Journal of KIIT. Vol. 16, No. 6, pp. 9-24, Jun. 30, 2018. pISSN 1598-8619, eISSN 2093-7571 9
http://dx.doi.org/10.14801/jkiit.2018.16.6.9

DEMATEL 기법과 내용분석을 이용한 임베디드 소프트웨어 결함요소 간의 영향 관계 분석

허상무*, 김우제**

Analysis of the Casual and Effect Relationship for Embedded Software Defects Using DEMATEL and Content Analysis

Sang-Moo Huh*, Woo-Je Kim**

요 약

임베디드 소프트웨어는 순수 소프트웨어 영역과 하드웨어를 제어하는 소프트웨어 영역이 있다. 임베디드 소프트웨어는 순수 소프트웨어와 마찬가지로 결함을 내포하고 있고, 결함으로 인하여 치명적인 사고가 발생될 수 있다. 순수 소프트웨어 분야에서는 핵심 결함을 제거하는 것이 효율적이고 효과적인 개발 방법론이라고 한다. 하지만 임베디드 분야에서는 체계적이고 통합적인 관점으로 임베디드 소프트웨어 결함을 도출한 연구는 없고, 연구자가 관심이 있는 분야에 대해서만 편향적으로 분석한 연구가 주를 이루었다. 임베디드 소프트웨어 결함에 대하여 인과관계를 분석하여 핵심 결함을 도출할 수 있다면, 핵심 결함을 집중적으로 관리하여 좀 더 안전한 임베디드 소프트웨어를 제작할 수 있을 것이다. 이에 요소 간의 인과관계를 산정할 수 있는 DEMATEL 기법을 이용하여 결함 간의 인과관계를 산정하고, 임베디드 소프트웨어의 핵심 결함을 도출하였다. 그렇지만, 결함에 대한 중요 가중치를 도출하지는 못했다. 향후 연구에는 결함을 가중치를 도출하는 연구가 남아 있다.

Abstract

Embedded software has a pure software area and a software area that controls hardware. Embedded software has defects to be inherent like pure software and can cause fatal accidents due to that. In the pure software field, eliciting and managing core defects is called efficient and effective development methodologies. Whereas, in embedded software fields, there are no studies that have developed embedded defects from a systematic and integrated perspective. Mainly, there are only biased studies that researchers examined for areas of their interest. If it is possible to calculate the significance of embedded software defects, it will be able to increase the quality of embedded software efficiently by intensively managing the core defects. To prove this, the DEMATEL method that estimate the causal relationship between elements was applied. Finally, the causal relationships between embedded software defects was derived and the core defects of embedded software was derived. However, we could not been to derive the weights for defects. And the future research, remains.

Keywords
embedded software defects, content analysis, bootstrapping, dematel, casual and effect, core defects

* 서울과학기술대학교 산업정보시스템과 박사수료
- ORCID ID: https://orcid.org/0000-0003-1405-0626
** 서울과학기술대학교 글로벌융합산업공학과 교수 (교신저자)
- ORCID ID: https://orcid.org/0000-0002-1638-645X

· Received: Mar. 27, 2018, Revised: May 02, 2018, Accepted: May 05, 2018
· Corresponding Author: Woo-Je Kim
Dept. Dept. of Convergence of Industrial Engineering, Seoul National University of Science and Technology, 232 Gongneung-ro, Nowon-gu, Seoul, 01811, Korea
Tel.: +82-2-970-6449, Email: wjkim@seoultech.ac.kr

Ⅰ. 서 론

임베디드 소프트웨어는 자동차, 철도, 항공, 조선, 우주 등의 분야와 건설, 의료, 국방무기체계에 이르기까지 적용범위가 확대되고 있다. 임베디드 소프트웨어는 순수 소프트웨어와 마찬가지로 결함이 내재되어 있고, 결함으로 인하여 치명적인 사고가 발생하기도 한다. 의료계통에서는 방사선 치료기의 safety-critical 시스템의 결함으로 약 2년 동안 6건 이상의 과도한 방사선 방출로 인한 인명사고가 발생하였다. 나사 화성 탐색로봇인 패스파인더호가 화성에 착륙한 지 며칠 만에 시스템이 재기동이 되는 현상이 발생하였으며, 결국에는 250일 만에 통신이 두절되었고, 원인은 공유메모리 사용에 결함이 있는 것으로 밝혀졌다. 항공분야에서는 보잉 737기가 그리스 칼라모스 부근에서 이상이 발생해, 115명의 여행객과 승무원이 전부 사망하는 사건이 발생했었다. 이와 같이 역사 속의 중대한 시스템 사고의 이면에는 임베디드 소프트웨어의 결함에 의한 사고가 그것의 중심에 있다고 해도 과언이 아니다.

또한, 다양한 산업 분야에서 임베디드 소프트웨어가 사용되면서 소프트웨어의 위협도 높아져서 소프트웨어 오동작에 대한 안전성 위협이 큰 이슈로 부각되고 있다. 그로 인하여 결함을 체계적으로 제거하기 위하여 HAZOP(Hazard and Operability)기법이나 FMEA(Failure Modes and Effects Analysis) 기법 등을 이용하여 결함 원인과 영향, 상대적인 중요성 등을 평가하여 체계적으로 결함을 제거하려고 노력하고 있으며, 다양한 테스트 기법을 적용하여 임베디드 소프트웨어 내재된 결함을 제거하고 있다.

임베디드 소프트웨어는 일반 소프트웨어와 유사한 제어 논리가 있고, 일반 소프트웨어에서는 찾아보기 힘든 하드웨어를 밀접하게 제어하는 소프트웨어가 있다. 순수 소프트웨어 분야에서는 다양한 결함을 수집하여 핵심 결함을 도출한 연구가 있다. 이 연구에서는 소프트웨어에 내재된 결함은 전부 제거할 수 없으므로 핵심 결함을 집중적으로 제거하여 결함을 최소화하는 것이 효율적이고 효과적으로 소프트웨어를 개발하는 방법이라고 제시한다[1]. 하지만 임베디드 소프트웨어 결함에 대하여 조사한 결과, 체계적이고 통합적인 관점으로 분석한 연구는 없고, 연구자가 관심이 있는 분야에 대해서만 편향적으로 임베디드 결함을 분석한 연구가 주를 이루었다.

선행 연구에서는 임베디드 소프트웨어 결함에는 어떤 결함이 존재하고, 결함 간에는 어떻게 영향을 미치고 있으며, 어느 결함이 핵심 결함인 지에 대한 연구는 미흡한 것으로 파악되었다. 이에 본 연구에서는 기존에 연구자들이 연구한 임베디드 소프트웨어 결함을 수집하여 분류하고, 요소 간의 인과관계를 도출할 수 있는 데마텔(DEMATEL) 기법을 적용하여, 임베디드 소프트웨어 결함 간의 인과관계를 도출하고 핵심 결함을 도출하도록 한다.

임베디드 소프트웨어 결함 별로 인과관계를 산정할 수 있다면, 임베디드 프로젝트를 시작하기 전에 프로젝트 특성에 맞게 핵심 결함을 도출할 수 있을 것이다. 핵심 결함을 산정할 수 있다면, 좀 더 효율적이고 효과적으로 결함을 관리할 수 있을 것이며, 임베디드 소프트웨어 품질을 좀 더 용이하게 향상시킬 수 있을 것이다.

제 2장은 관련연구와 이론적 배경을 살펴보고, 제 3장은 연구 설계와 결과를 도출하고, 제 4장에서는 도출된 결과를 이용하여 임베디드 소프트웨어 결함 간의 인과관계와 핵심 결함을 분석하고 제 5장에서는 결론을 제시하였다.

Ⅱ. 관련연구와 이론적 배경

2.1 선행연구의 고찰

임베디드 소프트웨어 결함은 순수 소프트웨어 결함과 하드웨어와 제어하는 소프트웨어 결함이 존재한다. 우선, 순수 소프트웨어 결함에 대해서 살펴보면, 소프트웨어 결함 가중치 연구에서는 표 1과 같이 기관, 기업 및 연구자들이 소프트웨어 결함을 연구하였고, 연구된 결함을 수집하여 표 2와 같이 결함을 분류하였다. 그리고 인사, 급여, 회계 등 일반 어플리케이션에 대한 결함 간의 영향관계를 분석하여 ANP(Analytic Network Process)기법을 적용하여 핵심 결함을 산정하였다.

