Journal of The Korea Society of Computer and Information
Vol. 24 No. 4, pp. 107-112, April 2019
www.ksci.re.kr
https://doi.org/10.9708/jksci.2019.24.04.107

The Video on Demand System Failure Evaluation of Software Development Step

Jin-Wook Jang*

Abstract

Failure testing is a test that verifies that the system is operating in accordance with failure response requirements. A typical failure test approaches the operating system by identifying and testing system problems caused by unexpected errors during the operational phase. In this paper, we study how to evaluate these Failure at the software development stage. Evaluate the probability of failure due to code changes through the complexity and duplication of the code, and evaluate the probability of failure due to exceptional situations with bugs and test coverage extracted from static analysis. This paper studies the possibility of failure based on the code quality of software development stage.

▸Keyword: video on demand, failure testing, failure evaluation, code complexity, code duplication, test coverage, Bug

I. Introduction

장애 테스트는 시스템이 장애 대응 요구사항에 맞춰 동작하는지를 검증하는 테스트로 시스템을 개발하고 운영하는 단계에서 주로 야기될 수 있는 결함을 대처하는데 목적이 있다. 또한 장애 테스트의 경우 운영단계에서 예기치 못한 오류로 인한 시스템의 문제점을 확인하고 테스트하는 것으로 운영 시스템을 대상으로 접근한다.

특히 SI(system Integration)프로젝트의 개발과정에서 장애를 예측하고 사전에 품질을 고려한 개발을 추진하기에는 비용 및 일정부분에서 많은 어려움이 있다.

기존 연구에서는 장애평가를 위하여 운영시스템을 대상으로 장애 이력을 분석하거나 장애원인을 분석하는 방식이었으나 본 연구에서는 개발과정의 시스템을 대상으로 장애 가능성을 평가하였다는 점에서 차별성이 있다.

본 논문에서는 주문형 비디오 서비스(video on demand) 시스템의 소프트웨어 개발과정 중 코드의 복잡도, 코드의 중복평가, 정적분석 평가, 테스트 커버리지 평가의 4가지 요소를 기반으로 장애 가능성을 평가하였다.

본 논문은 소프트웨어의 개발단계에서 품질지표를 기반으로 장애를 예측할 수 있는 기준을 제시하고 예방할 수 있다. 제시된 4가지 기준으로 산출된 수치를 통하여 장애발생가능성을 수치화하여 기준으로 활용할 수 있으며 지속적인 소프트웨어 제품품질 향상활동에 활용할 수 있다[4].

II. Preliminaries

1. Related research

장애 테스트는 시스템이 장애가 발생했을 때 이에 대한 감지와 장애 복구 능력을 검증하는 테스트이다. 기업의 서버 시스템의 경우 여러 대의 서버로 구성되어 일부 시스템이 장애가 발생하더라도 거래정보가 유실이 없이 작동해야 한다. 이에 대한 테스트가 장애 테스트이다.

또한 장애가 발생할 수 있는 곳에 부하를 주거나 일부 시스템을 비정상 종료 시키는 등의 방법으로 강제적인 장애를 발생시킨다. 이처럼 장애가 발생한 경우에도 시스템이 계획된 것처럼 동작하는 지를 체크하게 된다[1,3].

장애의 유형은 여러 가지가 있지만 대표적으로 아래와 같은 장애의 유형에 따라 테스트를 진행한다.

첫째, 미들웨어 장애 테스트로 서버 소프트웨어를 기동하기 위한 미들웨어로 예컨대, 데이터베이스, 웹서버, 어플리케이션, 서버 등이 장애가 났을 때에 대한 장애 대응능력을 검증한다. 이러한 형태의 장애 대응은 미들웨어 자체의 클러스터링 기능에 의해서 처리되는 경우가 많다. 미들웨어 장애 테스트는 부하테스트 중에 강제적으로 가동 중인 미들웨어 인스턴스를 종료시킨 후에 시스템으로의 요청이 실패 없이 처리되는 지를 검증한다.

