임베디드 소프트웨어 테스팅 전략 및 실제

최병주 (이화여자대학교), 서주영 (아주대학교)

Ⅰ. 서 론

최근 자동차, 스마트폰, 의료기기와 같은 다양한 임베디드 산업 분야의 소프트웨어 의존도와 비중이 커지면서 소프트웨어 품질이 경쟁력의 핵심요소로 부상하고 있다. 특히 도요타 자동차의 리콜 사태와 같이 임베디드 소프트웨어의 품질이 사회적 문제로 가시화되기 시작하자 산업 현장에서의 임베디드 소프트웨어 테스트에 대한 관심은 그 어느 때보다 높다.

현재 국내·외적으로 소프트웨어 품질에 대한 열망은 소프트웨어 테스트 기술과 자동화 도구들에 대한 많은 발전을 이루어 냈다. 그러나 임베디드 산업 분야의 소프트웨어 테스트는 여전히 경험에 의존하는 노동집약적 테스트가 중심이 되고 있으며, 일반 소프트웨어 산업 분야처럼 다양한 테스트 기법과 도구를 활용한 기술집약적 테스트는 미약한 실정이다. 이러한 임베디드 소프트웨어 테스트 현황의 저변에는 일반 소프트웨어 테스트 기술을 그대로 임베디드 소프트웨어에 적용하려는 데서 오는 문제가 있다.

사실 일반적인 소프트웨어 테스트 기술을 임베디드 소프트웨어 테스트에 적용하기엔 알아야할 많은 차이들이 존재한다. 그 중 가장 중요한 차이는 일반 소프트웨어에 비해 임베디드 소프트웨어는 자원 제약이 심한 환경에서 운용된다는 사실이다. 또한 임베디드 산업 분야의 소프트웨어와 하드웨어는 타겟 제품 기능에 맞춤되어 설계되기 때문에, 서로에 대한 의존도와 결합력이 높고, 소프트웨어 혹은 하드웨어의 일부분만으론 시스템 기능을 확인하는 것이 매우 제한적인 특징이 있다. 이러한 특징 때문에 임베디드 소프트웨어는 시스템으로 통합된 이후에 실제 운용되면서 야기되는 상호 작용을 기반으로 한 런-타임 동적 테스트의 비중이 높고 실제로 매우 중요하다.

런-타임은 '시스템이 실행하는 동안, 즉 시스템이 시작되어 종료되기까지 동작하는 모든 동적 상황'을 의미한다. 즉 런-타임에서의 활동은 단순히 하나의 소프트웨어에 국한된 실행이 아니라 이와 연관된 컴포넌트들을 로딩하고 통합하는 동적 연동 상황, 멀티쓰레드 오퍼레이션, 공유 자원 관리와 같이 프로세스들 혹은 쓰레드들 사이의 다양한 동적 행위를 포함하며 이들을 중심으로 자원 제약이 심한 운용 환경에서도 검증할 수 있는 테스팅 전략이 필요하다.

본 고에서는 일반 소프트웨어와 구분되는 임베디드 소프트웨어의 특징과 이로 인한 테스트 어려움을 살펴본다. 또한 이를 극복하는 테스트 전략으로 "인터페이스 기반 동적 테스트"를 제안하고 실제 산업 현장에 적용된 사례를 통해 그 의미를 살펴본다.

Ⅱ. 임베디드 소프트웨어 테스트

임베디드 소프트웨어는 일반적으로 '높은 결합력, 잦은 소프트웨어 변경, 시스템 통합 이후 테스트 집중, 다양한 외부 환경 입력'과 같은 개발 특징 및 현황 때문에 테스트에 많은 어려움을 겪는다 [1,2]. 임베디드 소프트웨어 특징을 요약하면 다음과 같다.

- **시스템 자원 최적화를 위한 높은 결합력**
  임베디드 소프트웨어는 제품의 '실시간 성능과 저전력 소모'의 효율적 시스템 자원 관리를 위해 프로그램 코드를 끊임없이 최적화한다. 이러한 고성능에 관한 요구들은 불필요한 코드를 제거하여 메모리 효율과 CPU 성능을 향상시키기 위한 코드 최적화로 이어져 결과적으로

제품을 구성하는 소프트웨어와 하드웨어 컴포넌트들을 매우 긴밀히 결합하게 만든다.