표 1. 순수 소프트웨어 결함의 출처
Table 1. Sources of pure software defects

Researchers	Software defects
IEEE 1044[2]	IEEE 1044-1993 defect standard
IBM의 ODC[3]	ODC(Othogonal Defect Classification)
HP Huber JT[4]	HP Defect Origins, Types & Modes
wikipedia[5]	Wikipedia Software Defect
Many foreign researcher & Korean researchers	

표 2. 선행연구에서 정의한 순수 소프트웨어 결함
Table 2. Pure software defects defined in prior study

Category	Sub defects
Logic	Condition statement
	Rotation logic
	Concurrent logic
Computation	Division by zero
	Expression
	Precision loss
I/F, Timing	External I/F
	Wrong sub Function (Internal interface)
	I/O Timinig
Data	Data structure
	Data usage
	Data value self

이 연구에서는 프로젝트 수행 시 막연한 결함관리 보다는 핵심결함을 집중적으로 제거하면, 좀 더 효율적이고 효과적으로 소프트웨어 품질을 향상시킬 수 있다고 한다[1]. 이 연구는 순수 소프트웨어 결함에 대하여 인과관계를 분석하여 핵심결함을 도출한 연구로서 본 연구와 유사한 면이 있다. 하지만, 이 연구는 임베디드 소프트웨어 결함에 대한 연구는 아니다. 이 연구에서 정의한 순수 소프트웨어 결함은 체계적이고 통합적으로 분류했기 때문에 본 연구의 순수 소프트웨어 결함으로 활용하기로 한다.

다음으로 하드웨어를 제어하는 임베디드 소프트웨어 결함에 대한 연구를 살펴보면, 임베디드 시스템의 결함에 대하여 소프트웨어 테스트 방법을 제시한 도서가 있다. 이 도서에서 언급되고 있는 결함은 데이터 계산 결함, 구조적 논리 흐름, 하드웨어와 소프트웨어 간의 인터페이스, 시간 결함, 소프트웨어 오류 복구, 성능 테스트, 휴먼 인터페이스, 모바일/임베디드 보안 등 다양한 결함에 대하여 테스트하는 방법을 제시하였다[6]. 이 도서의 내용은 임베디드 시스템을 테스트할 때 유용하게 활용할 수 있을 것으로 보인다. 그렇지만, 임베디드 결함 간의 인과관계와 핵심결함을 도출하려는 연구는 아니다.

임베디드 소프트웨어의 중요한 열 가지 결함은 경쟁조건, 재진입할 수 없는 함수, 스택 오버플로우, 메모리 힙, 메모리 누수, 데드락(Deadlock), 우선순위 역전 같은 결함이라고 제시한 연구가 있다[7][8]. 하지만 이 연구도 임베디드 결함 간의 인과관계를 이용하여 핵심결함을 도출한 연구는 아니다.

임베디드 소프트웨어를 함체에 탑재하기 전에 대상 플랫폼 및 기능이 정확하게 동작하는 지 테스트하는 평가모델을 수행한 연구가 있다. 이 연구에서는 태스크 관리, 태스크 간 통신, 시간관리, 인터럽트, 예외처리, 메모리관리, 입출력관리, 네트워킹, 파일 시스템 등에 대하여 평가항목을 이용하여 테스트를 수행하였고, 수행 전에 문제가 없었다고 판단된 시스템에 대하여 결함을 발견하기도 하였다[9]. 이 연구는 아홉 가지 임베디드 결함에 대하여 평가항목을 마련하여 결함을 도출하는 연구로서 결함을 충실하게 평가할 수 있는 연구로 보인다. 하지만 이 연구에서 평가한 아홉 가지 임베디드 결함에 대한 출처가 명확하지 않고, 결함의 중요도를 고려하지 않았으며, 단순히 결함을 발견하는 측면으로만 진행한 연구로서, 결함 간의 인과관계를 도출하여 핵심결함을 도출하려는 연구는 아니다.

이외에도 여러 임베디드 소프트웨어 결함 관련 자료를 조사하였고 여러 연구자들의 임베디드 소프트웨어 결함을 접할 수 있었지만, 체계적이고 통합적인 관점으로 분석한 연구는 찾아보기 어려웠다. 주로 연구자가 관심이 있는 분야에 대해서만 결함을 분석한 연구들이 존재하였다.

이상과 같이 임베디드 소프트웨어 결함에는 어떤 결함들이 존재하고, 결함 간에는 어떤 인과관계가 있으며, 그로 인하여 핵심 결함은 어떤 것인지에 대한 연구는 미흡한 것으로 파악되었다. 이에 본 논문에서는 소프트웨어 결함과 임베디드 소프트웨어 결함을 수집하여 데마텔 기법을 이용하여 임베디드 소프트웨어 결함 간의 인과관계를 도출하고 중요결함을 도출하고자 한다.

2.2 실험방법의 이론적 배경

2.2.1 내용분석(Content Analysis) 기법

임베디드 결함을 살펴보면, 용어는 다르지만 의미가 동일한 경우가 있고, 연구자들 간의 분류가 불일치하는 경우가 있다. 내용분석 기법은 분석할 용어의 특성, 의미의 범주를 기준으로 분류하고 통합하는 기법이다[10]. 내용분석은 허니의 내용분석(Honey's Content Analysis)기법과 부트스트래핑(Bootstrapping Content Analysis)기법이 있다. 허니의 내용분석은 요소들 간의 차이를 유사도로 분류하는 기법이다. 본 연구에서 적용한 부트스트래핑 기법은 표 3과 같이 7단계를 통하여 수행한다.

부트스트래밍 기법은 연구자와 전문가가 각자 표 3의 절차를 따라서 용어를 분류한다. 분류한 이후에 서로 만나서 분류 관점을 토의하며 신뢰도 테이블(Reliability Table)를 이용하여 서로 일치하는 분류를 찾아낸다. 일치하는 분류는 합의된 것으로 채택하고, 일치하지 않는 것은 시간을 갖고 다시 분류한 후, 대면하여 분류 관점을 토의한다, 토의한 후 일치하는 분류는 합의된 것으로 채택하고 일치하지 않는 분류는 다시 수행한다. 이런 과정을 반복하여 합의된 분류를 도출한다. 합의된 분류에 대한 신뢰도는 분류지수(Categorization Index)를 이용하며 측정한다. 신뢰도는 대각선에 위치된 합의된 수치의 비율을 의미하며, 일반적으로 80%이상이면 양호하고 90%이상이면 우수하다고 한다[10].

표 3. 내용분석 절차
Table 3. Procedure of content analysis

Step	Process
1	Create a category for the first defect
2	Create a new category if the following a defect is different as a first defect
3	Distribute the following defect to similar categories
4	Combine and detach existing categories as needed to create new categories
5	Repeat until all defects are classified
6	The defects that can not be classified should be classified as miscellaneous
7	Proceed with the classification until the defects in the miscellaneous group are less than 5 % of the total.

2.2.2 데마텔 기법

데마텔은 1972년부터 1979년까지 제네바의 Battelle 기념연구소에서 복잡하고 얽혀있는 문제를 해결하기 위하여 개발되었다. 데마텔 기법은 구성요소들 간의 인과관계를 해결하는 가장 좋은 기법 중 하나로 알려져 있으며 여섯 단계를 통하여 인과관계를 도출한다[11].

첫 번째, 직접영향관계행렬(DRM, Direct Relation Matrix)을 도출하는 단계로서, 임베디드 소프트웨어 결함(i)이 다른 결함(j)에게 미치는 영향력을 m 명의 전문가에게 설문을 받은 후, 아래 수식을 이용하여 평균을 산출한다. 결함 요소들 간의 영향력을 설문으로 수집한다.

$$Ai = \frac{1}{m}\sum_{i=1}^{m} X_i \tag{1}$$

행렬 A는 전문가 m 명이 평가한 값을 산술평균한 직접영향관계행렬(DRM) 값이며, Ai는 i번째 전문가가 평가한 행렬 값이다.

