 따른 분류

귀납적 기법 (Inductive techniques)	예비 위험원 분석 PHA
	결함 위험원 분석 FHA
	운용 위험원 분석 OHA
	고장모드 및 영향 분석 FMEA
	사상수목분석 ETA
연역적 기법 (Deductive techniques)	결함수목분석 FTA

2.4.3 분석방향에 따른 분류

상향성 접근 기법 (Bottom-up approaches)	예비 위험원 분석 PHA
	결함 위험원 분석 FHA
	운용 위험원 분석 OHA
	고장모드 및 영향 분석 FMEA
	사상수목분석 ETA
하향성 접근 기법 (Top-down approaches)	결함수목분석 FTA

최종적으로 2.3의 소프트웨어 장애 제거를 위한 기법과 2.4의 조건에 따른 분석기법등을 고려하여 연구에 적용된 기법은 일반적으로 많이 적용하고 있는 PHA, FMEA, HAZOP, FTA 기법들이다.

2.5 상호 적용성

위에서 언급된 PHA, FMEA, HAZOP, FTA 기법들은 단독적 행위만으로도 만족스러운 결과를 도출할 수 있지만, 기법의 분류에서와 같이 충족할 수 없는 부분이 반드시 존재함으로 여러 기법의 적용으로 이를 충족시키고 만회할 수 있다.

안전성의 확보를 위해서는 정성적인 결과만으로는 개발된 소프트웨어가 안전하다고 결정할 수 없기에 정량적인 값의 산출로 증명해 보여야 한다. 그렇기에 정성적 기법인 FMEA와 HAZOP을 수행한 후 정량적 값 산출을 위해 FTA를 수행해야만 한다.
그러나 각각의 기법을 적용하여 도출된 결과는 어떻게 관리되고 적용되어야 할까?
이에 대한 답은 다음의 그림에서 확인할 수 있다.

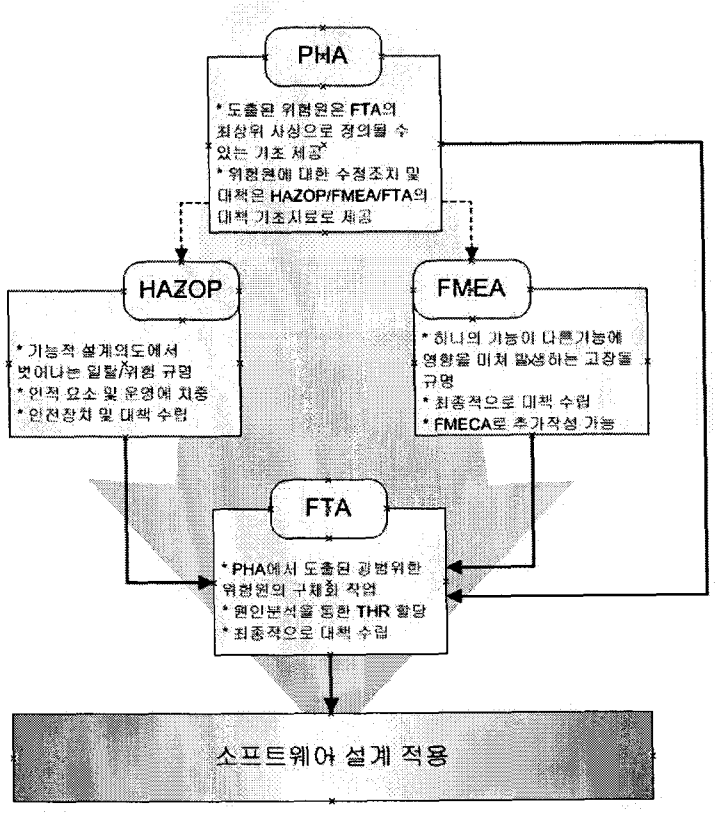

그림 2. 분석기법간 관계

위의 그림은 각 기법의 결과가 다른 기법에 어떠한 영향을 미치는지 어떻게 적용되는지를 나타내고 있다. 각 기법의 설명은 다음과 같다.

2.5.1 PHA
PHA는 개발하고자 하는 소프트웨어의 세밀한 기능에 대한 분석을 수행하는 것이 아니다. 소프트웨어의 정확한 개념을 통해 전반적인 구성과 흐름에 대해 위험성을 분석하게 된다. PHA는 기존의 유사 시스템이나 동종 시스템이 존재할 경우 기존의 기록된 오류나 사고를 토대로 도출하게 되는데, 그러다 보니 너무 세부적인 하위 기능레벨 또는 너무 광범위한 부분에 대해 결과가 도출 되는 사례가 종종 발생하게 된다.

> 본 연구에서 제시하고자 하는 PHA의 결과는, FTA의 정상사상에 적용할 수 있어야 함을 전제로 도출되어야 한다. 즉, PHA에서 도출된 위험원(장애, 오류)은 차후 수행하게 되는 FTA의 정상사상에 입력되어야 하기에 그 범위를 사용자가 쉽게 정의할 수 있게 된다. 그 이유로 PHA는 귀납적, 상향성 접근 방법으로 도출된 결과이므로 FTA의 연역적, 귀납적 방법에 적용될 경우 크게 FTA 정상사상의 범위를 벗어나지 않게 된다.