둘째, 하드웨어 장애 테스트로 하드웨어에 대한 장애 테스트

• First Author: Jin-Wook Jang, Corresponding Author: Jin-Wook Jang
*Jin-Wook Jang (jinwjang@anyang.ac.kr), College of Liberal Arts, Anyang University, Anyang, Korea
• Received: 2019. 01. 28, Revised: 2019. 03. 10, Accepted: 2019. 03. 26.
www.kci.go.kr

를 들 수 있는데, 하드웨어 장애가 발생했을 때의 대처 능력을 검증한다. 해당 하드웨어의 전원을 강제적으로 내림으로써 인위적으로 장애를 발생시킨다.

셋째, 네트워크 장애 테스트로 네트워크 장애는 각 서버간의 연결 장애에 대한 대응 능력 검증으로 네트워크 선을 스위치나 서버에서 강제적으로 뽑아서 장애를 발생시키고 테스트를 수행한다.

이러한 장애에 대한 대처는 하드웨어나 소프트웨어에 대한 이중화 구성을 통해서 이루어지는 것이 일반적이다.

장애가 났을 때 장애가 나지 않은 나머지 시스템으로 요청을 전달하여 처리하는 것을 시스템 대체 작동(folivore)이라고 한다. 장애가 발생했을 때 테스트를 했다면 반대로 장애가 복구되었을 때도 테스트를 수행해야 하는데 이를 장애복구(fallback)라고 하고 이에 대한 검증 테스트를 장애 복구 테스트라고 한다. 장애 복구 테스트는 장애 테스트와 역순으로 진행하는데 서버를 강제 종료 시켰다면 서버를 재 기동 시키거나 네트워크 선을 뽑았다면 다시 연결해서 상황이 정상화되었을 때 장애가 났던 시스템이 다시 정상적으로 동작하는 지를 확인한다.

소프트웨어의 장애를 평가하는 방법으로 FMEA(failure mode and effect analysis)가 활용되고 있으며 이는 시스템을 사용하는 중에 발생하는 사고와 원인의 관계를 계열적으로 해석하는 신뢰성 해석수법의 한 가지 방법이다. 고장형태 영향분석이라고도 한다.

시스템요소를 포함한 기계부품의 고장이 기계 즉 시스템 전체에 미치는 영향을 예측하는 해석방법으로 기계부품 등의 기계요소가 고장을 일으킨 경우에 기계 전체가 받는 영향을 규명해 나가는 것이다[2].

FMEA에서는 예상되는 고장빈도, 고장의 영향도, 피해도 등에 관하여 평가기준을 설정해두고 개개의 구성요소에 대하여 고장 평가를 하고 이것을 종합하여 지명도를 구한다. 치명도가 높을수록 중점적인 관리가 필요하다[8].

기존 연구에서도 코드의 복잡도, 코드의 중복을 통한 품질지표로 코드의 라인 수(line of code), 사이클로 매틱(cyclomatic), 할 스테드 매트릭스(halstead metrics), 객체지향 매트릭스(object metrics) 등이 활용되고 있었으며 만 듯이 복잡도가 높다고 해서 경함의 발생하는 것은 아니나 연관관계를 찾으려는 연구가 이루어 졌다[4].

III. The Proposed Scheme

1. Design of Failure Model

본 논문의 주문형 비디오 서비스 시스템의 소프트웨어의 장애가능성을 판단하기 위하여 소프트웨어 장애를 분석하고 가능성을 평가한다.

이를 위하여 장애 발생하는 원인을 정의하였다. 첫째, 부적절한 변경에 의한 장애 발생은 개발 측면에서 개발 복잡도가 높은 경우와 유사한 시스템을 개발하는 과정에서 발생될 수 있는 중복 정도가 높은 경우로 평가한다.