두 번째, 행렬을 정규화하는 단계이다. 행렬 A에서 행의 합의 최대값과 열의 합의 최대값으로 나눈 값 중에 작은 값을 선택한 후 그 값으로 곱하여 정규화된 행렬 N을 구하는 단계로서 수식 (2)와 (3)과 같이 표현할 수 있다. A_{ij}는 행렬 A의 i번째 행의 j번째 열에 해당하며, i, j={1,2,3,...n}이다.

$$s = \min\left[\frac{1}{\max\sum_{j=1}^{n} A_{ij}}, \frac{1}{\max\sum_{i=1}^{n} A_{ij}}\right] \tag{2}$$

$$N = sA \tag{3}$$

세 번째, 전체 영향관계행렬(TRM, Total Relation Matrix)을 도출하는 단계이다. TRM을 T라고 했을 때, 단위행렬에서 N행렬을 뺀 값에 대한 역행렬이며 수식 (4)와 같이 표현할 수 있다.

$$T = N + N^2 + N^3 + ... + N^m = N(I-N)^{-1} \tag{4}$$

네 번째, 영향을 주는 요소들과 영향을 받는 요소들을 분리하는 단계이다. T 행렬에서 행(Row)의 합(D)과 열(Column)의 합(R)을 구하고 중심도(D+R)와 원인도(D-R)을 구한다. 원인도인 D-R>0이면 속성이 원인요소가 되고 D-R<0이면 속성이 결과요소가 된다.

$$D = [D_i]_{n\times 1} = \left[\sum_{j=1}^{n} T_{ij}\right]_{n\times 1} \quad (5)$$

$$R = [R_i]_{1\times n} = \left[\sum_{i=1}^{n} T_{ij}\right]_{1\times n} \quad (6)$$

다섯 번째, 인과관계도를 도출하기 위한 기준점(Threshold)를 선정해야 한다. 기준점은 T 행렬의 요소들의 평균에 의하여 구할 수 있고, 수식은 (7)과 같다[11].

$$\sigma = \frac{\sum_{i=1}^{n}\sum_{i=1}^{n}[T_{ij}]}{n} \quad (7)$$

마지막으로 기준점을 이용하여 T 행렬에 적용하여 인과관계도(Casual & Effect Diagram)를 도식화한다.

III. 연구 설계 및 결과

3.1 연구 모델 설계 및 연구 수행

본 연구는 임베디드 소프트웨어 결함을 수집, 분류하고 결함 요소 간의 인과관계를 이용하여 핵심 결함을 도출하는 연구이다. 연구 절차는 그림 1과 같이 데이터 수집, 용어 표준화, 용어 분류, 설문 수행, 결과도출단계로 구성되어 있다.

첫 번째, 데이터 수집단계는 선행연구에서 연구된 임베디드 소프트웨어 결함 및 중요 요소를 수집한다. 두 번째, 용어 표준화단계는 수집된 결함 요소들의 관점 또는 용어에 따라서 중복되거나 다르게 분류되는 오류를 배제하기 위하여 용어 표준화를 수행한다.

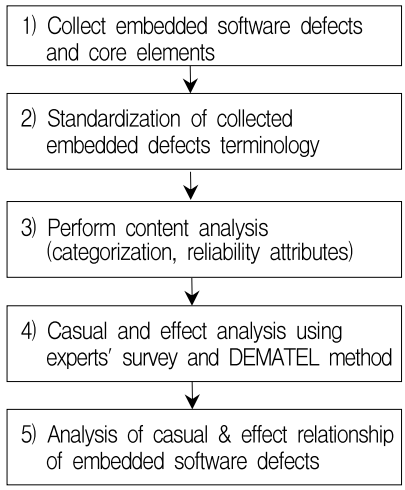

그림 1. 연구절차
Fig. 1. Research procedures

세 번째, 용어 분류단계는 표준화된 결함 용어를 기반으로 연구자와 전문가가 표 3과 같은 내용분석의 부트스트래핑 기법을 통하여 용어를 그루핑한다. 네 번째, 설문 수행단계는 그루핑된 결함에 대하여 설문을 통하여 전문가들의 견해를 수집하고, 데마텔 기법에 적용하여 결함 요소간의 인과관계를 도출한다. 마지막 결과도출단계는 결함요소 간의 인과관계도를 도식화하고 원인인자, 결과인자, 중심도 등을 산정한다.

3.2 결함요소 수집 및 논의

3.2.1 임베디드 소프트웨어 결함에 대한 논의

임베디드 소프트웨어 결함은 첫 번째 순수한 소프트웨어 측면의 결함을 수집해야 하고, 두 번째, 하드웨어를 제어하는 소프트웨어 결함을 수집해야 한다. 그렇지만, 본 연구는 하드웨어 자체 결함은 포함하지 않는다.

첫 번째, 순수 소프트웨어 결함은 표 1과 같이 기관과 연구자들에 의해 연구되었고, 소프트웨어 결함 가중치 연구[1]에서 표 2와 같이 순수 소프트웨어 결함을 분류하였다. 다음으로, 하드웨어와 밀접한 임베디드 소프트웨어 결함을 위하여, 다른 연구자들이 연구한 결함과 임베디드 시스템 아키텍처에서 중요 요소를 조사하였고, 표 4와 같이 수집되었다.

표 4. 임베디드 소프트웨어 결함 및 출처
Table 4. Embedded software defects and sources

<table>
<tr><th>Researchers</th><th colspan="2">Embedded defects & Core elements</th><th>Researchers</th><th colspan="2">Embedded defects & Core elements</th></tr>
<tr><td rowspan="2">Hagar, J. D.[6]</td><td colspan="2" rowspan="2">Data computation bug
Structural logic flow
S/W to H/W & H/W-to-S/W Interface
Long duration control
Logic and control law
H/W S/W communications bug
S/W error recovery
S/W-system fault tolerant
Data, Computation
Time-related, Human Interface</td><td>Barr, M[7]</td><td colspan="2">Race condition
Non-reentrant function
Missing volatile ketword
Stack overflow
Heap fragmentation</td></tr>
<tr><td>Barr, M[8]</td><td colspan="2">Memory leak
Deadlock
Priority Inversion
Incorrect priority assignment
Jitter</td></tr>
<tr><td rowspan="4">Leszak, M. and Perry, D. E.[12]</td><td rowspan="2">Implementation</td><td rowspan="2">Data design/usage
Resource alloc& usage
Exception handling
Algorithm
Functionality</td><td rowspan="2">Lutz, R. R. [13]</td><td>Program fault</td><td>Internal faults
Interface faults
Functional faults(conditional faults, incorrect condition or limit values)</td></tr>
<tr><td>Process flow</td><td>Interface spec</td></tr>
<tr><td rowspan="2">Interface</td><td rowspan="2">Data design/usage
Function design/usage
Communication protocol
Process coordination
Unexpected interations
Change coordination</td><td rowspan="3">Bennett, T. L., and Wennberg, P. W[14]</td><td>Function faults</td><td>Operating faults
Condition faults
Behavior faults</td></tr>
<tr><td colspan="2">Interface faults</td></tr>
<tr><td>External</td><td>Concurrent work</td><td colspan="2">Internal faults</td></tr>
<tr><td rowspan="2">Nigel Jones[15]</td><td colspan="2" rowspan="2">Initialization
Pointer dereferencing
Atomicity
Interrupt response times
Resource allocation mistakes
Priority/scheduling issues
Deadlocks
Priority inversion
Race conditions</td><td>Rodriguez-Dapena, P[16]</td><td colspan="2">Interface between component
Control flow between component
Internal functionality, Data structure
Control flow, Calculations
S/W to S/W interface
H/W to S/W Interface
Timing, User interface</td></tr>
<tr><td>Tammy Noergaard[17]</td><td colspan="2">Device Driver
Flash memory</td></tr>
<tr><td>Ji, S., and Bao, X[18]</td><td colspan="2">Initialize
Input
Interface
Output
Control
Fault detection
fault handle
performance</td><td rowspan="6">H.W. Jung[19]</td><td>Sensor</td><td>Reference conversion table error during A/D conversion
No S/W connection logic between sensor and system</td></tr>
<tr><td rowspan="5">Sung, A. et al.[20]</td><td colspan="2" rowspan="2">Data flow
Intratask
Intertask
Independent varialbe
Dependent variable
Process bug
Arithmetic bugs
Init bugs
Data flow bugs</td><td>Key</td><td>Reference conversion table error during A/D conversion
Key alone event</td></tr>
<tr><td>LCD</td><td>Display error</td></tr>
<tr><td>IND</td><td>Indication error</td></tr>
<tr><td>Buzzer</td><td>Buzzer error</td></tr>
<tr><td>Motor actuator+heater</td><td>A/D conversion error
D/A conversion error</td></tr>
<tr><td>Additional factors</td><td>Infinite loop</td><td>Hyunmi Choi et</td><td colspan="2">Control logic
Task management
Inter-task communication</td></tr>
</table>