- **하드웨어 플랫폼 의존성에 따른 잦은 소프트웨어 변경**
  임베디드 시스템은 제품이 갖는 목적에 따라 하드웨어와 소프트웨어가 매번 맞춤되어 제작되는 경향이 있다. 임베디드 소프트웨어는 하드웨어에 탑재되어 동작하기 때문에 소프트웨어 측면에선 동일한 기능이더라도, 변경된 하드웨어 플랫폼 설계에 따라 프로그램 코드가 자주 포팅되고 변경된다.

- **시스템 통합 이후 테스트 집중**
  임베디드 시스템의 소프트웨어와 하드웨어는 특수 기능에 맞춤 설계되어 있고 긴밀히 연계되어 있기 때문에, 소프트웨어 혹은 하드웨어의 일부분만으론 시스템 기능을 확인할 수 있는 부분이 매우 제한적이다.
  소프트웨어 컴포넌트 각각을 독립적으로 실행시켰을 때보다 시스템으로 통합한 이후에 활성화되는 기능의 비중이 높기 때문에 시스템 통합 이후에 테스트가 집중되는 경향이 있다.

- **시스템 운용 환경으로부터의 다양한 입력**
  일반 소프트웨어는 주로 사용자의 입력으로 구동하지만, 임베디드 소프트웨어의 경우엔 시스템이 운용되는 외부 환경으로부터의 입력에 의해 구동되는 기능의 비중이 높다. 차량 인포테인먼트 시스템의 내비게이션 소프트웨어의 예를 들면, 주요 기능은 외부의 GPS 나 TPEG 시그널의 입력에 의해 동작한다.

이러한 임베디드 소프트웨어 특징들로 인해 실제 산업 현장에선 임베디드 소프트웨어를 테스트함에 있어 일반적으로 다음과 같은 어려움을 겪고 있다.

- **빅뱅 통합과 단계적 테스트의 어려움**
  일반적으로 소프트웨어 테스트는 테스트 목적에 따라 '단위-통합-시스템'의 단계적 테스트가 이루어져야한다. 그러나 '높은 결합력과 하드웨어 의존성'의 임베디드 소프트웨어 특징은 특정 소프트웨어 컴포넌트만 독립적으로 실행하거나 테스트하기 어렵게 만들기 때문에 '단위-통합' 테스트가 쉽지 않다. 특정 소프트웨어만을 독립적으로 테스트하려면 상호작용하는 다른 소프트웨어 컴포넌트에 대한 스텁 코드나 하드웨어에 대한 시뮬레이션 환경이 필요하게 된다 [2]. 이를 위해선 엄청난 테스트 자원과 비용이 요구되기 때문에 일반적으로 특정 소프트웨어만 테스트하고 싶더라도, 연관되는 다른 소프트웨어와 하드웨어 컴포넌트들이 빅뱅 방식으로 통합된 시스템 상태에서 테스트할 수밖에 없으며 많은 산업 현장에선 현실적으로 단계적인 테스트가 수행되기 힘든 현황이다.

- **다양한 개발 업체의 참여와 인터페이스 결함 발생 가능성 증가**
  제품을 구성하는 임베디드 소프트웨어와 하드웨어 컴포넌트들은 매우 다양한 개발 업체들이 참여하여 구현된다. 일반 소프트웨어의 경우, 상대적으로 매우 안정화된 하드웨어 플랫폼 상에서 동작하고, 함께 운영되는 다른 소프트웨어와도 매우 독립적이다. 그러나 임베디드 소프트웨어의 경우는 제품을 구성하는 소프트웨어와 하드웨어 컴포넌트들이 거의 동시에 개발되고, 점점 짧아지는 제품 사이클 때문에, 여러 업체의 안정화되지 않은 소프트웨어와 하드웨어 컴포넌트들이 동시에 통합되고 테스트된다. 이러한 개발 현황 때문에 일반 소프트웨어와 달리 제품을 이루는 모든 컴포넌트들 사이에 결함이 발생할 가능성이 높다.

- **이질적 계층 구조와 결함 추적의 어려움**
  임베디드 시스템은 '서로 다른 기능을 구현한 컴포넌트, 서로 다른 개발자가 구현한 컴포넌트, 서로 다른 업체에 의해 구현한 컴포넌트' 와 같은 블랙 박스로 서로 행위를 구체적으로 잘 알 수 없는 이질적 컴포넌트 계층들로 구성된다. '하드웨어, HAL (Hardware Abstraction Layer), 시스템 소프트웨어, OS 커널, 미들웨어, GUI (Graphic User Interface), 어플리케이션'이 일반적으로 언급되는 임베디드 시스템의 이질적 계층들이며, 이들 내부는 기능에 따라 다양한 계층으로 세분화된다. 이렇게 내부 구조와 동작을 정확히 알기 힘든 이질적 계층 간의 밀접한 상호작용은 결함 발견 시 결함 위치와 원인을 추적하기 어렵게 만든다.

