제 4 장 품질 특성과 비기능 테스트

4.1 개 요

이 절에서는 ISO 25010에 따라 품질 특성을 테스트하는 방법에 대해 간략하게 설명한다. ISO 25010 품질 모델에서는 소프트웨어 품질 특성을 다음과 같이 8가지 주특성과 각 주특성에 따른 하위 품질 특성을 정의하고 있다(그림 4.1 참조).

- **기능 적합성**(Functional Suitability): 제품 또는 시스템이 명시적 또는 묵시적 요구를 충족시키는 기능을 제공하는 정도
- **성능 효율성**(Performance Efficiency): 명시된 조건하에서 사용된 자원의 양에 대한 성능의 정도
- **호환성**(Compatibility): 제품, 시스템 또는 구성 요소가 다른 제품, 시스템 또는 구성 요소와 정보를 교환하거나 필요한 하드웨어 또는 소프트웨어 환경을 공유하면서 필요한 기능을 수행할 수 있는 정도
- **사용성**(Usability): 특정한 사용자들이 주어진 사용 환경(특정 사용 컨텍스트)에서 특정한 목적을 달성하기 위해 제품이나 시스템을 사용할 때의 효율성, 효과성 및 만족도에 대한 정도
- **신뢰성**(Reliability): 특정 조건에서 특정 기간 동안 오동작 없이 요구되는 기능을 수행하는 정도
- **보안성**(Security): 시스템이 정보 및 데이터를 보호하는 정도
- **유지보수성**(Maintainability): 제품 또는 시스템이 유지보수 될 수 있는 효율성의 정도
- **이식성**(Portability): 다양한 플랫폼에서 운영될 수 있는 소프트웨어의 능력

그림 4.1 ISO 25010 품질 특성

표 4.1은 부특성에 대한 설명이다. 이러한 품질 특성에 따라 다양한 테스트 설계 기술을 사용하여 테스트 케이스를 설계할 수 있다.

표 4.1 ISO 25010 품질 특성

주특성	부특성	설명
기능 적합성 (Functional Suitability)	기능 완전성 (Functional completeness)	기능 집합이 모든 명시된 요구사항을 포괄하는 정도
	기능 정확성 (Functional Correctness)	시스템이 정의된 정밀도로 정확한 결과를 제공하는 정도
	기능 적절성 (Functional Appropriateness)	기능이 목적 달성에 도움을 주는 정도
성능 효율성 (Performance Efficiency)	시간 반응성 (Time-behaviour)	기능 수행 시 시스템의 응답・처리 시간 및 처리율이 요구사항을 충족시키는 정도
	자원 효율성 (Resource Utilization)	기능 수행 시 시스템이 사용하는 자원이 요구 사항을 충족시키는 정도
	수용성 (Capacity)	시스템 매개 변수(동시 사용자 수, 통신 대역폭, 트랜잭션 처리량 양 등)의 최대 한계가 요구사항을 충족시키는 정도
호환성 (Compatibility)	공존성 (Co-existence)	다른 소프트웨어에 나쁜 영향을 미치지 않고 자원을 공유하면서 요구되는 기능을 효율적으로 수행할 수 있는 정도
	상호운영성 (Interoperability)	둘 이상의 시스템 또는 구성 요소가 정보를 교환하고 교환된 정보를 사용할 수 있는 정도
사용성 (Usability)	적합 인식성 (Appropriateness Recognisability)	사용자가 자신의 필요에 시스템이 적합한지 여부를 인식할 수 있는 정도
	학습 용이성 (Learnability)	사용자가 소프트웨어의 사용법을 배워 명시된 목적을 달성할 수 있는 정도
	운영 용이성 (Operability)	시스템이 쉽게 조작하고 제어할 수 있는 속성을 갖는 정도
	사용자 오류 방지성 (User Error Protection)	시스템이 사용자로 하여금 오류를 범하지 않게 하는 정도
	사용자 인터페이스 심미성 (User Interface Aesthetics)	사용자 인터페이스가 사용자에게 만족스러움을 주는 정도
	접근성 (Accessibility)	사용자의 특성이나 능력(예 연령과 장애)에 관계없이 시스템을 사용할 수 있는 정도

신뢰성 (Reliability)	성숙성 (Maturity)	시스템 또는 구성 요소가 정상 작동 상태에서 신뢰성 요구를 충족시키는 정도
	가용성 (Availability)	사용자가 시스템 또는 구성요소를 사용하고자 할 때 사용 및 접근이 가능한 정도
	결함 허용성 (Fault Tolerance)	하드웨어나 소프트웨어 결함이 있음에도 불구하고 시스템 또는 구성 요소가 의도한 대로 작동하는 정도
	복구성 (Recoverability)	중단 또는 장애가 발생한 경우 시스템이 영향을 받은 데이터를 복구하고 상태를 재설정할 수 있는 정도
보안성 (Security)	기밀성 (Confidentiality)	접근 권한이 있는 사람에게만 데이터에 액세스할 수 있도록 하는 정도
	무결성 (Integrity)	시스템 또는 구성 요소가 컴퓨터 프로그램 또는 데이터에 무단으로 접근하거나 이의 변경을 방지하는 정도
	부인 방지성 (Non-Repudiation)	사건 및 행위 후에 부인하지 못하도록 행동 및 사건을 입증할 수 있는 정도
	책임성 (Accountability)	각 개인을 유일하게 식별하여 행위를 기록하고 필요 시 그 행위자를 추적할 수 있는 능력
	인증성 (Authenticity)	사건 및 행동에 대해 실제 행위자임을 증명할 수 있는 정도
유지보수성 (Maintainability)	모듈성 (Modularity)	하나의 구성 요소에 대한 변경이 다른 구성 요소에 미치는 영향이 최소화되도록 시스템 또는 컴퓨터 프로그램이 개별 구성요소로 구성된 정도
	재사용성 (Reusability)	시스템 자산이 하나 이상의 시스템에서 사용될 수 있는 정도, 또는 다른 자산을 구축할 수 있는 정도
	분석성 (Analyzability)	부분에 의도된 변경이 전체 시스템에 미치는 영향을 평가하거나 결함 또는 결함 원인에 대해 제품을 진단하거나 수정될 부분을 식별할 수 있는 정도
	변경 용이성 (Modifiability)	결함이나 품질 저하 없이 효과적이고 효율적으로 수정될 수 있는 정도
	테스트 용이성 (Testability)	테스트 수행을 용이하게 하는 정도
이식성 (Portability)	적응성 (Adaptability)	시스템이 다른 하드웨어, 소프트웨어 혹은 기타 사용 환경에 효과적이고 효율적으로 적용될 수 있는 정도
	설치 용이성 (Installability)	특정 환경에서 시스템을 성공적으로 설치 및 제거할 수 있는 정도
	대체 용이성 (Replaceability)	시스템이 동일한 환경에서 동일한 목적을 위해 다른 지정된 소프트웨어 제품으로 대체될 수 있는 정도