<table>
<tr><td rowspan="2"></td><td>Structural Bugs</td><td>Control and Sequence bugs</td><td rowspan="2">al.[9]</td><td colspan="2" rowspan="2">Time management
Interrupt, signal process
Memory management, I/O management
Networking, File system</td></tr>
<tr><td>Logic bugs</td><td>if statement</td></tr>
<tr><td>Duraes, J., and Madeira, H.[21]
H.J. Lee[22]</td><td colspan="2">Assignment
Checking
Interface
Algorithm</td><td>H.J. Lee et al. [23]
H.J. Lee and J.W. Park[24]</td><td colspan="2">Data violation
Time out
Complete with delay
Error without effect
Exception(added)</td></tr>
<tr><td>J. Y. Seo[25]</td><td colspan="2">S/W to H/W Interface
S/W to S/W Interface</td><td rowspan="3">Y.N. Choi [26]</td><td>Memory allocation</td><td>Leakage
Zero Alloc
Fail Alloc</td></tr>
<tr><td>D.H. Jung et al. [27]</td><td colspan="2">Types
Declarations and definitions
Arithmetic type conversion
Pointer type conversion
Expressions
Control statement expressions
Control flow
Switch statements
Functions</td><td>Memory free</td><td>Illegal Free
Null Pointer Free
Duplicate Free</td></tr>
<tr><td></td><td></td><td>Memory access</td><td>Null Pointer Access
Free Pointer Access
Invalid Pointer Access
Outbound Access
Collision</td></tr>
<tr><td>J. Y. Seo and B. j. Choi[28]</td><td colspan="2">Memory, I/O device, Timer
Task management
Inter-task communication
Exception handling
Vitual memory management
Phsical memory management
Timer management
Interrupt process
I/O management
Networking
File system</td><td>A.Y. Sung, et al.[29]</td><td colspan="2">Memory, I/O device, Timer
Hardware initialization
Task management
Inter-task management
Time management
Interrupt, signal and exception handling
Memory management
I/O management
Networking
File system</td></tr>
<tr><td rowspan="3">S. Y. Lee et al.[30]</td><td>Input date logic</td><td>AD sampling
AD converting
Fail-safe
Interrupt</td><td rowspan="2">A.Y. Sung[31]</td><td colspan="2">Hardware interface</td></tr>
<tr><td>Output date logic</td><td>Output port
Stop output
Fail-safe</td><td>Kernel interface</td><td>Task management
Inter-task communication
Time menagement
Interrupt/exception handling
Memory management
I/O management
Networking, File system</td></tr>
<tr><td>Control logic</td><td>Calculation
Data processing
Branch control
Loop control</td><td>S.H. Lee[32]</td><td>Exception handling</td><td>Device connection
Device disconnection
Device no date
Device illegal data</td></tr>
</table>

각 연구자들이 연구한 표 2와 표 4의 결함의 결함들은 기관 및 연구자들의 관점에 따라 동일한 의미를 다르게 표현하거나 중복된 용어가 존재하므로 용어 표준화가 필요하다. 표준화는 선행연구에서 분류된 결함을 기반으로 임베디드 소프트웨어 전문가와 논의하여 대표 단어로 재정의하여 분류하였고, 표 5와 같이 표준화된 결함요소들이 도출되었다.

표 5와 같이 표준화된 임베디드 소프트웨어 결함에 대하여 표 3의 내용분석 절차를 이용하여 연구자와 전문가가 각자 결함을 분류하였고, 그림 2의 신뢰도 테이블(Reliability Table)를 이용하여 서로 일치하는 분류를 도출하였다.

표 5. 표준화된 임베디드 소프트웨어 결함
Table 5. Standardized embedded software defects

Code	Defects	Code	Defects
1-1	Logic error	5-1	Optimization
1-2	Infinite loops	5-2	Time out
1-3	If & case statements	5-3	Time fault causes data loss
1-4	Error checking	5-4	Complete with delay
1-5	Check variables	5-5	Time delay
1-6	Serialization	5-6	Feedback error
1-7	Deadlock	5-7	Set time & read
1-8	Concurrent processing	6-1	Wrong H/W interface
1-9	Task management	6-2	I/O device
1-10	Recursion	6-3	User interface
2-1	Wrong function	6-4	External I/F
2-2	Non re-enterant function	6-5	send & receive packet error
2-3	Incorrect objects	6-6	Networking
2-4	Incorrect relationship	6-7	Input value error
2-5	Incorrect return	6-8	Output signal
3-1	Missing computation	6-9	Data I/O process
3-2	Incorrect operand & operator	6-10	Sensor data incorrect
3-3	Incorrect parenthesis	6-11	Jitter
3-4	round & truncate	7-1	Date access
3-5	Sign convention	7-2	Shared memory
3-6	Divide by zero	7-3	Data violation
3-7	Arithmetic over- flow & underflow	7-4	Data boundary error
4-1	Wrong interrupt	7-5	Type mismatch
4-2	Wrong subroutine called	7-6	Storage data save
4-3	Nonexistent subroutine call	7-7	Flash memory
4-4	Wrong parameter	7-8	Memory init
4-5	Inter-task communication	7-9	Memory management
4-6	Internal I/F	7-10	Memory access
4-7	Module I/F	7-11	Resource leaks
4-8	Incorrect API usage	7-12	Memory free
4-9	Incorrect protocol	7-13	Memory overflow
4-10	SW architecture	7-14	Memory violations
4-11	Exception handling	-	-
4-12	None sensor logic		

표 6. 최종적으로 분류된 임베디드 소프트웨어 결함
Table 6. Derived final embedded software defects

Defects	definition	Detailed defects
Incorrect Logic (D1)	Control logic & calculaton	Control flow, if, case, loop statements, divided by zero
Incorrect function (D2)	Function self defects	Non re-enterant function, Incorrect Objects.
Task management (D3)	Concurrent processing error	Deadlock, Race condition, Task management
Exception handling (D4)	Device driver exception handle error	S/W exception handling excluding device driver error
Internal S/W interface (D5)	Communicatierror on between software	Internal I/F, Inconsistent module I/F Wrong parameter
External Interface (D6)	Communication error with external system	Networking, Send & receive packet error Human Interface
Device driver (D7)	Hardware control device driver	I/O device, I/O port process I/O device status
Hardware interrupt (D8)	Processing routine for hardware interrupt	Non interrupt routine, Incorrect interrupt routine, process error
Timing error (D9)	Defects that can't complete in time	Time out, Time delay, feedback error, Set time & read time
Data, shared memory (D10)	Data & static Memory	Data definition, Data access, Shared memory
Dynamic memory (D11)	Defect using dynamic memory	Memory init, Memory management, Resource leaks, Memory overflow
Flash memory & File system (D12)	Data storage device	Flash memory, Storage data save

이런 과정을 여러 번 수행한 결과, 표 6과 같이 최종적으로 합의된 열두 가지의 임베디드 소프트웨어 결함이 도출되었다. 합의된 결함 분류의 신뢰도를 확인하기 위하여 분류 지수(Categorization Index)를 이용하였다.

분류지수는 그림 2의 대각선에 위치된 합의된 수치에 대한 비율을 의미하며, 결함 요소 총 66 건 중 합의된 건수가 64건으로 분류 지수는 약 96% 정도로서 우수한 수준으로 분류되었다.