Ⅲ. 임베디드 소프트웨어 테스트 전략

이질적 계층들이 밀접히 결합되고, 불안정한 빅뱅 통합과 빈번한 상호작용은 '관련되는 모든 영역을 테스트하였는지, 결함 원인이 어떤 컴포넌트에 있는 지, 결함 발생 위치가 정확히 어디인지'를 판단하기 어렵게 만들며, 테스트를 매우 고달픈 작업으로 만든다. 이러한 임베디드 소프트웨어 테스트의 어려움을 해결하기 위해서 통합되는 이질적인 소프트웨어

들과 하드웨어의 상호작용인 인터페이스는 우선적으로 테스트되어야 할 핵심 위치이다 [3].

또한 테스트 기술은 정적 테스트와 동적 테스트로 구분할 수 있는데 비실행기반의 정적 테스트는 실제 임베디드 시스템이 운용되는 환경에서 발생하는 결함에 대하여는 역부족이며 실행 기반의 동적 테스트가 매우 중요하다.

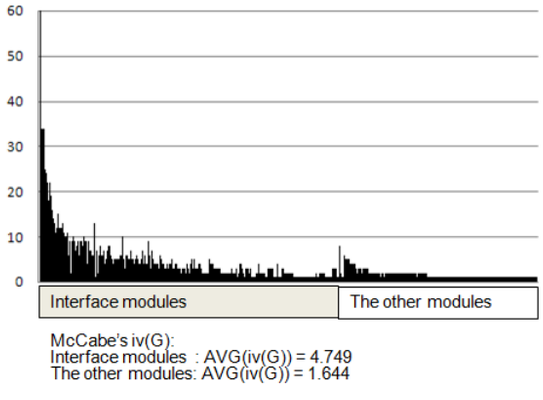
1. 인터페이스 테스트

임베디드 소프트웨어 테스트에서 인터페이스가 핵심 위치임을 보이기 위해선, 인터페이스가 결함 발생 가능성에 미치는 영향과 실제 알려진 임베디드 소프트웨어 결함 중 인터페이스 결함이 많은 비중을 차지하고 있음을 분석할 필요가 있다. 이를 위해 본 고에선 다음과 같이 임베디드 시스템의 인터페이스 복잡도와 인터페이스 결함을 분석한다.

가. 인터페이스 복잡도

인터페이스가 결함과 깊은 연관이 있음을 보이기 위해, 임베디드 시스템의 이질적 계층 사이의 인터페이스 복잡도를 측정하였다. 소프트웨어 복잡도는 결함 예측을 위한 메트릭으로 잘 알려져 있으며, McCabe의 연구[4]를 통해 결함이 복잡도와 강한 연관이 있음이 증명된 바 있다. McCabe의 다양한 복잡도 메트릭 중 모듈 설계 복잡도 (Module Design Complexity: iv(G)) 메트릭[4]을 사용했으며 정확성을 위해 McCabe IQ 테스트 도구를 이용하여 측정하였다.

ARM9T 플랫폼 기반의 스마트폰용 시스템 소프트웨어 6개를 대상으로 한 분석 결과에 따르면, 〈그림 1〉과 같이 인터페이스를 포함한 모듈에 대한 iv(G) 평균 값 (4.749)이 포함하지 않는 다른 모듈에 대한 iv(G) 평균 값 (1.644)보다 훨씬 높게 나타났으며, 이는 인터페이스 모듈이 더 복잡하고 모듈간 의존도가 더 높음을 의미한다. 즉, 본 결과는 인터페이스가 모듈간 의존성과 복잡도에 영향을 미치며, 인터페이스에서



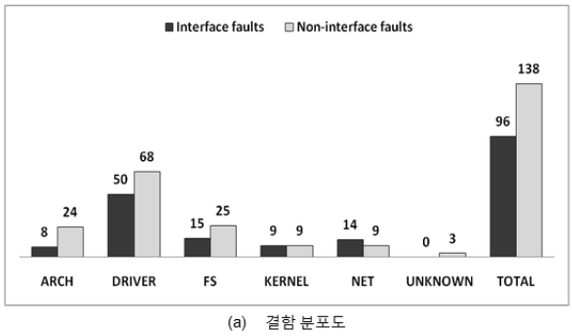
〈그림 1〉 인터페이스 복잡도

결함 발생 가능성을 증가시킬 수 있음을 보인다.