4.2 기능 적합성 테스트

기능 적합성(Functional Suitability) 테스트는 사용자의 요구사항을 시스템이 얼마나 만족하는지에 대한 정보를 제공한다. 표 4.1에서 확인하였듯이 기능 적합성의 부특성으로는 기능 완전성, 기능 정확성, 기능 적절성이 있다.

부특성 중 기능 완전성은 사용자가 요구하는 기능을 얼마나 제공하는지를 보는 것으로, 기능 완전성 테스트는 명세 기반 테스트 방법으로 테스트할 수 있다. 사용자의 요구사항이 유스케이스나 사용자 스토리 등으로 표현되었을 때, 이로부터 테스트 케이스를 추출하고 요구사항과 테스트 케이스 간의 추적성 정보를 유지함으로써 현재 시스템이 얼마만큼 기능을 제공하는지 파악할 수 있다.

기능 정확성은 시스템이 사용자가 기대하는 수준으로 얼마나 정확하게 동작하는지를 의미한다. 따라서 전체 소프트웨어 기능 중에서 사용자의 의도된 목적을 달성할 수 있을 정도로 정확하게 동작하는 기능의 수로 정확성을 판단할 수 있다. 기능 정확성은 명세 기반이나 구조 기반 테스트 방법을 모두 사용할 수 있다.

기능 적절성은 사용자의 사용 목적을 달성하는 데 도움을 주는 정도를 의미한다. 기능 적절성 테스트의 대상이 되는 목적을 나열하고 각 목적별로 적절성을 테스트한다. 예를 들어, 해당 목적을 달성하는 데 필요한 기능이 누락되지는 않았는지 또는 올바르게 구현되었는지 테스트한다.

4.3 성능 효율성 테스트

소프트웨어 시스템의 성능을 평가할 때 고려해야 할 요소로는 CPU 사이클, 디스크, 주 기억공간과 같은 자원의 사용, 주어진 시간 동안 처리할 수 있는 작업량, 자원이 할당되기를 기다리는 태스크의 수 등이 있다. 이러한 요소들은 시스템 특성에 따라 주로 결정된다. 예를 들면, 데이터베이스를 주로 사용하는 시스템에서는 데이터베이스에 접근하는 비율이 시스템의 성능을 결정하는 중요한 요소가 될 수 있다.

따라서 이러한 요소를 포함하여 성능에 관한 요구사항은 구체적이면서 검증 가능한 형태로 명세서에 기술되어야 한다. 그러나 많은 경우 성능에 관한 요구사항이 명세서에 명시적

으로 기술되어 있지 않거나 성능에 관한 요구사항이 기술되었다 할지라도 그러한 요구 사항들이 만족되었는지 아니면 만족되지 않았는지 검증이 불가한 형태로 기술되는 문제가 발생한다.

다음과 같이 작성된 컴파일러의 성능 요구사항을 생각해보자: "컴파일러는 어떤 모듈이든 1초 이내에 컴파일할 수 있어야 한다." 이러한 요구사항의 문제점은, 만족하지 않음은 쉽게 보일 수 있지만, 만족됨을 보장하기란 매우 어렵다는 점이다. 예를 들어, 컴파일 시간이 1초 이상 걸리는 모듈을 선정하면 컴파일러의 성능이 요구사항을 만족하지 않음을 보이기는 매우 쉽다. 그러나 성능 테스트를 위해 사용된 다수의 모듈이 1초 이내에 컴파일된다고 해서 컴파일러가 모든 모듈을 1초 이내에 컴파일할 수 있다고 단정할 수는 없다.

그렇다면 어떻게 성능에 관한 요구사항을 기술해야 하는가? 앞에서 언급하였듯이 검증 가능한 형태로 기술되어야 한다. 예를 들어, "평균적인 부하가 있는 상황에서 시스템을 가동할 때 CPU 사용률이 50%를 초과해서는 안 된다."와 같이 기술할 수 있다. 또 다른 검증 가능한 성능 요구사항 명세의 예로, "관찰된 기간 동안에 최대 부하가 걸린 상황과 동일한 상태에서 CPU 사용률이 90%를 초과해서는 안 된다."와 같이 작성할 수 있다.

이와 같이 검증 가능한 형태로 요구사항을 작성하려면 실제로 사용자들이 시스템을 사용하는 패턴인, 시스템 운영 프로파일(Operational profile)이 필요하다. 만약 시스템이 처음 개발되는 경우이거나 시스템의 사용 정보를 수집하지 못한 경우와 같이 시스템의 운영 프로파일을 이용할 수 없는 상황이라면, 주위에 유사한 시스템이 있는지 살펴보아야 한다. 이마저도 여의치 않다면 전문가팀이 부하 상황을 추정하여 예측할 수밖에 없으며 정보가 추가되어 감에 따라 요구사항들이 좀 더 정제될 수 있을 것이다.

시스템의 성능을 테스트하는 전통적인 방법으로는 벤치마크(Benchmark)의 개념을 사용하는 벤치마크 테스트(Benchmark test)가 있다. 벤치마크 테스트란, 실존하는 비교 대상을 두고 하드웨어나 소프트웨어의 성능을 비교 시험하고 평가하는 것으로, 일반적인 성능 테스트와는 달리 실제와 상황이 같은 시험 환경에서 한 개 또는 여러 개의 대표적인 비교 대상과 비교 시험을 반복하여 성능을 평가하는 방법이다.