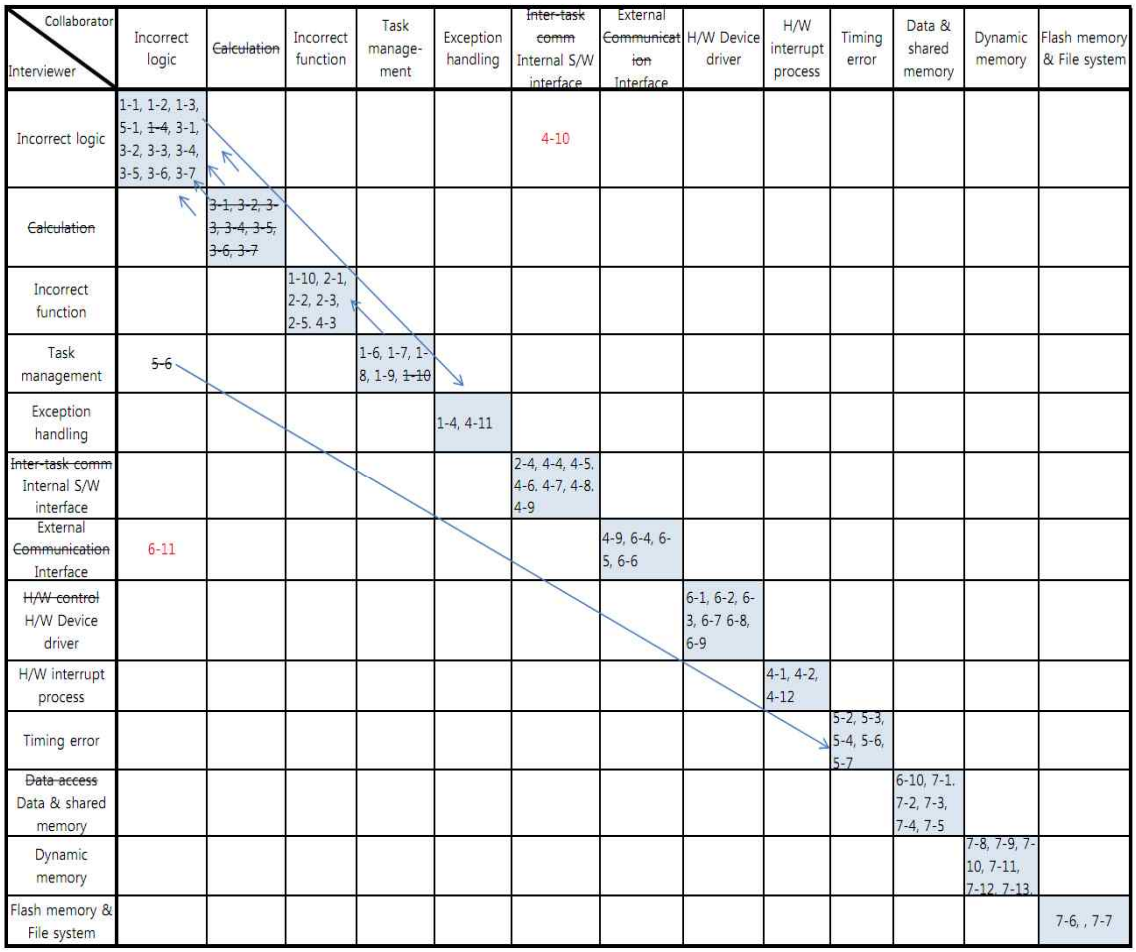

그림 2. 표 5의 결함에 대한 내용분석결과
Fig. 2. The result for content analysis of table 5

3.2.2 전문가 설문 수행 및 분석

표 6과 같이 도출된 열두 가지의 결함요소 간의 인과관계를 데마텔기법으로 도출하기 위하여, 임베디드 개발 전문가의 의견을 설문으로 수집하였다. 설문 응답자는 표 7과 같이 기술사를 중심으로 감리사와 특급기술자 14인을 대상으로 구성하였다. 설문은 열두 가지의 결함이 다른 결함에게 영향을 주는 정도를 0부터 4까지의 영향력(0.영향력없음, 1.영향력낮음 2.보통영향력, 3.높은영향력, 4.매우높은영향)을 선택하는 방식으로 진행하였다.

설문 데이터의 신뢰도를 진단하기 위하여 SPSS 도구를 이용하여 크론바흐 알파계수(Cronbach's α)를 측정하였다. 크론바흐 알파계수는 설문 값의 신뢰도를 측정하는 지표로서, 값의 범위는 0에서 1사이로 도출되며, 1 값에 가까울수록 신뢰도가 높다. 크론바흐 알파계수를 진단한 결과, 0.895라는 높은 값이 도출되었고, 응답자별로는 표 8과 같은 값으로 데마텔 설문 데이터의 신뢰도를 확인할 수 있었다.

표 7. 임베디드 전문가 응답자 요약
Table 7. Summary of embedded expert respondents

Class	Contents
Embedded average career	9.7 year
Gender	Male(78.5%), Female(21.4%)
Career	Professional Engineers(64.3%), Auditor(21.4%) Top Engineer(14.3%)
Embedded Experiences	100%

표 8. 크론바흐 알파 신뢰도 분석결과
Table 8. The analysis result of cronbach alpah

Respondents	If the item was deleted, the scale average	When the item is deleted, the scale is distributed	Modified – full correlation	If this item is deleted, the cronbach alpha coefficient
R1	28.24	86.982	.748	.880
R2	26.70	90.923	.779	.882
R3	27.58	94.175	.328	.899
R4	27.08	92.239	.471	.892
R5	26.99	89.951	.659	.885
R6	25.56	85.060	.589	.889
R7	27.50	93.748	.594	.888
R8	27.04	90.656	.564	.888
R9	26.87	89.150	.585	.887
R10	26.42	89.190	.695	.883
R11	27.64	93.659	.420	.894
R12	27.65	91.265	.454	.893
R13	25.65	84.804	.807	.877
R14	26.18	85.967	.625	.888

임베디드 소프트웨어 전문가의 결함 간의 영향관계 의견이 수집되었으므로, 설문을 기반으로 임베디드 소프트웨어 결함의 인과관계 도출을 위하여, 데마텔기법을 여섯 단계로 적용하였다. 첫 번째, 수집된 인과관계 행렬 값에 대하여 식 (1)을 대입하여 산술평균을 내었고, 표 9와 같이 일반화된 행렬(A)을 도출하였다.

두 번째, 일반화(A)된 행렬에 대한 정규화를 위하여 식 (2)을 이용하여 행의 합의 최대값, 열의 합의 최대값 중에서 최소값을 도출하여 식 (3)를 적용하였고, 표 10과 같은 정규화된 행렬(N)을 도출하였다.

세 번째, 정규화된 행렬(N)를 단위행렬의 역행렬을 구하는 식 (4)를 이용하여 적용하였고, 표 11과 같은 종합인과관계 행렬(T)를 구하였다. 네 번째, 표 11의 종합인과관계행렬(T)에서 행의 합(D)과 열의 합(R)을 구하고 식 (5)와 식 (6)을 이용하여 중심도(D+R)요인과 원인도(D-R)요인을 구한 결과, 표 12와 같은 결과가 도출되었다. 다섯 번째, 영향력을 판단하는 기준점은 식 (7)을 이용하여 계산하였고, 0.375라는 평균값이 도출되었다. 행렬 T에서 기준점보다 작은 값은 영향을 미치지 않는 것이고, 기준점보다 큰 값은 영향을 미치는 것이다. 마지막으로. 도출된 표 12를 이용하여 인과관계도를 도식화한다. 표 12의 원인도(D-R) 요인은 y축으로, 중심도(D+R)요인은 x축으로 기준점을 설정한 후, 표 12 값 중에서 기준점 0.375보다 큰 값을 이용하여 인과관계를 도식화하였고, 그림 3과 같은 인과관계도가 도출되었다. 도식화하는 과정 중에 결함들이 중심도(D+R)요인 8.75부터 9.25까지 집중되어 있어서, 용이한 식별을 위하여 x축을 점선 박스로 확대하여 도식화하였다.

표 9. 일반화된 인과관계행렬(A)
Table 9. Generalized cause & effect matrix(A)

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
D1	0.00	2.36	2.57	2.36	1.64	1.57	2.43	2.21	2.71	2.57	2.71	2.64
D2	2.29	0.00	2.64	2.29	2.21	2.07	2.14	2.07	2.14	2.21	2.29	1.57
D3	1.79	1.64	0.00	2.00	2.36	2.29	2.14	2.43	2.57	2.21	2.21	1.64
D4	2.29	1.79	2.21	0.00	1.64	1.86	1.64	1.71	2.00	1.71	1.64	1.36
D5	1.86	2.00	2.29	1.79	0.00	2.14	2.07	1.86	2.14	1.43	1.64	1.57
D6	1.57	1.29	1.43	1.57	1.71	0.00	1.79	2.07	2.64	1.21	1.14	0.86
D7	1.43	1.43	1.64	1.50	1.71	2.14	0.00	2.21	2.36	1.64	1.71	1.21
D8	1.36	1.14	1.79	1.29	1.64	2.07	2.64	0.00	2.36	1.57	1.57	1.36
D9	1.57	1.64	2.29	1.64	1.93	2.64	2.36	2.36	0.00	1.64	1.36	1.36
D10	2.21	2.29	2.21	1.86	1.86	1.71	1.93	1.86	1.86	0.00	2.36	1.50
D11	2.21	2.29	2.21	2.07	1.93	1.64	1.86	1.79	1.93	2.29	0.00	1.64
D12	1.79	1.71	2.21	1.43	1.64	1.71	2.00	2.21	1.86	2.14	1.93	0.00

표 10. 정규화된 인과관계행렬(N)
Table 10. Normalized cause & effect matrix(N)