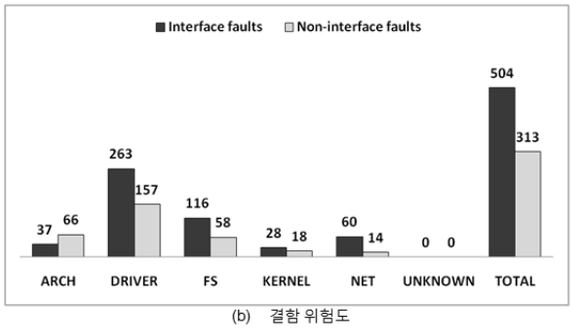
나. 인터페이스 결함

앞의 인터페이스 복잡도 분석은 인터페이스가 시스템의 복잡도를 증가시켜 결함 발생 가능성을 높임을 보였다. 이제 실제로 '얼마나 많은 결함이 인터페이스에서 발생하고 있는지'와 '얼마나 복구하기 어려운 결함인지'를 살펴보기 위해 오픈 소스인 리눅스 v2.4 플랫폼 기반 임베디드 소프트웨어에서 실제 보고된 결함들을 분석하였다 [5]. 특히 '얼마나 복구하기 힘든 결함인지'를 분석하기 위해 다음과 같이 '수정 횟수, 디버깅 의존도'의 디버깅 노력들을 가늠할 수 있는 기준들로 결함 위험도를 측정하였다.

〈그림 2〉-(a)에서 보듯이 전체 결함 중 96개인 41%가 인터페이스에서 발생하였다. 특히 커널 계층과 하드웨어 계층 양쪽에 모두 밀접히 결합된 디바이스 드라이버의 경우 인터페이스 결함 비중이 매우 높게 분석되었다. 〈그림 2〉-(b)는 비중 상으론 41%에 해당하는 인터페이스 결함이 위험도 면에선 62%를 차지하며, 다른 결함보다 인터페이스 결함이 더 위험함을 확인할 수 있다.



(a) 결함 분포도



(b) 결함 위험도

〈그림 2〉 인터페이스 결함

2. 동적 테스트

임베디드 소프트웨어의 런-타임 운용 환경은 시스템 자원 제약이 심하기 때문에 '테스트 대상, 테스트 환경, 테스트 성능, 테스트 범위' 측면에서 일반 소프트웨어 테스트와는

달리 반드시 해결되어야 할 기술적 요구 조건들이 있다. 동적 임베디드 소프트웨어 테스트를 위해 준수해야할 조건들을 정리하고 이를 기반으로 반드시 확인해야 할 주요 동적 테스트 결함 유형을 살펴본다.

가. 동적 테스트 조건

• 테스트 대상: 배포이진코드

동적 테스트는 소스 코드 존재 여부, 디버깅 정보 포함 여부와 최적화된 코드 사용 여부에 상관없이 최종 배포된 이진 코드를 대상으로 해야 한다. 또한 시스템의 고유의 실행 방식을 보장하기 위해 테스트 대상의 원본 소스나 이진 코드를 변경하지 않고 시스템에 로딩되어 실행되는 이진 이미지를 그대로를 대상으로 하는 테스트이어야 한다.

• 테스트 환경: 테스트 목적의 테스트 환경 변경 최소화

일반적인 테스트 도구들은 디버깅 목적의 메모리를 추가로 설계하거나 테스트 도구 상에서 프로그램을 구동시켜 테스트한다. 그러나 시스템이 실제 배포되어 운용되는 환경과의 차이는 테스트 오버헤드를 발생시키며, 실시간에 민감한 임베디드 시스템에선 이러한 테스트 오버헤드로 인해 추가로 문제가 발생할 수 있다. 즉 테스트를 위한 별도의 하드웨어 추가나 의존성이 없으며, 테스트를 위해 특별히 제작된 커널이나 가상머신을 사용하지 않는 동일한 소프트웨어와 하드웨어 환경에서 테스트되어야 한다.

• 테스트 성능: 시스템 오버헤드 최소화

많은 경우 임베디드 시스템은 제품 단가를 낮추기 위해 최적화된 메모리 공간 내에서 운용되도록 설계된다. 즉 테스트를 위한 별도의 메모리 공간이나 CPU를 고려하여 설계하지 않기 때문에 최소한의 자원을 사용한 테스트가 필요하다.