벤치마크는 시스템이 동작되는 운영 환경 및 작업 부하를 대표해야 한다. 예를 들면, 운영 체제의 성능을 테스트할 때 100명의 사용자가 사용하는 것이 일반적인 패턴이라면 벤치마크는 이 패턴을 반영해야 한다. 그리고 평균 작업 부하 상태에서 CPU 활용도가 50%를 초과하지 않아야 한다는 성능 요구사항이 있다면, 이 요구사항을 시스템이 만족하는지 검증

하기 위해 100명의 사용자가 동시에 시스템을 사용하는 환경이 있어야 하고, 동일한 작업 부하를 시스템에 줄 수 있어야 한다.

또한, 성능 테스트는 시스템에 가해지는 평균적인 작업 부하뿐만 아니라 극단적 작업 부하가 걸린 상태도 고려해야 하므로 벤치마크를 성능 테스트에 사용하는 경우에는 사용되는 작업 부하 패턴을 정확하게 반영할 수 있도록, 운영 프로파일(Operational profile) 형태로 제공되는 것이 바람직하다.

벤치마크를 개발할 때 추가로 고려할 사항은 평균 부하(또는 최고 부하)가 걸린 상황을 어느 기간 동안 반영할 것인지를 결정하는 일이다. 즉, 1시간 동안 시스템에 걸린 부하 상황을 반영할 수도 있고, 24시간 동안의 부하 상황을 반영할 수도 있다. 경우에 따라서는 일주일 또는 한 달 동안 시스템 부하 상황을 보고 벤치마크를 개발할 수도 있다. 중요한 점은 반복성(Repeatability)이다. 예를 들어, 일주일 동안 시스템 운영 상황이 다음 일주일 동안에도 유사하게 발생한다면 일주일 동안 시스템에 걸린 부하 상황만 고려하여 벤치마크를 개발할 수 있다. 물론 공휴일이나 국가적으로 큰 재앙이 발생한 것과 같은 특수한 상황은 제외해야 한다.

표 4.2는 대표적인 성능 테스트 종류를 비교한 표이고, 그림 4.2는 이러한 성능 테스트의 차이를 가시화한 그래프이다.

표 4.2 **성능 테스팅 종류**

종류	설명
부하 테스팅 (Load testing)	부하를 계속 증가시키면서 시스템의 임계점을 찾는다. 임계점이란, 처리량이 더는 증가하지 않거나 CPU 이용률이나 메모리 사용량이 비정상적으로 증가하는 지점을 의미한다. 이 테스팅을 통해 병목 지점을 찾고 병목 현상을 제거하는 과정을 반복한다.
스트레스 테스팅 (Stress testing)	시스템 처리 능력 이상의 부하, 즉 임계점 이상의 부하를 가하여 비정상적인 상황에서의 처리를 테스트한다.
스파이크 테스팅 (Spike testing)	이 테스팅은 짧은 시간에 사용자가 몰릴 때 시스템의 반응을 측정한다. BTS와 같은 세계적인 아이돌 그룹의 공연 예매를 지원하는 시스템에 짧은 시간에 수많은 사용자가 몰리는 상황을 생각해볼 수 있다.
내구성 테스팅 (Endurance testing/ Soak testing)	오랜 시간 동안 시스템에 높은 부하를 가하여 시스템의 반응을 파악한다.

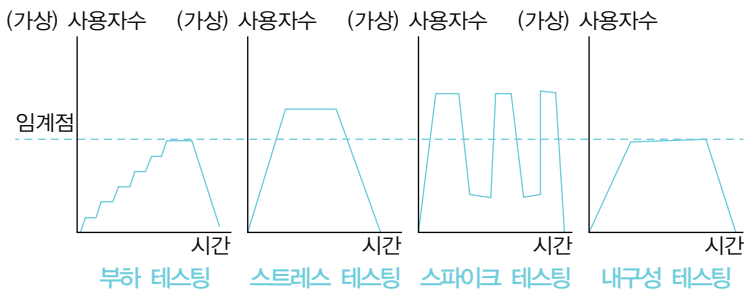

그림 4.2 성능 테스팅 비교

심화노트 Little's law와 성능 테스트

리틀의 법칙은 성능 테스트에서 반드시 알아야 하는 매우 유명한 법칙이다. 리틀의 법칙은 발명자 존 리틀 박사의 이름을 따서 만들었으며 매우 간단하고 직관적이다. 리틀의 법칙은 시스템에 오랜 시간 동안 머물러 있는 고객의 평균 수치는 오랜 시간 동안에 걸친 평균 실제 도착률과 시스템에서 고객이 머문 평균 시간을 곱한 값과 동일하다는 것이다. 성능 테스트 관점에서 이 법칙을 설명하기 위해 필요한 용어를 다음과 같이 정리한다.

용어	설명
동시 사용자 (Concurrent user)	Active user + Inactive user
Active user	요청 후에 응답을 기다리는 사용자
Inactive user	세션 정보를 가지고 있지만, 요청을 보내지 않는 사용자(e.g., 앞선 요청에 대한 결과를 보고 있는 사용자)
처리량 (Throughput)	단위 시간 동안 시스템(서버)에서 처리되는 요청 수이다. 보통 TPS(Transactions Per Second)로 나타낸다.
응답 시간 (Response time)	요청을 보낸 후 응답이 처리되어 결과가 사용자에게 전달될 때까지 걸리는 시간이다.
씽크 타임 (Think time)	요청을 보낸 후에 응답 결과를 수신하고 다음 요청을 보낼 때까지 걸리는 시간이다.
요청 간격 (Request interval)	요청을 보낸 후에 다음 요청을 보낼 때까지 걸리는 시간이다. Request interval = Response time + Think time

이러한 정의를 바탕으로 리틀의 법칙을 성능 테스트 관점에서 기술하면 아래 공식들로 표현된다.

```
- Concurrent user = 처리량(TPS)*요청 간격(Request interval)
                  = 처리량(TPS)*(Response time + Think time)
- Active user = 처리량(TPS)*Response time
- Inactive user = 처리량(TPS)*Think time
```