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
D1	0.000	0.091	0.100	0.091	0.064	0.061	0.094	0.086	0.105	0.100	0.105	0.102
D2	0.089	0.000	0.102	0.089	0.086	0.080	0.083	0.080	0.083	0.086	0.089	0.061
D3	0.069	0.064	0.000	0.078	0.091	0.089	0.083	0.094	0.100	0.086	0.086	0.064
D4	0.089	0.069	0.086	0.000	0.064	0.072	0.064	0.066	0.078	0.066	0.064	0.053
D5	0.072	0.078	0.089	0.069	0.000	0.083	0.080	0.072	0.083	0.055	0.064	0.061
D6	0.061	0.050	0.055	0.061	0.066	0.000	0.069	0.080	0.102	0.047	0.044	0.033
D7	0.055	0.055	0.064	0.058	0.066	0.083	0.000	0.086	0.091	0.064	0.066	0.047
D8	0.053	0.044	0.069	0.050	0.064	0.080	0.102	0.000	0.091	0.061	0.061	0.053
D9	0.061	0.064	0.089	0.064	0.075	0.102	0.091	0.091	0.000	0.064	0.053	0.053
D10	0.086	0.089	0.086	0.072	0.072	0.066	0.075	0.072	0.072	0.000	0.091	0.058
D11	0.086	0.089	0.086	0.080	0.075	0.064	0.072	0.069	0.075	0.089	0.000	0.064
D12	0.069	0.066	0.086	0.055	0.064	0.066	0.078	0.086	0.072	0.083	0.075	0.000

표 11. 종합 인과관계행렬(T)
Table 11. Total cause & effect matrix(T)

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
D1	0.362	0.433	0.506	0.438	0.425	0.450	0.496	0.485	0.531	0.459	0.462	0.395
D2	0.419	0.325	0.479	0.412	0.419	0.440	0.459	0.453	0.484	0.421	0.423	0.338
D3	0.390	0.373	0.372	0.390	0.412	0.436	0.446	0.452	0.484	0.408	0.407	0.330
D4	0.366	0.339	0.405	0.279	0.346	0.376	0.382	0.382	0.415	0.351	0.348	0.287
D5	0.361	0.355	0.418	0.353	0.297	0.397	0.408	0.398	0.432	0.351	0.357	0.302
D6	0.302	0.284	0.334	0.298	0.310	0.269	0.344	0.351	0.391	0.294	0.290	0.237
D7	0.320	0.311	0.367	0.317	0.333	0.369	0.305	0.381	0.408	0.331	0.332	0.268
D8	0.314	0.297	0.367	0.306	0.327	0.363	0.394	0.298	0.404	0.325	0.324	0.269
D9	0.348	0.339	0.414	0.344	0.363	0.411	0.415	0.412	0.353	0.354	0.343	0.291
D10	0.388	0.380	0.433	0.370	0.378	0.397	0.419	0.413	0.439	0.314	0.397	0.312
D11	0.391	0.383	0.436	0.380	0.383	0.398	0.420	0.414	0.445	0.398	0.316	0.320
D12	0.357	0.345	0.414	0.339	0.355	0.381	0.404	0.408	0.421	0.374	0.366	0.244

표 12. 임베디드 소프트웨어 결함의 인과관계 결과
Table 12. Cause & effect analysis result of defects

code	Defects	D	R	D+R	D-R
D1	Incorrect Logic	5.44	4.32	9.76	1.13
D2	Incorrect function	5.07	4.16	9.24	0.91
D3	Task management	4.90	4.94	9.84	-0.05
D4	Exception handling	4.28	4.22	8.50	0.05
D5	Internal S/W interface	4.43	4.35	8.78	0.08
D6	External Interface	3.70	4.69	8.39	-0.98
D7	Device driver	4.04	4.89	8.93	-0.85
D8	H/W interrupt	3.99	4.85	8.83	-0.86
D9	Timing error	4.39	5.21	9.59	-0.82
D10	Data & shared memory	4.64	4.38	9.02	0.26
D11	Dynamic memory	4.68	4.36	9.05	0.32
D12	Flash memory & File system	4.41	3.59	8.00	0.81

Ⅳ. 결함 인과관계 분석

4.1 임베디드 소프트웨어 결함 인과관계 분석

도식화된 그림 3을 분석하면 원인도(D-R)요인이 양수 값이거나 순수하게 원인 결함이라면, 다른 결함에게 영향을 미치는 그룹으로 분류할 수 있다. 원인도(D-R)요인 값 중에 제일 큰 값의 결함은 다른 결함에게 가장 많은 영향을 미친다[33]. 본 논문에서는 플래쉬 메모리 및 저장장치(Flash Memory & File System) 결함, 예외처리(Exception Handling, D4), 내부 소프트웨어 인터페이스(Internal Software Interface, D5), 동적메모리 결함(Dynamic Memory, D11), 데이터 처리 및 공유메모리(Data & Shared

Memory, D10), 함수결함(Incorrect Function, D2), 논리결함(Incorrect Logic, D1)들이 영향을 주는 원인 그룹으로 분석되었고, 그 중에 논리 결함, 함수결함, 플래쉬 메모리 및 저장장치, 예외처리 결함, 동적메모리 결함 순으로 영향 원인이 큰 것으로 나타났다.

원인도(R-C)요인 값이 음수 값이거나 다른 결함에게 영향을 받기만 한다면, 영향을 받는 효과(Effect)그룹이라고 한다. 본 연구에서는 외부 인터페이스(External Interface, D6), 하드웨어 인터럽트(Hardware Interrupt, D8), 디바이스 드라이버(Device Driver, D7), 타이밍 결함(Timing Error, D9), 태스크 관리(Task Management, D3)결함이 효과그룹으로 분석되었고, 그중에 외부 인터페이스, 하드웨어 인터럽트, 디바이스 드라이버, 타이밍 결함, 태스크 관리 결함 순으로 영향을 많이 받는 것으로 분석되었다.

결함 간의 인과관계가 밝혀졌기 때문에, 핵심결함을 분석하면, 핵심 결함은 다른 결함으로부터 영향을 많이 받는 효과그룹에 많이 존재한다.

첫 번째로 제일 중요한 핵심 결함은 외부 인터페이스 결함으로 도출되었다. 외부 인터페이스는 네트워크, 시리얼 포트, 휴먼 인터페이스 등이 있다. 만약 휴먼 인터페이스에 결함이 있다면 사용자가 원하는 기능을 요청해도 다른 기능이 동작하여 문제가 발생할 수 있고, 외부 시스템에서 송신하는 명령을 잘못 수신하여, 잘못 동작하면 중대한 문제를 발생시킬 수 있으므로 제일 중요한 결함이라고 이해할 수 있다.

두 번째로 중요한 결함은 하드웨어 인터럽트로, 타이머, 0 으로 나누기, 오버플로우, 언더플로우 등의 하드웨어 인터럽트가 존재하며, 개발자가 인터럽트를 처리 루틴을 직접 작성해야 한다. 만약 인터럽트 처리에 결함이 발생하면 중대한 문제가 발생될 수 있으므로 중요한 결함이라고 이해할 수 있다.

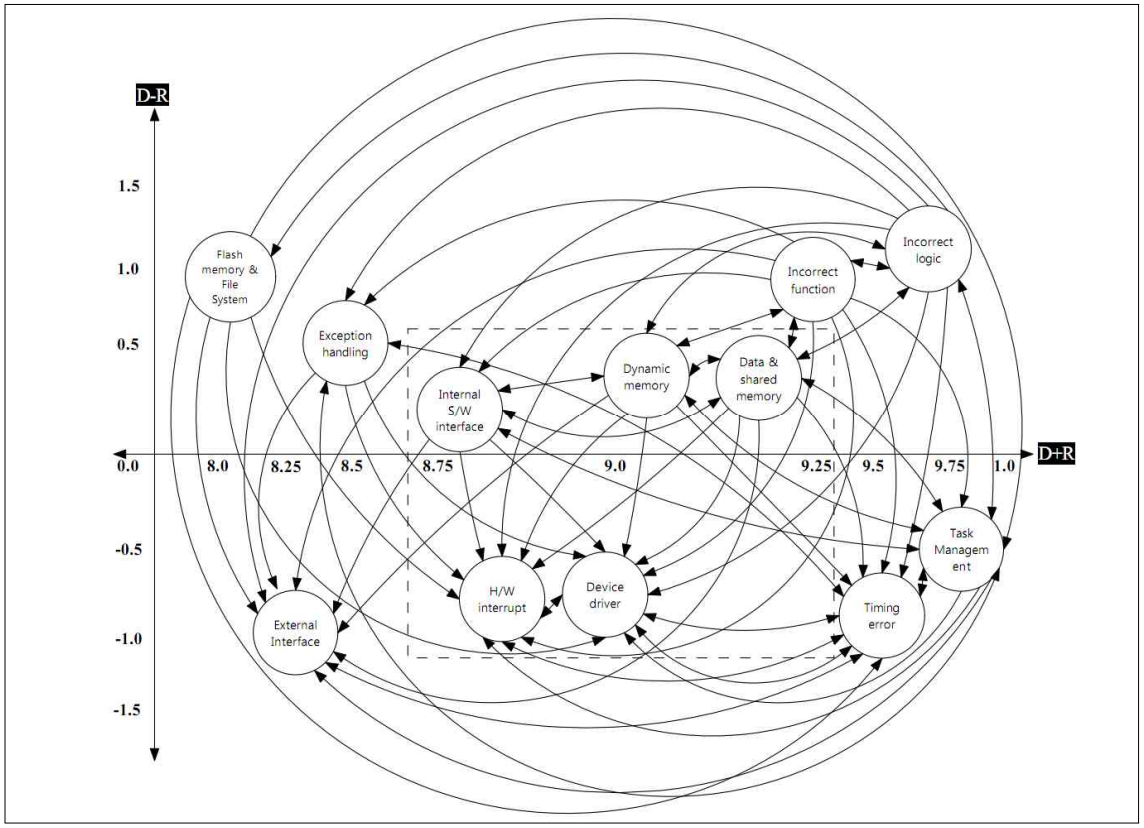

그림 3. 임베디드 소프트웨어 결함 간의 인과관계도
Fig. 3. The casual and effect diagram of embedded software defects