• 테스트 범위: 시스템 내의 모든 소프트웨어

임베디드 소프트웨어의 런-타임 동적 테스트는 탑재된 소프트웨어들이 독립적이지 않고 긴밀히 상호작용하기 때문에 시스템에 탑재된 모든 소프트웨어들의 행위가 테스트되어야 한다.

나. 동적 테스트 결함

시스템 런-타임 동적 테스트를 통해 검증해야할 주요 문제들은 시스템이 다운되거나, 정지해있는 듯 보이거나, 시스템이 느려지고 중간 중간 멈추는 듯한 증상을 보이는 결함이다. 이러한 문제의 80% 이상이 잘못된 '메모리, 프로세스간 통신, 성능'과 관련된 결함이다.

• 메모리 결함

메모리 결함은 오랜 기간 동안 시스템을 운용할 때 발생하는 잠재 문제나 시스템 다운의 주요 원인으로 잘 알려져 있다. 시스템의 메모리 현황과 연관 깊은 메모리 할당 실패, 메모리 누수, 메모리 접근 권한 문제, 잘못된 포인터 사용과 같은 메모리 결함을 발견할 수 있어야 한다.

• 프로세스간통신 결함

시스템이 다운되지 않았음에도 정지해있는 듯한 상태는 프로세스간통신 문제와 연관이 깊다. 소프트웨어 컴포넌트들은 이들 사이의 효율적인 통신을 위해 '메시지 교환, 메시지 큐, 공유 메모리 및 동기화 객체'와 같은 프로세스간통신 메커니즘들을 사용하여 상호운용한다. 잘못된 프로세스간통신 매커니즘으로 구현되었거나 스케줄링 문제와 연관한 동기화 문제를 발견할 수 있어야 한다.

• 성능 결함

시스템이 느려지거나 잠시 멈췄다 진행하는 증상은 시스템 성능 병목과 관련이 있다. 성능 병목의 원인은 매우 다양하긴 하지만, 일반적으로 메모리가 원인이 되는 성능 병목은 사용 가능한 메모리가 부족할 때에 발생할 수 있으며, 메모리 부족 증후를 식별할 수 있는 핵심 메모리 성능 카운터로는 페이지폴트율과 메모리 사용률이 있다. 프로세서가 원인이 되는 성능 병목을 식별하기 위한 카운터로는 시스템 실행 시간, 어플리케이션 실행 시간, 커널의 서비스 처리 시간이 있다. 시스템 이상 증상의 이유가 병목에 있음을 판단하기 위해 이들 성능 카운터들을 수집할 수 있어야 한다.

Ⅳ. 적용 성공 사례

테스트 활동은 자동화가 뒷받침되지 않는다면 산업체 현장에선 현실적으로 수행하기 힘든 활동이 된다. 우리는 인터페이스 기반 동적 테스트를 지원하는 임베디드 테스트 자동화 도구로 Justitia[6,7]와 AMOS[8]를 개발하였다.

현재 Justitia는 스마트폰용 플랫폼의 시스템 소프트웨어 테스트에 적용되고 있으며, AMOS는 차량 인포테인먼트 시스템 실차 테스트에 적용되고 있다. 본 절에선 도구별로 산업체 적용 사례 1건씩을 소개하고 적용 효과를 기술한다.