리틀의 법칙은 성능 테스트에서 사용되는 목표 처리량에 요구되는 동시 사용자 수를 산정할 때 사용한다. 동시 사용자 수를 너무 적게 산정하면 실제 시스템에서 발생할 수 있는 상황과는 거리가 먼 테

스트 결과가 나올 수 있으며 너무 많이 산정하면 성능 테스트에 필요 이상으로 비용이 들게 된다.
이해를 돕기 위해 예를 들어보자. 만약 목표 처리량이 50TPS이고 시스템 평균 응답 시간(think time은 0라고 가정한다)이 2초라면 성능 테스트에 필요한 가상 사용자 수는 100이다. 따라서 100명의 가상 사용자를 대신하는 쓰레드를 생성하여 성능 테스트를 수행할 필요가 있다.
리틀의 법칙과는 직접적인 관계는 없지만, 성능 테스트에서 자주 사용되는 용어에 Ramp up과 Ramp down이 있다. Ramp up은 가상 사용자를 이용하여 부하를 주는 패턴을 말하며, 가상 사용자를 동시에 유입할지 어느 정도 시간 간격에 따라 유입할지를 정의한다. 예를 들어, Ramp up 기간이 100초이고 10명의 가상 사용자가 있다면 100/10 = 10초 주기로 가상 사용자가 한 명 유입된다. 반면에 Ramp down이란 성능 테스트가 종료된 후에 가상 사용자 수를 줄이는 패턴이다.

4.4 호환성 테스트

ISO 25010 품질 모델은 호환성(Compatibility)의 부특성을 공존성(Co-existence)과 상호운영성(Interoperability)으로 설명하고 있다. 공존성은 다른 소프트웨어와 환경 및 자원을 공유하면서 요구된 기능을 효율적으로 수행하는 정도를 나타내며, 상호운영성은 여러 시스템이 정보를 교환하거나 교환된 정보를 성공적으로 사용할 수 있는 정도를 의미한다.

공존성이 유지되지 않을 때 발생할 수 있는 문제의 예로, 보안 프로그램을 여러 곳에서 다운로드 받아 설치할 때 기존 애플리케이션이 정상적으로 동작되지 않는 경우를 들 수 있다. 이러한 문제는 설치된 보안 프로그램들이 메모리와 같은 자원을 사용하여 충돌이 일어나거나 자원을 독점적으로 사용할 때 발생한다.

공존성은 특히 시스템의 장애 발생이 인명 손실이나 막대한 재산 피해, 또는 치명적인 환경 파괴를 가져올 수 있는 안전성 필수(Safety-critical) 시스템에서 매우 중요하다.

일례로, 자동차 기능 안전성 국제 표준인 ISO 26262에서는 다른 시스템 요소들이 공존하기 위한 기준을 위험 등급으로 엄격하게 규정하고 있다. ISO 26262 준수를 위한 핵심사항인 ASIL(Automotive Safety Integrity Level)은 위험 노출 가능성(E:Exposure, E0~E4), 통제 가능성(C:Controllability, C0~C3), 심각성 평가(S:Severity, S0~S3)를 기반으로 위험성을 평가한다. 이 세 가지 사항을 조합하여 도출된 위험성 등급은 ISO 26262 준수가 불필요한 QM(Quality Management), 그리고 가장 낮은 등급인 A등급부터 D등급까지로 분류된다.

상호운영성은 특별하게 노력을 기울이지 않아도 시스템 또는 제품이 다른 시스템이나 제품과 함께 잘 동작할 수 있는 능력을 의미한다. 이러한 상호운영성은 사물인터넷(IoT)이나

국방정보시스템을 구축할 때 매우 중요하다.

많은 전문가는 IoT 가치의 상당 부분은 다양한 IoT 시스템 간의 상호운영성 달성 능력에 달려 있다고 입을 모은다. 구글, 마이크로소프트, 애플, 시스코, 인텔, IBM 등 대형 IT 기업이 자신의 IoT 기술을 가지고 있다. 하지만 어느 한 기업이 보유한 IoT 솔루션만으로 모두를 만족시킬 수는 없다. 따라서 IoT 공급업체의 기술을 평가할 때 상호운영성이 핵심 기준이 될 수밖에 없다.

IoT 상호운영성을 위해 데이터 교환 및 관리 역할을 하는 국제표준 기반의 플랫폼 기술이 활발히 개발되고 있으며, 각 표준협의체에서는 각각의 인증프로그램을 운영하여 상호운영성 보장을 위한 표준에 적합한 제품이 시장에 출시되도록 유도하고 있다.

국방정보시스템은 지휘 통제 분야, 전략 전술 분야, 물자 관리 분야 등 다양한 분야에서 사용되고 있다. 군의 전력을 극대화하기 위해서는 다양한 국방정보시스템 간의 효과적인 상호운영성이 전제되어야 한다.

LISI(Level of Information System Interoperability) 모델은 국방정보시스템 간의 상호운영성을 평가하기 위한 모델로 미국방성에 연구비를 받아 카네기 멜론대학의 SEI 연구소에서 개발되었다. LISI(표 4.3 참조)는 시스템 간의 정보 교환 능력을 설명하는 상호운영성 성숙도 모델(Interoperability maturity model)을 보여 준다. 이 성숙도 모델은 요구되는 상호운영 능력을 절차, 응용, 인프라, 데이터 네 가지 측면으로 구분하여 0부터 5까지 여섯 단계 수준으로 구성된다.