세 번째로 중요한 결함은 디바이스 드라이버 결함이 도출되었다. 디바이스 드라이버는 응용 소프트웨어에서 하드웨어를 제어할 수 있도록 인터페이스를 제공하는 역할을 수행한다. 만약 디바이스 드라이버에 결함이 발생하면 임베디드 시스템이 어떻게 동작할지 예측할 수 없기 때문에 중요한 결함으로 도출된 것으로 이해할 수 있다.

네 번째로 중요한 결함은 시간 결함(Timing error)으로 도출되었다. 임베디드 시스템은 자동차 자율주행시스템, 항공분야의 항법시스템, 원자력 발전소 제어장치, 국방 분야의 미사일 제어 장치 등 인간의 생명과 직결될 수 있다. 만약 즉각적으로 반응해야 하는 기능에 대하여 시간이 초과된다면, 예측할 수 없는 결과를 초래될 수 있을 것이다. 그런 면에서 중요한 결함으로 도출되었다고 이해할 수 있다.

다섯 번째로 중요한 결함은 태스크 관리 결함으로 도출되었다. 태스크 관리 결함은 데드락, 경쟁조건 등으로 인하여 태스크들이 정상적으로 수행되지 않을 수 있으므로 중요한 결함요소로 도출되었다. 이와 같이 다섯 가지 결함이 가장 중요한 결함으로 도출되었고, 나머지 결함은 핵심 결함만큼 중요도가 미치지 못하는 것으로 파악되었다.

핵심 결함을 종합적으로 해석하면, 임베디드 시스템은 외부 시스템 및 환경에 영향을 받지 않고 견실하게 수행되어야 하며, 인터럽트는 정확하게 처리되어야 하며, 다양한 하드웨어 종류에 적합하게 동작해야 하고, 제 시간 내에 응답하도록 구현되어야 하며, 다양한 여러 태스크들이 착종되지 않고 자기 기능을 충실하게 수행할 수 있도록 제작하는 것이 가장 중요하다고 나타났다.

연구 결과의 신뢰성을 확인하기 위하여 임베디드 전문가 6인에게 임베디드 소프트웨어 결함 중에서 중요하다고 판단되는 결함에 대한 의견을 수집하였다. 표 13과 같이 전문가 중에는 논리결함이나, 예외처리, 데이터, 동적 메모리 결함도 중요하다고 의견을 준 경우도 있었으나, 중요 결함에 대한 전문가들의 공통된 의견은 하드웨어 인터럽트, 디바이스 드라이버, 외부 인터페이스, 타이밍, 태스크 관리 결함이 중요하다고 의견을 주었다. 이와 같이 전문가들이 중요하다고 생각하는 결함과 본 연구에서 도출한 중요 결함을 비교한 결과, 미미한 차이가 있을 뿐 전부 일치하는 것으로 확인되었다.

표 13. 임베디드 S/W 전문가의 중요결함에 대한 의견
Table 13. Opinion of key defects of embedded experts

Defects	Experts						Rank of defects
	1	2	3	4	5	6	
Incorrect Logic		○				○	7
Incorrect function							12
Task management	○		○	○		○	5
Exception handling			○	○	○		6
Internal S/W interface			○				11
External Interface		○	○	○	○	○	3
Device driver	○	○	○		○	○	2
H/W interrupt	○	○	○	○	○	○	1
Timing error	○	○		○	○	○	4
Data & shared memory			○	○			9
Dynamic memory	○						10
Flash memory & File system	○					○	8

V. 결론 및 향후 과제

5.1 연구의 제안

임베디드 시스템은 순수 소프트웨어와 유사한 논리 로직이 있는 반면에, 일반 소프트웨어에서는 찾아보기 힘든 하드웨어를 제어하는 소프트웨어가 있다. 임베디드 시스템은 자동차 자율주행, 항공기 항법시스템, 핵 발전소 제어장치, 미사일 제어 장치, 우주 분야에 사용되고 있으며 인간의 생명과 직결될 수 있다. 그러므로 임베디드 시스템에 결함이 수행되면 치명적인 결과가 야기될 수 있다.

소프트웨어에 내재된 결함은 전부 제거하기 어려우므로, 일반 소프트웨어 영역에서는 핵심 결함을 집중적으로 제거하여 결함을 최소화시키는 것이 효율적으로 소프트웨어를 개발할 수 있다고 한다. 하지만, 임베디드 소프트웨어 결함에 대하여 조사한 결과, 체계적이고 통합적인 관점으로 임베디드 결함을 도출한 연구는 없고, 연구자가 관심이 있는 분야에 대해서만 편향적으로 조사한 연구가 주를 이루

고 있었다. 이와 같이 임베디드 소프트웨어 결함에는 어떤 결함들이 존재하고, 결함들 간에는 어떻게 영향을 미치며, 어떤 결함이 핵심 결함인지에 대한 연구는 미흡한 것으로 파악되었다. 이에 본 논문에서는 소프트웨어 결함과 임베디드 소프트웨어 결함을 수집하여 내용분석기법과 데마텔기법을 이용하여 임베디드 소프트웨어 결함간의 인과관계를 도출하고 중요 결함을 도출하였다.

5.2 연구의 의의

본 논문에서는 소프트웨어 결함과 임베디드 소프트웨어 결함을 수집하여, 내용분석을 분석한 결과, 논리결함(Incorrect Logic), 함수결함(Incorrect Function), 태스크 관리결함(Task Management), 예외처리결함(Exception Handling), 내부 소프트웨어 인터페이스(Internal S/W interfacr), 외부 인터페이스(External Interface), 디바이스 드라이버 결함(Device Driver), 하드웨어 인터럽트(Hardware Interrupt), 타이밍 결함(Timing Error), 데이터 처리 및 공유메모리(Data & Shared Memory), 동적 메모리 사용 결함(Dynamic Memory) 및 플래쉬 메모리/파일 시스템(Flash Memory & File System) 결함 등 열두 가지 결함이 도출되었다.

도출된 열두 가지 결함에 대하여 데마텔기법을 이용하여 인과관계를 도출하였고, 그림 3와 같은 임베디드 소프트웨어 결함 인과관계도를 도출할 수 있었다. 도출된 인과관계도를 분석한 결과, 외부 인터페이스 결함, 하드웨어 인터럽트 결함, 디바이스 드라이버 결함, 타이밍 결함, 태스크 관리 결함이 중요한 것으로 분석되었다.

본 연구의 학문적인 의의로는, 첫째, 임베디드 소프트웨어 품질을 향상시키기 위하여 결함 자체로 접근하여, 순수 소프트웨어 결함과 임베디드 결함을 수집하여 정리하였다는 데 의의가 있을 수 있다. 둘째, 정리된 결함을 이용하여 결함 간의 영향력을 파악할 수 있는 인과관계도를 도출하였다는 데 의의가 있을 수 있다. 셋째, 임베디드 결함 간의 인과관계를 이용하여 결함의 중요도 가중치를 산정할 수 있는 방법을 제시했다는 점이다. 향후에 임베디드 소프트웨어 결함의 중요도 가중치 산정을 위한 기준이 필요할 때, 본 연구를 활용할 수 있을 것으로 기대한다.