1. 스마트폰 플랫폼을 위한 디바이스 드라이버 테스트

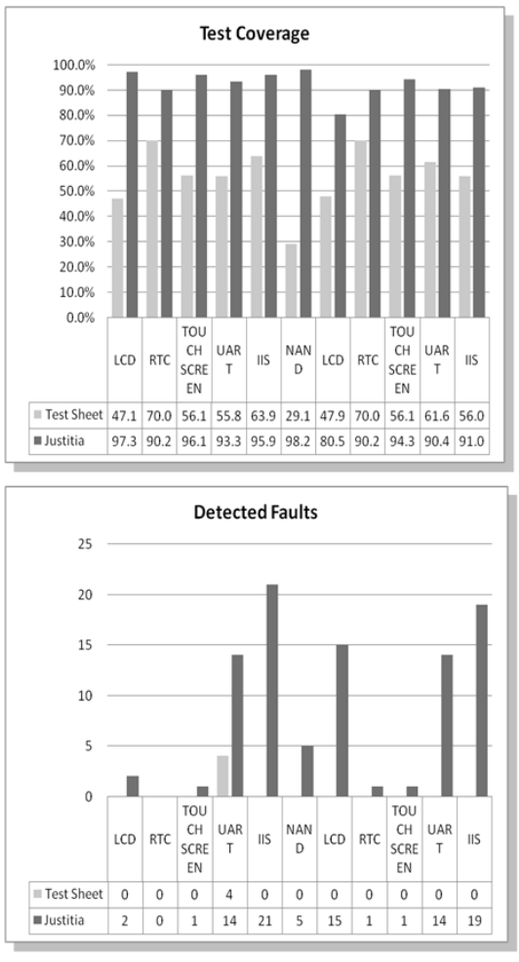
우리는 Justitia를 이용한 인터페이스 기반 디바이스 드라

이버 테스트를 통해 2006년 2개 버전의 AP (Access Point)의 11개의 디바이스 드라이버들을 대상으로 총 93개의 인터페이스 결함을 발견하였다.

가. 인터페이스 중심의 테스트 커버리지 증가를 통한 잠재 결함 발견 사례

디바이스 드라이버는 스스로 구동하기 힘들며 관련된 어플리케이션에 의해서만 실행된다. 실제 대부분의 현장에선 디바이스 드라이버를 테스트하기 위해 디바이스 드라이버를 구동시킬 어플리케이션과 통합한 상태에서 테스트하고 있다. 예를 들어 SOUND 디바이스 드라이버 테스트를 위해 Media Player 어플리케이션을 실행시켜 테스트한다. 그러나 이러한 테스트는 디바이스 드라이버를 중심으로 한 테스트가 아닌 어플리케이션 기능을 중심으로 한 테스트가 될 수밖에 없다. 우리는 기존 방식의 디바이스 드라이버 테스트와 Justitia를 이용하여 인터페이스 기반 테스트를 비교 수행하였다.

〈그림 3〉의 결과에서 보듯이 기존 방식의 테스트를 통해 평균 55.8% 커버리지에 4개의 결함을 발견하였고, Justitia를 통해선 평균 92.5% 커버리지에 93개의 결함을 발견하였다. 즉 기존 방식의 어플리케이션 기능을 중심으로 한 테스트로는 디바이스 드라이버의 모든 기능들이 테스트되지 못하고 있었으며, 테스트되지 않은 영역에서 결함이 잠재되어 있음을 알 수 있다. 개발자는 테스트를 충분히 했다고 믿었지만 잠재 결함은 결국 사용자 손에 의해 발견될 것이며 이는 제품의 신뢰도를 떨어뜨리는 치명적인 문제일 수밖에 없다.



〈그림 3〉 스마트폰 플랫폼을 위한 디바이스 드라이버 테스트 결과

나. 적용 효과

• 초기 개발품질 개선

평균 30~70%이던 테스트 커버리지를 90~97%까지 높임으로 잠재 결함을 발견하고 초기 개발 품질을 높일 수 있었다.

• 개발일정 단축

디바이스 드라이버를 구동시킬 어플리케이션이 준비되지 않더라도 Justitia를 통해 초기에 결함 발견할 수 있으며 평균 12개월 걸리던 개발 일정을 약 2개월 정도 앞당기는 효과를 보았다.

• 개발비용 절감

개발일정 단축으로 인해 AP 한 버전 당 약 80억원의 비용 절감 효과를 보았고, 2006년 한해 기준으로 3개 AP 버전을 출시하여 총 약 240억원 정도의 경제적 효과를 거둔 것으로 보고되었다.

2. 차량 인포테인먼트 시스템 실차 테스트

우리는 AMOS 의 메모리 테스트를 약 23개의 차량 모델에 현장 적용한 결과, 2008년 10월부터 현재까지 실시된 실차 테스트에 대해 총 519개의 메모리 결함들을 발견하였다. 성능 테스트는 AMOS v2가 개발된 2009년 10월부터 현장에 적용되어 해마다 새로 개발되는 플랫폼들에 대해 2번의 성능 튜닝을 수행하였다. 최근 확장된 IPC 테스트에 대해선 2010년 10월부터 현재 2개의 모델에 적용되어 총 64개의 IPC 결함을 발견하였다.