또는 원거리 시스템과의 연동개발에서 발생되는 IP 변경에 대한 오류, IP 변경의 적용이 중복되었기에 발생했다고 볼 수 있다. 둘째, 소스코드 및 환경의 변화가 없는데도 장애가 발생하는 경우로 기존 소스코드에 있던 예외 및 추가 로 인하여 발생하는 경우가 있다.

그리고 시스템 공격 또는 다른 장애에 의한 경우로 개발단계별 단위테스트, 통합테스트, 시스템테스트, 인수 테스트가 수행되었는지 와 테스트 커버리지(test coverage)를 기준으로 위험도를 평가한다. 또는 정적 분석을 통해 예외 발생할 가능성을 확인하고 심각한 버그(critical bug)가 얼마나 있는 지로 평가한다. 그리하여 본 논문에서는 Fig. 1 과 같이 실행과정에서 테스트되지 않은 경로(untested path)를 통하여 발생가능 한 장애를 대상으로 연구 하였다.

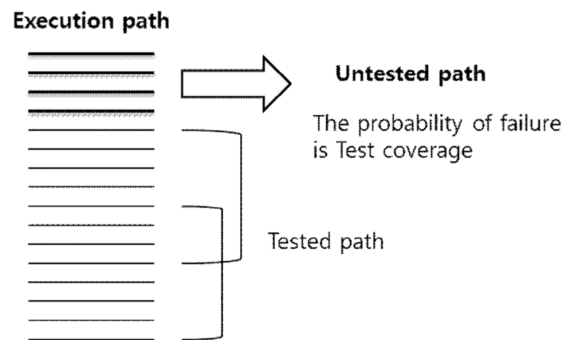

Fig. 1. presumes of Failure evaluation

다음 Table 1 와 같이 장애의 원인을 정의하고 분석 대상으로 하였다.

Table 1. Causes of Failure

Division	Contents
Causes of Failure1	Failure is caused by untested execution paths
	Indicator 1) Code Complexity
	Indicator 2) Code Duplication
Causes of Failure 2	Failure by change
	Indicator 3) Critical Defect of Static Analysis
	Indicator 4) Test Coverage

주문형 비디오 서비스의 장애발생 가능성을 분석하기 위하여 코드의 복잡도, 코드의 중복평가, 정적분석 평가, 테스트 커버리지의 4가지 항목을 정량화하기 위하여 다음 Table 2 과 같이 정의하였다.

www.kci.go.kr

Table 2. Criteria

Division	Complexity	Duplication	Static Analysis	Test Coverage
Number	E_1	E_2	E_3	E_4
probability	P(KPI_1 | E)	P(KPI_2 | E)	P(KPI_3 | E)	P(KPI_4 | E)

본 연구에서는 주문형 비디오 서비스 시스템을 대상으로 개발과정에서 장애를 평가하고 대응하기 위하여 다음 가정에 근거하여 장애 위험도 평가 기준을 Fig. 2 같이 코드의 복잡도, 중복, 정적분석, 테스트 커버리지의 4가지 항목으로 가정하고 하였으며 본 연구의 소프트웨어 제품품질 측정 범위로 정의하였다[3].

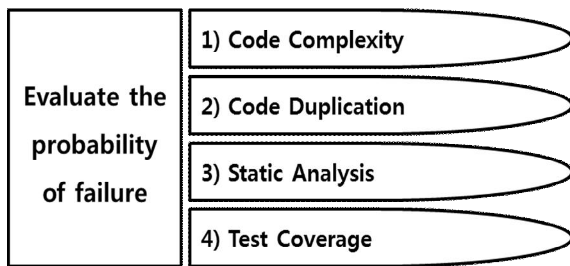

Fig. 2. Model of Failure evaluation

첫째, 코드의 복잡도(code complexity)가 상대적으로 높으면 코드 변경에 따른 장애발생 가능성이 높다. 둘째, 중복된 코드(duplication)가 많으면 코드 변경에 따른 장애발생 가능성이 높다. 셋째, 정적 분석에 의해 발췌된 문제를 해결하지 않을수록 예외 상황에 따른 장애가 발생할 가능성이 높다[5].