표 4.3 LISI 능력 모델

수준		설명
전군적	5	전군의 도메인 간 자료공유와 협력의 가상공간을 제공하는 상호운영 수준
도메인	4	도메인 내부에서 개별적인 응용 프로그램으로 공유된 데이터베이스를 이용하여 상호운영이 가능한 수준
기능적	3	개별적인 응용 프로그램으로 이종의 자료에 대하여 상호운영이 가능한 수준
연결	2	동종의 자료를 이용하여 컴퓨터 간의 방식으로 이루어지는 상호운영 수준
불완전	1	상호운영이 이루어지고 있으나 사람이 디스켓 등을 이용하여 수동적으로 이루어지는 수준
격리	0	타 시스템과 상호운영이 이루어지지 않고 독립적으로 운영되는 수준

한국군에서도 LISI 모델을 바탕으로 상호운영성을 측정하고 있으며, LISI에 따른 상호운영성 평가를 위해 전자질의평가 방법을 주로 사용해 왔다. 이 방법은 미리 준비된 상호운영

에 관한 질의서에 개발 시스템 구현 정보 옵션을 입력하고, 질의서에 입력된 정보를 바탕으로 평가 대상 시스템의 상호운영 능력과 관련된 정보를 종합한 상호운영 프로파일을 작성한 후, 상호운영 프로파일을 바탕으로 상호운영 수준을 평가하는 것이다.

4.5 사용성 테스트

ISO 25010에서 사용성(Usability)은 특정한 사용자들이 주어진 사용 환경(특정 사용 컨텍스트)에서 특정한 목적을 달성하기 위해 제품이나 시스템을 사용할 때 얻게 되는 효율성, 효과성 및 만족도로 정의된다.

ISO 9241-11(Guidance on usability)에서도 이와 동일하게 사용성을 정의하고 있다. 표 4.4는 ISO 9241-11과 ISO 25010에서 설명된 효과성, 효율성 및 만족도에 대한 설명이다.

표 4.4 사용성 평가 품질특성

품질 특성	설명
효과성	사용자가 특정 목표를 달성하는 정확성 및 완전성
효율성	사용자가 목표를 달성하는 정확성 및 완전성과 관련하여 소비되는 자원(정신적 또는 육체적 노력, 시간, 재료 또는 재정적 비용 등)
만족도	지정된 사용 환경에서 사용자가 불편함이 없는 정도와 제품 사용에 대한 태도, 지정된 사용 환경에서 사용될 때 사용자 요구가 충족되는 정도

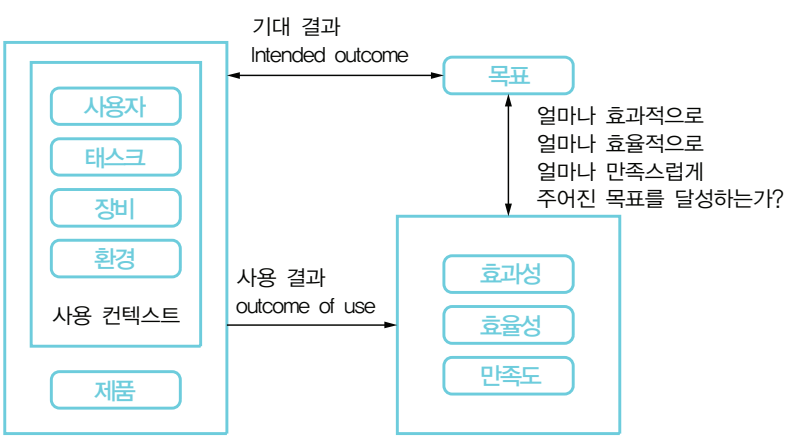

그림 4.3 ISO 9241-11 사용성 프레임워크

그림 4.3은 ISO 9241-11에서 규정한 사용성을 측정하고 평가하기 위하여 구성요소들을 측정·검증 가능하도록 분해하고, 그 관계를 표현한 프레임워크이다. 이 프레임워크에서 볼 수 있듯이 효과성, 효율성, 만족도는 시스템이나 제품의 사용성 수준을 측정하는 기준으로 사용될 수 있다.

표 4.5는 한국과학기술정보연구원(KISTI)에서 2011년 발간한 "웹사이트 사용성 개선을 위한 단계별 전략"에 관한 보고서에 소개된 ISO 9241-11의 효과성, 효율성, 만족도를 기반으로 하는 사용성 측정 예시이다.

표 4.5 사용성 측정 예시

측정항목	정의	측정방법
효과성	태스크 성공/실패 여부	[100-(등급별 만족도 * 오류 횟수)] 각 태스크 수행 후 만족도 수준 파악 • A(매우 만족): 0점 • B(만족): 5점 • C(보통): 10점 • D(불만): 15점 • E(매우 불만): 20점
효율성	태스크를 수행하는데 걸리는 시간	[태스크 수행 총 소요 시간 – 이용자와 연구자와의 인터뷰 진행 시간 제외]
만족도	각 태스크 수행 후 이용 난이도 수준 파악	• A(매우 만족): 100점 • B(만족): 90점 • C(보통): 80점 • D(불만): 70점 • E(매우 불만): 60점

ISO 25010에서 사용성은 다음 6가지 부특성으로 나뉘며, 사용성 평가에 이러한 특성을 고려할 수 있다.

- 적합 인식성(Appropriateness recognisability): 사용자가 자신의 필요에 시스템이 적합한지를 인식할 수 있는 정도
- 학습 용이성(Learnability): 사용자가 소프트웨어 사용법을 배워 명시된 목적을 달성할 수 있는 정도
- 운영 용이성(Operability): 시스템이 쉽게 조작하고 제어할 수 있는 속성을 갖는 정도
- 사용자 오류 방지성(User error protection): 시스템이 사용자로 하여금 오류를 범하지

않게 하는 정도

- **사용자 인터페이스 심미성**(User interface aesthetics): 사용자 인터페이스가 사용자에게 만족스러움을 주는 정도
- **접근성**(Accessibility): 사용자의 특성이나 능력(예 연령과 장애)에 관계없이 시스템을 사용할 수 있는 정도

현재 사용성 평가를 위한 다양한 방법이 개발되어 있으며 전문가가 행하는 휴리스틱 평가(Heuristic Evaluation), FGI(Focus Group Interview), 인지적 워크쓰루(Cognitive Walk-through), 설문(Questionnaire) 등의 방법이 주로 사용된다.