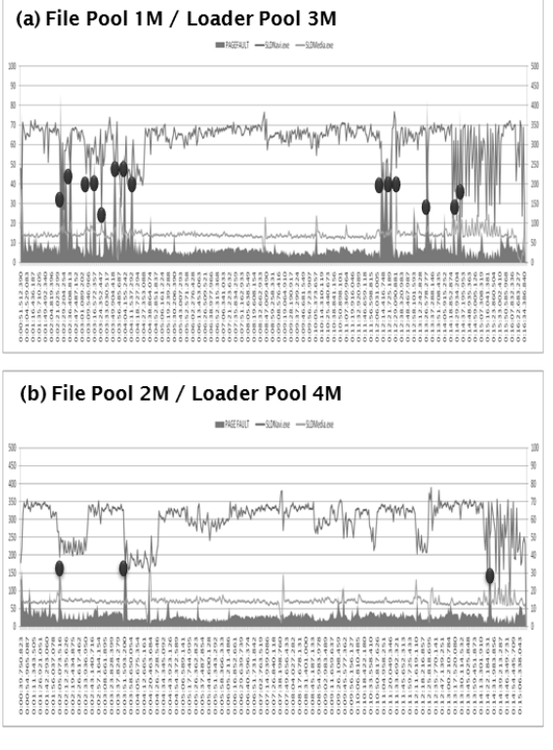
가. 간헐적 화면 정지 증상에 대한 성능 개선 사례

차량 주행 중에 흔히 겪는 내비게이션 이상 증상 중 하나가 간헐적으로 내비게이션 화면의 일부가 갱신되지 못하고 잠시 깨져보이거나 주행 속도에 맞추지 못하고 화면이 멈춰 있다 건너뛰듯이 화면이 갱신되는 증상이다. 내비게이션 개

발자들에겐 심각하지만 그 원인을 찾기 난해한 성능 결함 중 하나이며, 이를 해결하기 위해 AMOS를 사용하여 CPU 사용률, 메모리 사용률, 페이지폴트율과 같은 성능 카운터들을 측정하였다. 그 결과 우리는 페이지폴트율이 과도하게 발생하는 경우 내비게이션 소프트웨어의 CPU 사용률이 낮게 떨어지는 패턴을 볼 수 있었다.

내비게이션 소프트웨어는 주행 시 특별한 사용자 명령이 아니더라도 주기적으로 전송되는 GPS 좌표에 따라 주변 화면을 갱신한다. 이를 위해 실험 대상 시스템의 경우엔 평균 45% 정도의 CPU 사용률을 지속적으로 유지하는 것이 정상이며, 이 값 이하로 떨어졌다는 것은 내비게이션 소프트웨어가 화면을 제대로 갱신하지 못했을 가능성을 보여준다. 〈그림 4〉-(a)에서 보듯이 내비게이션 소프트웨어의 CPU 사용률이 45%이하로 떨어질 때 페이지폴트률이 300/sec 이상인 경우가 많았으며, 4(b)처럼 시스템의 페이지풀을 기본 설정 (file pool 1M, loader pool 3M)에서 2M 증가 (file pool 2M, loader pool 4M) 시키니 내비게이션 소프트웨어의 CPU 사용률이 기본 성능인 45%이하로 떨어지는 비율을 현저히 줄일 수 있었고 화면이 매끄럽지 못하게 넘어가는 현상을 개선할 수 있었다.

일반적으로 시스템의 성능 문제가 나타나면 CPU 사용률이 높은 코드를 튜닝하거나 메모리 자원을 증가시키는 방법으로 문제를 해결하려한다. 그러나 이 사례와 같이 메모리를 증가시키지 않고 페이지풀 설정을 대상 시스템의 특성에 맞게 적절히 조절하는 것만으로도 시스템 성능을 현격히 개선할 수 있음을 보여준다.



〈그림 4〉 차량 인포테인먼트 시스템 성능 개선 실험 결과

나. 적용 효과

• 필드 크레임 감소

차량 인포테인먼트 시스템과 관련된 필드 크레임이 약 5% 정도 감소하였고, 이에 따른 AS비용 감소와 신뢰도 향상으로 인한 매출 증대 효과가 가시화되고 있다.

• 개발비용 절감

AMOS 적용 효과는 카인포테인먼트 시스템의 품질 개선뿐만아니라 개발 비용 절감의 직접적인 경제적 효과로도 나타나고 있다. 2009년 한해 기준으로 현대기아자동차에서 실시된 AMOS 적용 효과 분석에 따르면, 총 11개 차종의 20개 모델에 적용한 결과, 아웃소싱 비용 절감, 매출 증대, 인건비 절감의 항목으로 약 20억원 정도의 경제적 효과[9]를 거둔 것으로 보고되었다.