넷째, 테스트 커버리지(coverage)가 충분하지 못하면 예외 상황에 의해 장애가 발생할 가능성이 높다.

그동안 발생한 장애 이력의 분석을 통해 장애 위험도의 평가 기준을 다음과 같이 설정했다.

첫째, 장애에 영향을 미치는 측정값은 코드의 복잡도, 코드의 중복, 버그(bug), 코드 커버리지로 정의한다. 둘째, 측정값이 장애에 영향을 미치는 정도를 정의한다[4]. 셋째, 측정값에 따른 장애 가능성 함수를 정의하며 함수는 다음과 같이 소스코드 파일단위로 코드의 복잡도, 중복, 버그, 테스트 커버리지를 중심으로 이해관계자와 개발단계별 개발목표를 고려하여 Table 3의 장애영향도 가중치를 적용하였으며 계산 formula 1 과 같다. 이때 가중치의 경우 시스템의 개발목표를 고려하여 복잡도, 중복정도, 버그, 커버리지 순으로 정의하였다.

Table 3. Weight of Failure

Section	Weight
coverage	0.4
duplication	0.3
bug	0.2
coverage	0.1

formula 1:

$$R_{FILE}(f) = W_{CC} \times R_{CC}(f) + W_{DUP} \times R_{DUP}(f) + W_{BUG} \times R_{BUG}(f) + W_{COV} \times R_{COV}(f)$$

넷째, 어플리케이션(App)의 장애 심각도를 정의하였다. 어플리케이션의 장애 심각도를 고려하여 전체 장애 위험도를 계산하였으며 다음 formula 2와 같이 정의하였다.

formula 2:

$$R(UI\ 5.0) = \sum_{HE\,h} R_{HE}(h) = \sum_{HE\,h} \sum_{APP\,a} R_{APP}(a) = \sum_{HE\,h} \sum_{APP\,a} S_{APP}(a) \times \sum_{FILE\,f} R_{FILE}(f)$$

2. Application of Model

주문형 비디오 서비스 시스템의 소프트웨어 개발단계에서 임의의 부분을 선택하여 사이클로 매틱 복잡도를 Fig. 3와 같은 방식으로 분석하였다. Table 4 기준으로 볼 때 전체 복잡도는 970으로 매우 높다.

코드의 복잡도에 따른 기준으로 Table 4 과 같이 정의할 수 있으며 본 사례 연구에서는 이해관계자와 프로젝트의 단계별 목표를 고려하여 50이하를 목표로 진행 하였다[6,7].

Table 4. Code complexity criteria

complexity	Explanation	Risk
1-4	A simple procedure	Low
5-10	A well structured and stable procedure	Low
11-20	A more complex precedure	Moderate
21-50	A complex procedure, alarming	High
>50	An error-prone, extremely troublesome, untestable procedure	Very high

주문형 비디오 서비스의 구매 관련 scs-purchase release 모듈의 경우 전체 복잡도는 Table 4 기준으로 볼 때 4122 이며 특정 세부 모듈의 경우 최대 462로 매우 높다. 구조적인 코딩 규칙정의와 특정 부분에 있어 메소드의 이름을 지어 별도의 메소드로 정의하거나 삭제하는 등 의 리팩토링(refactoring) 필요하다. 다음 Fig. 3은 특정 모듈의 코드로 복잡도와 중복정도를 유발 시키는 부분을 체크한 예이다.