2.5.2 HAZOP
HAZOP은 원래 설계된 대로 작동하는 한 근본적인 위험성이나 운용상의 문제가 없음을 기본 전제로 시작한다. 그러나 어떠한 이유로 인해 위험원이 발생하게 되면, 시스템이나 소프트웨어가 이를 감당할 수 있느냐 없느냐에 따라 사고가 발생한다는 것이다.
PHA 보다는 보다 세밀하고 기능적으로 접근하는 기법이기에 그 결과 또한 많고 복잡하다. 그러나 HAZOP은 정성적인 결과를 제시하게 된다. 이러한 정성적인 결과를 보다 신빙성 있는 결과로 제시하기 위해서는 정량적인 값이 증명되어야 한다.

본 연구에서는 이러한 HAZOP의 정성적인 결과를 정량화 할 수 있는 방법으로 FTA를 제시하였다.
FTA는 정량적 분석기법이다. HAZOP의 결과는 FTA의 중간 또는 하위 사상에 적용할 수 있다. 이렇게 적용된 결과는 HAZOP에서는 정성적이지만 FTA를 통해 정량적인 수치를 더하게 되어 그 신빙성과 신뢰성이 높아진다.
즉, HAZOP의 결과는 FTA의 중간이나 하위 사상에 기초 자료로 이용된다.

2.5.3 FMEA

FMEA는 전형적인 귀납적 분석방법이며 상향성(bottom-up), 정성적인 분석기법의 대표적인 방법이라고 할 수 있다. FMEA는 FTA와 같이 소프트웨어 구성요소간 상세한 기능적 연관관계나 종속성에 관한 정보를 제공하지 않으며, 도출된 결과 또한 정성적이며 사용자에게 비쥬얼하게 접근하지 못한다.

따라서 FMEA도 HAZOP과 마찬가지로 정성적인 결과를 정량화 할 수 있는 FTA의 기초자료로서 활용되어야 한다.
즉, FMEA의 결과는 FTA의 중간이나 하위 사상에 기초 자료로 이용된다.
예외적으로 치명도를 포함한 FMECA일 경우엔 정량적인 치명도 값이 FTA로 인해 수정되지는 않는다.

2.5.4 FTA

FTA는 정상사상이라는 바람직하지 않은 사상을 시작으로 그 발생원인이나 이에 기여하는 조건, 요인들을 찾아 시간적 흐름을 거슬러 분석해 나가는 하향성, 연역적 구조를 가지고 있다.
FTA의 분석절차는

①분석범위의 정의 및 분석수준의 결정,
②대상 소프트웨어의 특성파악,
③정상사상 결정,
④결함수 구성,
⑤정성적 분석,
⑥정량적 분석,
⑦결과 평가

이상과 같은 절차를 거치게 되는데, 본 연구에서 제시하고 있는 기법 간 상호 연계성을 적용하게 되면 ①~⑤의 단계까지 PHA, HAZOP, FMEA를 통해 기본적인 틀이 마련되게 된다.

2.6 결과

소프트웨어의 구상과 설계 그리고 개발, 시험까지 전반적인 개발 생명주기에서 수행되는 안전성 활동은 실제 개발에 들어가기 전인 설계까지 안전성 확보를 저해하는 요인을 밝혀 제거할 수 있도록 설계하는 것이 매우 중요하다. 이는 개발에 필요한 시간과 비용을 대폭 줄일 수 있기 때문에 반드시 수행해야 하며 명확한 결과를 도출해야 한다.
그렇기 때문에 본 연구에서는 소프트웨어의 위험원을 보다 체계적이고 명확하게 도출하기 위해 위와 같은 방법을 적용하였으며, 다음과 같은 최종적인 결과를 얻었다.

- 명확한 분석범위 정의
- FTA의 기간 단축
- 체계적인 분석기법 적용
- 위험원 추적성 마련

5. 결론

안전성 확보를 위해서는 명확하게 장애 및 오류를 판별하고 그 원인을 도출하여 제거하거나 안전한 범위 이내로 감소 시켜야 한다. 그러나 '명확하게 장애 및 오류의 판별'과 '그 원인의 도출' 이 두 가지 과제부터 우선 완벽하게 해결해야만 안전성을 확보할 수 있게 된다. 이 과제를 해결할 수 있는 것이 분석기법의 적용이다.
일반적으로 최소 2~3가지의 기법을 적용하여 장애 및 고장에 대한 분석을 수행하게 되는데, 각각의 결과가 상호간에 관련성이 많이 떨어짐을 기존의 실제 사례를 통해 확인하였다. 그리하여 본 연구에서는 분석기법을 다양한 조건으로 분류하였으며, 개발하고자 하는 소프트웨어에 적합한 분석기법을 적용하여

분석가법간 상호 관련성을 가지고 유기적인 흐름에 대해 연구결과를 제시하였다.
PHA는 소프트웨어 전반적인 장애 및 오류를 찾아 FTA의 정상사상 결정에 기초를 제공하게 되며, HAZOP은 인적오류 및 운용의 오류에 대해 FTA의 중간 Event나 Gate에 기초를 제공하게 되며, FMEA는 소프트웨어의 하위 기능과 구성요소간의 메커니즘 장애에 대해 FTA의 중간 Event나 Gate에 기초를 제공하게 된다.
이렇게 모든 결과가 적용된 FTA는 보다 치밀한 구성으로 작성될 수 있으며, 그 결과에 신뢰성이 높아질 수 있다.
향후 이러한 분석기법의 결과에 대한 관리와 이를 증명 검토할 수 있는 방안에 대한 연구가 지속적으로 이루어질 예정이며, 국내 철도산업에서도 적용하기 적합한 방법을 제시할 것이다.

참고문헌

1. UK Yellow Book 3,4, 2000
2. Patricia Rodriguez Dapena, Software Safety Verification in Critical Software Intensive System.
3. 김병석, 나승훈, '시스템 안전공학', 2002