휴리스틱 평가(Heuristic Evaluation)는 사용성 평가 전문가가 제품과 관련된 사용성 원칙을 기준으로 체크리스트를 통해 사용성에 관한 문제점을 도출하는 방법이다. 3명~5명 정도 되는 적은 인원의 전문가를 활용하여 비교적 쉽게 실시할 수 있고, 중요한 사용성 문제를 많이 도출할 수 있으며, 효율적이고 빠르게 수행할 수 있다는 장점이 있다.

FGI(Focus Group Interview)는 그룹 인터뷰 방법이다. 주로 시스템 개발 이전에 사용자의 요구사항을 파악하는 데 많이 사용되는 정성적인 평가방법으로, 공통점이 있는 사용자들을 그룹별(7~8명)로 모아 시스템이나 문제점 등에 관해 의견을 나누고 필요한 정보를 수집하는 방법이다.

인지적 워크쓰루(Cognitive walk-through)는 학습 용이성 분석에 중점을 둔 방법이다. 실제 사용자를 대상으로 사전설명 또는 안내 없이 제품을 사용하여 주어진 과제를 달성하도록 한다. 그리고 과제를 달성하는 각 단계에서 적합한 행동을 취하는지 분석한다. 과제를 수행하는 각 단계마다 진행자는 다음과 같은 질문을 던진다. 여기서 액션은 버튼(컨트롤)에 마우스를 클릭하는 행위 등을 의미한다.

- 사용자는 원하는 목적을 달성하기 위해 이 액션을 수행하는 시도를 할 것인가?
- 사용자가 올바른 액션을 수행하기 위해 필요한 컨트롤이 가시적인가?
- 이 액션을 통해서 사용자가 과제를 달성할 수 있는지 알아낼 수 있는가?
- 사용자는 적절한 피드백을 얻는가?

이러한 사용성 평가는 시스템 개발의 전 주기에 걸쳐 진행되어야 하며, 평가 목적에 따라 평가방법을 달리하는 것이 좋다. 사용성 평가 결과 나타난 문제의 원인을 체계적으로 분석하고 시스템 개선에 반영한다면 이용자의 만족도를 높이는 시스템을 설계할 수 있다.

4.6 신뢰성 테스트

신뢰성(Reliability)은 특정 조건에서 특정 기간 동안 시스템이 요구되는 서비스를 오동작 없이 제공하는 정도를 말한다. ISO 25010에서 신뢰성은 다음과 같이 네 가지 부특성으로 설명되어 있다.

- **성숙성**(Maturity): 시스템 또는 구성 요소가 정상 작동 상태에서 신뢰성 요구를 충족시키는 정도
- **가용성**(Availability): 사용자가 시스템 또는 구성요소를 사용하고자 할 때 사용 및 접근이 가능한 정도
- **결함 허용성**(Fault tolerance): 하드웨어나 소프트웨어에 결함이 있는데도 시스템 또는 구성 요소가 의도한 대로 작동하는 정도
- **복구성**(Recoverability): 중단 또는 장애가 발생한 경우 시스템이 영향을 받은 데이터를 복구하고 상태를 재설정할 수 있는 정도

일반적으로 신뢰성은 가용성(Availability), MTTF(Mean Time To Failure) 등의 척도로 정량화된다. 가용성은 시스템이 주어진 기간 동안 서비스를 실제로 제공할 수 있는지를 나타내는 속성이다. 예를 들면, 어떤 시스템의 가용성이 0.995를 갖는다고 했을 때 이 시스템은 1000시간 단위(날, 주, 달) 동안에 995시간 단위 동안 서비스를 제공할 수 있음을 의미한다. MTTF는 시스템이 운영된 후 오류가 발생할 때까지 걸리는 평균 동작 시간이다. 예를 들면, 시스템의 MTTF가 100을 갖는다는 사실은 100시간 단위마다 1개의 오류가 발생할 수 있음을 의미한다.

일반적으로 신뢰성을 테스트하기 위해서 통계적 테스트(Statistical testing) 방법을 사용한다. 통계적 테스트는 시스템이 실제로 사용자들이 쓰는 패턴인, 운영 프로파일(Operational profile)을 사용하여 테스트 케이스를 생성한다. 운영 프로파일은 가능한 입력들을 여러 개의 클래스로 분류하고, 분류된 각 클래스의 발생 확률로 구성된다.

통계적 테스트 방법으로 신뢰성을 추정하기 위해서는 일단 운영 프로파일을 작성하고, 각 클래스의 발생 확률에 따라 테스트 케이스를 생성한다. 소프트웨어의 신뢰성을 측정하기 위해서는 오류가 발생하기까지 걸리는 시스템 동작 시간이 중요하므로 오류가 발생한 시간과 오류 발생 후 다음 오류가 발생할 때까지 걸리는 동작 시간을 기록한다. 이렇게 모인 데이터로 여러 신뢰성 추정 모델을 사용하여 신뢰성을 추정하게 된다.

4.7 보안성 테스트

보안성은 시스템이 정보 및 데이터를 보호하는 정도를 뜻하는 용어로, 다음과 같은 다섯 가지 부특성을 갖는다.

- 기밀성(Confidentiality): 접근 권한이 있는 사람에게만 데이터에 액세스할 수 있도록 하는 정도
- 무결성(Integrity): 시스템 또는 구성 요소가 컴퓨터 프로그램 또는 데이터에 무단으로 접근 또는 변경되는 것을 방지하는 정도
- 부인 방지성(Non-repudiation): 사건 및 행위 후에 부인하지 못하도록 행동 및 사건에 관해 입증할 수 있는 정도
- 책임성(Accountability): 각 개인을 유일하게 식별하여 행위를 기록하고 필요시 그 행위자를 추적할 수 있는 능력
- 인증성(Authenticity): 사건 및 행동에 관해 실제 행위자임을 증명할 수 있는 정도

시스템의 보안성을 검증하는 데 주로 사용하는 방법으로 침입 테스트(Penetration test)가 있다. 침입 테스트는 침입자(Hacker)의 관점에서 소프트웨어 시스템의 취약성(Vulnerability)을 찾는 테스트 방법이다. 따라서 침입 테스트는 시스템 안전성에 관해 경험과 지식이 많은 사람이 행하는 것이 보통이다. 이들은 가능한 한 많은 시스템 침입 시나리오들(과거에 사용되었던)을 이용하여 소프트웨어 취약성을 찾고 해결책을 제시한다. 그러나 만약 침입 테스트로 소프트웨어의 취약성이 발견되지 않았다 할지라도 시스템의 보안성이 확보되었다고 확신할 수는 없다. 그 이유는 수많은 침입자가 만들어내는 침입 시나리오를 미리 테스트하기란 거의 불가능하며 미리 테스트할 수 있다 할지라도 침입자들이 또 다른 침입 시나리오를 개발하여 시도할 것이기 때문이다.