V. 결 론

최근 품질과 직결되는 소프트웨어 테스트 기술은 급속도로 발전하고 있으나 이 기술들을 임베디드 소프트웨어 테스트에 바로 적용하기엔 알아야할 임베디드 소프트웨어의 특징과 해결되어야 할 테스팅 전략이 있다. 자원제약이 심한 임베디드 소프트웨어의 대표적 특징은 시스템 내의 하드웨어와 소프트웨어 컴포넌트들을 긴밀히 연계되도록 만들고, 독립적인 실행을 방해하기 때문에 시스템 통합을 통해서 비로소 활성화되는 상호의존적 기능들이 대부분을 차지한다. 이러한 특징은 통합되는 인터페이스에서 결함 발생 확률을 높이기 때문에 인터페이스를 중심으로 한 테스트 전략이 필요하며, 코드 기반의 정적 분석보다는 실제로 어떻게 상호작용하는지를 검증하는 동적 테스트가 중요하다.

본 고에서는 임베디드 소프트웨어에 특화된 테스팅 전략 수립을 위해 일반 소프트웨어와 차별화되는 임베디드 소프트웨어 특징들과 테스트 어려움을 분석하고, 이를 극복하기 위해 인터페이스를 중심으로 한 동적 테스트에 대해 기술하였다. 또한 이를 자동화한 테스트 도구인 Justitia와 AMOS를 산업 현장에 적용한 사례를 소개함으로 인터페이스 기반 동적 테스트의 효과를 살펴볼 수 있었다.

특히 AMOS 테스트 도구는 2008년 10월 이후 현대기아자동차의 차량 인포테인먼트 시스템의 표준 양산 검증도구로

채택되어 관련된 임베디드 소프트웨어 인수 기준으로 적용되고 있으며[9], 현재 런-타임 결함을 실시간 능동 방어하는 연구로 확장 중에 있다.

참고문헌

[1] A.A. Jerraya and W. Wolf, Hardware/software Interface Codesign for Embedded Systems, IEEE Computer, pp.75-84, 2005.

[2] B. Broekman, E. Notenboom, Testing Embedded Software, Addison-Wesley, 2003.

[3] J. Seo, Y. Ki, B. Choi, K. La, Which Spot Should I Test for Effective Embedded Software Testing?, IEEE Conference on Secure System Integration and Reliability Improvement (SSIRI 2008), pp.135-142, Yokohama Japan, 2008.

[4] W. Arthur and M. Tom, NIST document - "Structured testing: a testing methodology using the cyclomatic complexity metric", available in http://www.mccabe.com/iq_research_metrics.htm

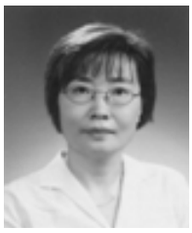
[5] Linux kernel Change Log, available in http://www.kernel.org/pub/linux/kernel/http://linux.bkbits.net:8080

[6] 임베디드 소프트웨어의 인터페이스 자동 추출 장치 및 그 방법, 특허등록 (10-1019209), 2011.02.24.

[7] Appratus and Method For Automatically Extracting Interface of Embedded Software, 미국특허출원 (IB17153-US) 등록결정 2011.12.20.

[8] 온라인 시스템 테스트 방법, 국내특허등록번호: 10-1091457, 2011.12.01.

[9] 최병주,서주영, 양승완.오정석, 동적임베디드 카인포테인먼트 시스템 테스트 적용 성과 , 정보과학회지 29권 9호, pp48-56 2011.9



최 병 주

1983년 2월 이화여자대학교 수학과 학사.
1987년 8월 Purdue Univ. Computer Sciences 석사.
1990년 12월 Purdue Univ. Computer Sciences 박사.
1991년 3월~1992년 2월 삼성종합기술원 선임연구원.
1992년 3월~1995년 2월 용인대 전산통계학과 조교수.
1995년 3월~현재 이화여자대학교 컴퓨터공학과 교수.
〈관심분야〉 소프트웨어공학, 소프트웨어 테스트, 임베디드 소프트웨어 테스트, 소프트웨어 아키텍처 평가와 성능테스트

서 주 영

1993년 2월 이화여자대학교 전자계산학과 학사.
2001년 2월 이화여자대학교 컴퓨터공학과 석사.
2009년 2월 이화여자대학교 컴퓨터공학과 박사.
1992년 12월~1997년 6월 삼성전자 반도체사업부 연구원.
2010년 3월~2010년 8월 이화여자대학교 컴퓨터공학과 연구교수.
2010년 9월~현재 아주대학교 정보컴퓨터공학부 강의교수.
〈관심분야〉 소프트웨어공학, 소프트웨어 테스트, 임베디드 소프트웨어, 테스트 자동화