산업적인 활용 방안으로는 첫째, 임베디드 시스템을 제작할 때, 막연한 품질 목표보다는 결함 중요도를 이용하여 효율적, 효과적인 결함관리를 수행할 수 있을 것이다. 둘째, 임베디드 소프트웨어 개발과 테스트 시에 효율적으로 결함을 제거하기 위하여, 핵심 결함에 역량을 집중할 수 있는 근거를 마련했다는 점이다. 셋째, 결함의 우선순위가 파악되었으므로 적합한 임베디드 소프트웨어 테스트 기법을 선정하는 데 도움이 될 것으로 기대한다. 넷째, 임베디드 소프트웨어 결함 주입 테스트 시에 핵심 결함에 집중적으로 주입하는 테스트를 수행할 수 있을 것으로 기대한다.

5.3 연구의 한계점

임베디드 소프트웨어 결함을 열두 가지로 분류하고 결함 간의 인과관계를 도출하였지만, 결함을 관점 별로 분류하지 못했다는 점이다. 만약 결함을 관점 별로 분류할 수 있다면 좀 더 유용하게 활용할 수 있을 것이다. 또한, 요소간의 네트워크 적인 영향관계를 이용하는 ANP 기법을 적용하여 좀 더 정확한 결함의 가중치를 도출하지 못한 점이라고 할 수 있다.

5.4 향후 연구 방향

향후 연구로는 열두 가지로 도출된 임베디드 소프트웨어 결함에 대하여 AMOS 구조방정식이나 부분최소자승법(PLS, Partial Least Squre)기법을 통하여 결함을 관점별로 그루핑하는 연구가 있으며, 그루핑이 완료되면 결함 간의 인과관계를 이용하는 기법을 적용하여 좀 더 정확한 임베디드 소프트웨어 결함에 대한 가중치를 산정하는 연구가 남아 있다.

References

[1] S. M. Huh and W. J. Kim, "A Method to

Establish Severity Weight of Defect Factors for Application Software using ANP", Journal of KIISE, Vol. 42, No. 11, pp. 1449-1460, Nov. 2015.

[2] IEEE, "IEEE Standard Classification for Software Anomalies", IEEE Std. 1044-1993, 1994.

[3] IBM, "Orthogonal Defect classification v5.2 for Software Design and Code", IBM, 2013.

[4] Huber JT, "A comparison of IBM's orthogonal defect classification to Hewlett Packard's defect origins, types and modes", In: Proceedings of International Conference on Applications of Software Measurement. San Jose, CA, pp. 1-17, 2000.

[5] Wikipidia, http://en.wikipedia.org/wiki/Software_bug, [Accessed: May. 02. 2018]

[6] J. D. Hagar, "Software Test Attacks to Break Mobile and Embedded Devices", CRC press, 2013.

[7] M. Barr, "Five top causes of nasty embedded software bugs", Embedded systems design, Vol. 23 No. 3, pp. 10-15, Apr. 2010.

[8] M. Barr, "Five more top causes of nasty embedded software bugs", Embedded systems design Vol. 23, No. 9, pp. 9-12, Apr. 2010.

[9] H. M. Choi, A. y. Sung, B. J. Choi, and J. W. Kim, "A Functionality - based Evaluation Model for Embedded Software", Journal of KIISE: Software and Applications, Vol. 32, No. 12, pp. 1192-1204, Nov. 2005.

[10] D. Jankowicz, "The Easy Guide to Repertory Grids", John wiley & sons, 2005.

[11] D. Sumrit and P. Anuntavoranich, "Using DEMATEL method to analyze the causal relations on technological innovation capability evaluation factors in Thai technology-based firms", International Transaction Journal of Engineering, Management, & Applied Sciences & Technologies, Vol. 4, No. 2, pp. 81-103, Jan. 2013.

[12] M. Leszak and D. E. Perry, "A Case Study in Root Cause Defect Analysis", Software Engineering, 2000. Proceedings of the 2000 International Conference, pp. 428-437. June 2000.

[13] R. R. Lutz, "Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems", In Requirements Engineering,, Proceedings of IEEE International Symposium on, pp. 126-133, Jan. 1993.

[14] T. L. Bennett and P. W. Wennberg, "Eliminating embedded software defects prior to integration test", Journal of Defence Software Engineering, Dec. 2005.

[15] Nigel Jones, "A Taxonomy of Bug Types in Embedded Systems", Oct. 2009.

[16] P. Rodriguez-Dapena, "Software Safety Verification in Critical Software Intensive Systems", Draft Phd thesis. Eindhoven Technical University, Jan. 2002.

[17] Tammy Noergaard, "Embedded System Architecture", Elsevier, 2005.

[18] S. Ji and X. Bao, "Research on Software Hazard Classification", Procedia Engineering, Vol. 80, pp. 407-414, 2014.

[19] H. W. Jung, "Failure Mode based Test Methods for Embedded Software", Ajou University Graduate School of Engineering, Jun. 2007.

[20] A. Sung, W. Srisa-an, G. Rothermel, and T. Yu, "Testing Inter-Layer and Inter-Task Interactions in RTES Applications", In Software Engineering Conference (APSEC), 17th Asia Pacific, pp. 260-269, Jun. 2010.

[21] J. Duraes and H. Madeira, "Emulation of Software Faults by Educated Mutations at Machine-Code Level", Proceedings of the Thirteenth IEEE InternationalSymposium on Software Reliability Engineering, ISSRE'02, pp. 329-340, Nov. 2002.

[22] H. J. Lee, "Statistical JTAG fault injection methodology for reliability verification of aerospace embedded systems", Department of

Electronics and Information Engineering Korea Aerospace University, Feb. 2012.

[23] H. J. Lee, J. H. Yoon, K. Y. Lee, D. W. Lee, and J. W. Na, "Reclassification of fault, error, and failure types for reliability verification of defense embedded systems", Institute of Control, Robotics and Systems, pp. 925-932, July 2012.

[24] H. J. Lee and J. W. Park, "JTAG fault injection methodology for reliability verification of defense embedded systems", Journal of the Korea Academia-Industrial cooperation Society, Vol. 14, No. 10, pp. 5123-5129, Aug. 2013.

[25] J. Y. Seo, "Embedded Software Interface Test Based on the Status of System", Department of Computer Science and Engineering The Graduate School of EWHA Womans University, Jan. 2009.

[26] Y. N. Choi, "Automated Debugging Cooperative Method for Dynamic Memory Defects in Embedded Software System Test", Department of Computer Science and Engineering The Graduate School of EWHA Womans University, Jan. 2010.

[27] D. H. Jung, S. J. Ahn, and J. Y. Choi, "Programming Enhancements for Embedded Software Development-focus on MISRA-C", Journal of KIISE : cmputing practives and letters, March, Vol. 19, No. 3, pp. 149-152, Jan. 2013.

[28] J. Y. Seo and B. J. Choi, "An Interface Test Tool Based on an Emulator for Improving Embedded Software Testing", Journal of KIISE : Computing Practices and Letters, Vol. 14, No. 6, pp. 547-558, Aug. 2008.

[29] A. Y. Sung, B. J. Choi, and S. K. Shin, "An interface test model for hardware-dependent software and embedded OS API of the embedded system", Computer Standards & Interfaces, Vol. 29, No. 4, pp. 430-443, Sep. 2006.

[30] S. Y. Lee, J. S. Jang, K. H. Choi, S. K. Park, K. H. Jung, and M. H. Lee, "A Study of Verification for Embedded Software", Industrial Engineering & Management Systems, pp. 669-676, 2004.

[31] A. Y. Sung, "Interface Based Embedded Software Test for Real-Time Operating System", Department of Computer Science and Engineering EWHA Womans University, July 2007.

[32] S. H. Lee, "Automated Method for Reliability Verification in Embedded Software System Exception Handling Test", Department of Computer Science and Engineering The Graduate School of Ewha Womans University, Jan. 2010

[33] M. G. ÖLÇER, "Developing a spreadsheet based decision support system using DEMATEL and ANP approaches(Doctoral dissertation, DEÜ Fen Bilimleri Enstitüsü), Jun. 2013.

저자소개

허 상 무 (Sang-Moo Huh)

2017년 8월 : 서울과학기술대학교 산업정보시스템과(공학바사수료)
2018년 5월 ~ 현재 : (주)한국전산감리원 수석
관심분야 : 소프트웨어 공학, 소프트웨어 품질, 소프트웨어 결함, IT서비스, 데이터베이스

김 우 제 (Woo-Je Kim)

1986년 2월 : 서울대학교 산업공학과 공학사
1988년 2월 : 서울대학교 산업공학과 공학석사
1994년 2월 : 서울대학교 산업공학과 공학박사
2003년 ~ 현재 : 서울과학기술대학교 글로벌융합산업공학과 교수
1988년 4월 ~ 1991년 2월 : 동양 경제연구소 연구원
1999년 ~ 2001년 : University of Michigan Visting scholar
관심분야 : 소프트웨어공학, IT서비스, 최적화 등