www.kci.go.kr

```
logger.debug("nowDate : "+ nowDate);
logger.debug("frDate.compareTo(nowDate) : "+frDate.compareTo
// 판매 가능한 상품 중 PrdPrcFrDt <= nowdate and PrdPrcToDt
if (frDate.compareTo(nowDate) <= 0 && toDate.compareTo(nowD
productInfoDTO.getAsisUseYn()) && "N".equals(productInfoDTO.
    if ("Y".equals(productInfoDTO.getUseYn())){
        if((CommUtil.TYPE_PPV.equals(productInfoDTO.getAsisP
        ()) && ((reqUserDto.getServiceCode().equals(productI
        equals(productInfoDTO.getCugYn()))))) {
        // 유 무료 Check (PPV 상품이면서 사용자와 동일한 패
        if ("10".equals(productInfoDTO.getAsisPrdTypCd()) &&
            reqUserDto.getIdPackage().equals(productInfoDTO.
            productFreePid = productInfoDTO.getPrdPrcId();
        }
        // IPTV 시청 권한이 없는 고객이 디펜드 상품 구매 시
        if (("20").equals(productInfoDTO.getPrdCompoCd()) ||
            if (("35").equals(productInfoDTO.getPrdTypCd()))
                iptvSetFlag = true; //IPTV 상품

                if("1".equals(reqUserDto.getIptvStatusCode(
                    return new CtsInfoAllIFRespDTO(EnumMessa
                    없습니다.\n다른 상품을 선택 해 주세요(46
                }
            }
        }
        // logger.debug("======판매 가능 상품 ==========");
        // 판매 가능 상품 묶기
        if(reqProductSalesList == null ){
            reqProductSalesList = new ArrayList<ProductInfo
        }
        reqProductSalesList.add(productInfoDTO);
        }

    // 3.2 조회된 전체 상품의 PID 리스트 묶기 (기 구매 Check
    if(reqProductIptvPidList == null ){
        reqProductIptvPidList = new ArrayList<String>();
    }
    if(reqProductPpmPidList == null ){
        reqProductPpmPidList = new ArrayList<String>();
    }
    if(reqProductEtcPidList == null ){
        reqProductEtcPidList = new ArrayList<String>();
```

Fig. 3. part of Source Code

중복된 코드는 평균 0.12로 주문형 비디오 서비스 시스템의 scs-gwsvc release 모듈의 경우 23.80%, scs-purchase release 모듈의 경우 39.70% 타 헤드엔드(HE) 모듈 평균대비 높다. 코드의 복잡도와 중복도가 높은 헤드엔드 대상으로 공통모듈을 뽑아 모듈화 하여야 한다. 버그는 평균 6.6개, 대상시스템의 scs-purchase release 모듈의 경우 24개로 평균대비 높으며 수정이 필요하다.

구문커버리지(state coverage)를 Fig. 4 와 같이 상용 정적분석 도구인 소나큐브(sonarqube)를 이용하여 분석해본 결과 평균 49.9%이며 결정커버리지(decision coverage)는 평균 37.3%이다, 커버리지의 경우 높을수록 좋으므로 개발단계별 목표를 고려하여 50%이상의 목표 커버리지를 설정하여 진행하였다.

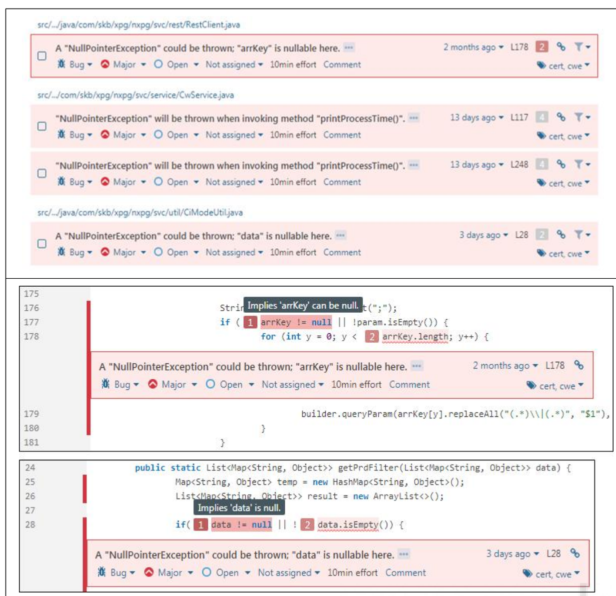

Fig. 4. static analysis

본 사례는 국내 대형 시스템 통합 사업으로 Table 5 과 같은 내부 소프트웨어 제품품질 기준을 제시하고 있다. 기업의 사업영역과 품질목표에 따라 재정의의 할 수 있으며 이러한 기준이 개발과정에서 꾸준히 실행되고 관리 될 수 있도록 많은 노력이 필요하다.