보안성 검증을 하는 또 다른 방법으로, 정적 분석을 사용할 수 있다. 이 방법은 보안성 높은 소프트웨어가 준수해야 할 소스 코드 수준에서 코딩 규칙을 정의하고, 이러한 코딩 규칙을 준수하는지를 정적 분석 도구로 검사하는 것이다.

CWE(Common Weakness Enumeration)는 소스코드에 존재할 수 있는 신뢰성 및 보안성과 관련된 코딩 규칙 목록을 정의한 것이다. 그 중 CWE-658은 C 언어로 작성된 소스코드에 관한 규칙 목록으로, Stack overflow, Buffer overflow, Null pointer dereference 등을 포함하며, CWE-659는 C++ 언어에 관한 규칙 목록으로, Uncaught exception, declaration

of catch for generic exception 등을 포함한다. CWE-660은 Java 언어에 관한 규칙 목록으로, Direct use of unsafe JNI, Public clonable() method without final 등을 포함한다. 이 CWE 코딩 규칙은 실제로 방위사업청에서 제시한 '무기체계 소프트웨어 개발 및 관리 매뉴얼'의 무기체계 연구개발사업에 관한 보안성 시험 부분에서 CWE-658, CWE-659, CWE-660 등을 바탕으로 정적 시험을 수행하도록 명시되어 있다.

4.8 유지보수성 테스트

누군가가 시스템을 사용한다면 시스템은 변경될 수밖에 없다. 유지보수성(Maintainability)이란, 시스템이 변경 요구를 만족시키는 능력이며, 유지보수성 테스트란, 이러한 능력을 테스트하는 것이다. 일반적으로 시스템 수정 요구는 다음과 같은 요구에 기인한다.

- **기능 개선 및 추가**: 기능이나 성능을 개선하거나 새로운 기능을 추가하기 위하여 프로그램을 수정하는 작업으로, 시스템 변경 작업 중에서 약 50%를 차지한다.
- **변경된 환경에 적응**: 운영체제나 인프라, 환경 등이 변경되었을 때 이 변화를 수용하도록 프로그램을 수정하는 작업으로, 시스템 변경 작업 중에서 약 25%를 차지한다.
- **오류 수정**: 소프트웨어에 오류가 발견되었을 때 이를 수정하는 작업으로, 시스템 변경 작업 중에서 약 20%를 차지한다.
- **예상치 못한 장애 예방**: 소프트웨어 장애가 발생하기 전에 코드를 재구성(Restructuring)하거나 문서를 갱신하여 장애가 발생하지 않도록 미리 예방하는 작업으로, 시스템 변경 작업 중에서 약 5%를 차지한다.

유지보수성 테스트는 요구사항을 만족하도록 얼마나 쉽게 변경할 수 있는지 테스트하기 때문에 동적 테스트보다는 정적 테스트를 이용하여 테스트한다. 물론, 수정 요구사항이 변경에 소요되는 시간이나 비용으로 표현되면 유지보수 절차를 수행하는데 실제 소요되는 시간이나 비용을 계산하여 요구사항과 비교하는 방식으로 동적 테스트를 해야 할 수도 있다.

ISO 25010에서는 유지보수성을 모듈성, 분석성, 재사용성, 변경 용이성 및 테스트 용이성으로 세분화하였다. 테스트 관점에서 모듈성, 분석성, 재사용성과 변경 용이성은 공통적인 특성이 많이 있으므로 구분하지 않고 한꺼번에 다루어도 상관없지만, 테스트 용이성은 이들과는 특성이 다르므로 분리해서 설명하겠다.

우선, 모듈성, 분석성, 재사용성, 변경 용이성을 테스트하기 위해서는 다음과 같은 특성들을 검토하여야 한다.

- 모듈화 정도
- 모듈 간 결합도
- 모듈 응집도
- 모듈 복잡도

모듈화는 시스템을 여러 부분으로 분할하여 각 부분이 정의된 기능을 잘 수행하도록 하는 설계 기법이며, 모듈화를 측정하는 척도로 fan-in과 fan-out이 있다. fan-in은 얼마나 많은 모듈이 주어진 모듈을 호출하는가를 나타내며 fan-out은 주어진 모듈이 호출하는 모듈 수를 나타낸다.

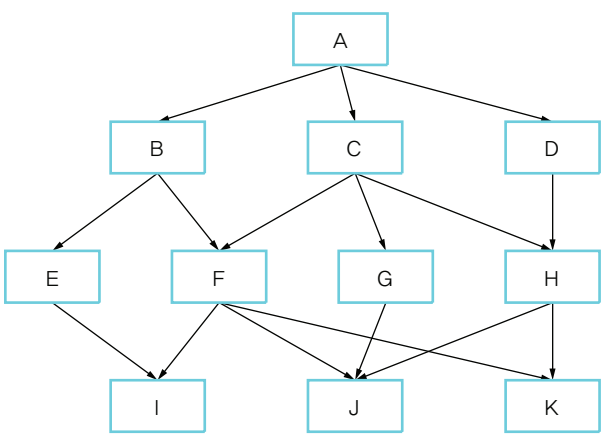

그림 4.4 모듈 구조도

예를 들어 시스템이 그림 4.4와 같이 모듈화되었을 때 모듈 F의 fan-in과 fan-out을 계산해보자. 모듈 F를 호출하는 모듈은 B와 C 2개이므로 fan-in은 2이고 모듈 F가 호출하는 모듈은 I, J, L이므로 fan-out은 3이다. 가장 문제시되는 모듈은 fan-in과 fan-out이 높은 모듈이다. 이 모듈이 변경되면 많은 모듈이 영향을 받게 되며, 또한 해당 모듈이 의존하는 모듈 중 하나라도 변경되면 이 모듈도 영향을 받기 때문이다. 이러한 모듈은 정적 분석에서 식별하고 재구성해야 한다.