Table 5. Self Standard

Divi.	formula	step	Aim	Result
Rate of Defect	(Action defects/Register ed Defects)*100	integration test	97% or more	Test Result
Defect level	Defects	BMT	cases	Test Result
statement coverage	Measure by tool	Test	50% or more	SonarQ ube

각 헤드엔드 별 코드크기(line of code), 복잡도, 중복, 버그, 테스트 커버리지 정보를 기준으로 Table 3의 가중치를 반영하여 복잡도, 중복, 버그, 정적분석 값을 fomula 1 을 이용하여 계산하였으며 종합적으로 리스크(risk)를 식별하였다. 다음 Table 6은 산출된 값을 보여준다.

Table 6. Source Analysis Result

Div.	LOC	CC	DUP	BUG	SC	BC	RISK
HE1	3250	456	1217	10	1%	0%	0.20
HE2	1921	462	510	0	2%	0%	0.16
HE3	1276	228	906	0	18%	21%	0.14
HE4	1294	260	786	0	0%	0%	0.15

※ SC: statement coverage, BC: branch coverage

장애 이력을 통해 장애 위험도의 평가 기준을 다음과 같이 설정했다. 첫째, 장애에 영향을 미치는 측정 값을 결정한다. 둘째, 측정값이 장애에 영향을 미치는 정도를 정한다. 셋째, 측정값에 따른 장애 가능성 함수를 정한다. 마지막으로 어플리케이션의 장애 심각도를 정한다.

위와 같은 방법으로 앞장에서 제시된 formula 1을 이용하여 장애 위험도 평가 결과 7.56 점으로 계산하였다. 이는 제한연구에서 제시된 4가지 항목을 기준으로 산출되었으며 차후 시스템 운영 단계에서 이 수치를 상대적으로 높으면 장애발생가능성이 높으며 낮으면 장애가능성이 낮아질 수 있는 판단치의 근거로 활용이 가능하다. 그러니 이 수치는 개발과정 중에 계산된 값으로 운영단계별 데이터를 누적하여 실효성 있는 기준 값이 될 수 있도록 지속적인 조정이 필요하다.

이는 첫째, 복잡도가 높고, 코드 중복이 많아 변경에 의한 장애 발생 가능성이 높으며 둘째. 정적 분석 결함이 많고, 테스트 커버리지가 낮으며 예외 상황에 의한 장애 발생 가능성이 높음을 설명하는데 활용이 가능하다. 어플리케이션 별 장애심각도 평가하기 위하여 formula 2를 이용하여 앱 별 산출하였으며 각 헤드엔드별 결과 값은 Table 7 와 같다.

www.kci.go.kr

Table 7. Weight of Failure

HE	App	Value
HE1	APP1	0.03
HE1	APP2	0.03
HE1	APP3	0.12
HE1	APP4	0.03
HE1	APP5	0.03
HE2	APP6	0.2
HE2	APP7	0.03
HE2	APP8	0.2
HE2	APP9	0.15
HE2	APP10	0.03
HE3	APP11	0.03
HE4	APP12	0.12

3. Improvement Plan

코드의 복잡도가 전체적으로 매우 높은 편이며 중복된 코드 중 필요한 부분과 불필요한 부분에 대한 선별 작업이 우선 되어야 하며 요약하면 다음과 같다.

첫째, 버그의 경우 코드를 중심으로 변경과정에서 발생된 경우로 코드를 수정한 후 관련되는 부분을 포함하여 동료검토가 필요하다.

둘째, 커버리지의 경우 도구에 의해 측정된 값으로 높을수록 좋으나 Table 8 과 같이 내부 목표치를 정의하여 추진하여야 한다.

셋째, 제시된 우선순위의 경우 품질목표에 따라 조정이 가능하며 프로젝트 상황을 고려하여 달성이 용이한 항목을 우선적으로 검토하고 실행하는 것이 필요하다.

우선순위를 프로젝트의 일정과 비용, 품질을 고려하여 반영하였으며 프로젝트의 상황에 따라 지속적인 검토가 필요하다.

Table 8. Suggestion of improvement plan

Div.	Improvement Plan	Priority
Code Complexity	Structured coding rule definition required Need to module review	Middle
Code Duplication	Structured coding guide	Middle
Static Analysis	All corrective actions	short-term
Test Coverage	need tp 50% more coverage	short-term

V. Conclusions

본 논문은 소프트웨어 프로젝트의 개발 초기부터 장애 위험도를 평가하고 관리함으로써 서비스 품질을 사전에 확보한다는 점에서 의의를 가진다. 본 논문에서 사용한 측정값인 코드의 복잡도, 코드의 중복, 버그, 코드 커버리지는 대표적인 소프트웨어 품질 지표를 제시하였다. 이를 통하여 제시된 지표의 평가 모델은 차후 시스템이 개발되는 과정에서 소프트웨어 품질 기준으로 활용이 가능하다는 점에서 의미를 가진다.

추가적으로 이 모델을 발전시키기 위하여 기존에 운영 중인 시스템의 장애이력 분석을 통해 장애에 영향을 미치는 측정대상을 추가하여 각각의 기준치에 좀 더 의미를 부여한다면 좀 더 현실적인 장애발생 가능성을 예측할 수 있다. 나아가 개발이 완료되고 운영단계에서 예측된 유형의 결함이 발견되는지도 비교해 볼 수 있을 것이다.

또한 본 논문에서의 제시한 장애 위험도 평가를 유사한 환경과 목적으로 운영되는 시스템을 조사하여 비교함으로 객관성을 제고하려는 노력도 필요하다.

본 논문에서 제시하는 장애 위험도 평가 모델은 개발단계의 코드 품질관리를 통하여 장애를 예측하여 운영단계의 시스템 위험도를 경감한다는 관점에서 연구의 의의를 가진다.

REFERENCES

[1] ByungRae Cha, MyeongSoo Choi, Sun Park, JongWon Ki,"Verification Test of Failover Recovery Technique based on Software-Defined RAID", Smart Media Journal, Vol.5, No.1, 2016.

[2] Park Gee-Yong, Kim Dong Hoon, Lee Dong Young, "Software FMEA analysis for safety-related application software", 96p ~ 102p, Elsevier, 2014.

[3] TTA, “Software Safety Inspection Guide”, 2016.

[4] Alan, Johnston, Ken, "how we test software at microsoft", Microsoft 2008.2.

[5] T. McCabe, “A Complexity Measure,” IEEE Trans. Software Eng.,vol. 2, no. 4, pp. 308-320, Dec. 1976.

[6] Tim Menzies, Member, IEEE, Jeremy Greenwald, and Art Frank, “Data Mining Static Code Attributes to Learn Defect Predictors”, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 32, NO. 11, JANUARY 2007.

[7] Alan, Ken, Johnston, "How We Test Software at Microsoft", Microsoft Pr, 2008.

[8] https://quality-one.com/fmea/

www.kci.go.kr

Author

Jin-Wook Jang received in Ph.D. of Management Engineering from Konkuk University. The main interests are project management, software quality, testing process, etc. Dr. Jang worked in Korea Defense Intelligence Command of Ministry National Defense as a Computational officer and in SK communications as a PMO Manager. He is Professor of College of Liberal Arts, Anyang University

www.kci.go.kr