모듈 간의 결합도는 낮게 설계되어야 한다. 결합도가 높다는 것은 모듈이 의존하는 모듈들이 변경될 때 영향을 받을 가능성이 크다는 것을 의미한다. 가능한 한 잘 정의된 인터페이스만을 통하여 모듈 간 상호작용을 하는 것이 바람직하며 공통 자료 영역이나 실행 흐름을

변경하는 제어요소를 통해 상호작용하는 것은 피해야 한다.

모듈의 응집도는 개개의 모듈을 구성하는 요소들이 얼마나 서로 관련되어 있는가를 나타낸다. 모듈의 응집도는 높을수록 좋다. 응집도가 높은 모듈은 다른 모듈에 의존 관계가 낮아 재사용성이 좋으며 변경이나 오류를 국부화할 수 있다.

객체지향 프로그램에서 클래스의 응집도를 나타내는 대표적인 척도로 LCOM(Lack of Cohesion in Methods)이 있다. 이 척도는 클래스 내의 메소드들이 얼마나 서로 연관되어 있는지를 나타내며 여러 버전이 있다. LCOM 척도 중의 하나인 LCOM4는 클래스 내의 메소드들이 동일한 필드에 접근하거나 호출 관계가 있으면 연관되어 있다고 간주하며, 모든 메소드가 연관되어 있으면 클래스가 하나의 기능(책임)을 수행하도록 설계되어 있다고 가정한다. 반면에 서로 연관되지 않은 메소드가 2개 이상 있으면 이들을 다른 클래스로 분할할 필요가 있다.

클래스 Foo		
A()	B()	C()
A1()	B1()	C1()
A2()	B2()	C2()
A3()		C3()

그림 4.5 예제 클래스

예를 들어 그림 4.5의 Foo 클래스에서 A() 메소드와 A1(), A2(), A3()가 호출관계로 연결되어 있으며, B() 메소드와 B1(), B2() 메소드가 동일한 필드에 접근하고, C() 메소드와 C1(), C2(), C3()가 호출 관계로 연결되어 있다고 할 때 LCOM(Foo)=3이다. 즉, 3개의 독립된 기능을 하나의 클래스에서 제공한다고 간주하며 이들은 각기 다른 3개의 클래스로 분할할 필요가 있다.

모듈의 복잡도도 유지보수성에 영향을 미친다. 모듈의 복잡도를 측정하는 대표적인 척도로 순환 복잡도가 있다. RIAC(Reliability Information Analysis Center)에서 발표한 복잡도와 소프트웨어 신뢰성과의 관계에서 순환 복잡도가 75 이상이면 결함을 수정하기 위해 수행한 작업이 새로운 결함을 발생시킬 가능성이 매우 큰 것으로 나와 있다. 따라서 정적 분석으로 순환 복잡도가 높은 모듈들을 식별하고 리팩토링 및 재구성을 통해 복잡도를 낮게 만들 필요가 있다.

테스트 용이성은 프로그램을 얼마나 손쉽게 테스트할 수 있는지 나타내는 특성을 말한다. 프로그램의 테스트 용이성을 높이기 위해 표 4.6과 같은 특성을 고려하여 테스트를 수행한다.

표 4.6 테스트 용이성

제어 용이성	프로그램의 실행을 제어하기 용이하도록 설계한다. 제어 용이성이 높아지면 테스트를 자동화할 수 있는 부분이 많아지고 최적화할 수 있다.
관찰 가능성	프로그램 내부 상태가 현재 어떤 상태인지 쉽게 파악할 수 있는 기능을 갖추도록 설계한다.
단순성	가능한 한 시스템 구조 등을 단순하게 설계한다. 코딩 표준에 따라 프로그램을 작성하여 쉽게 코드를 이해할 수 있게 하거나 프로그램 구조를 단순화하여 결함이 발생하였을 때 다른 곳으로 쉽게 전이되지 않도록 설계한다.
분할 용이성	테스트할 대상 영역을 제어함으로써 문제가 발생된 곳을 고립시켜 독립적으로 모듈에 테스트를 수행할 수 있도록 설계한다. 즉, 가능한 한 시스템 모듈 간의 의존성을 줄인다.
운영 용이성	프로그램에 결함이 발생하여도 테스트 작업을 계속할 수 있도록 설계한다.
안정성	테스트하는 동안 소프트웨어에 변경이 자주 발생하지 않도록 설계한다.
이해 용이성	소프트웨어 설계 정보가 잘 조직화되어 쉽게 접근 가능하도록 하여 소프트웨어를 더욱 잘 이해할 수 있도록 설계한다.

4.9 이식성 테스트

이식성(Portability) 테스트의 주요 목적은 서비스 이용자 단말기의 하드웨어 및 소프트웨어 환경이 달라도 동등한 서비스를 제공하는지 테스트하는 것이다. 이식성 테스트는 운영체제(OS)와 브라우저, 태블릿과 스마트폰 등 애플리케이션이 동작하는 운영환경과 사용환경이 다양해지고 복잡해지면서 점점 중요성이 높아지는 추세이다.

예를 들어, 우리나라 전자정부 서비스를 생각해보자. 전자정부 서비스는 다양한 단말기나 브라우저 및 운영체제에서 동작되는 것이 매우 중요하다. 행정안전부에서도 이 점을 고려하여 전자정부 서비스 호환성 준수지침을 제정하였고, 전자정부 서비스의 호환성 확보를 위해 새로 개발될 서비스나 유지 보수되는 서비스에 관해 지켜야 할 사항을 규정하고 있다. 예를 들어, 행정안전부 고시 제2017-26호 개정판 4조에서 웹사이트 호환성 확보에 관하여 다음과 같이 언급하고 있다.

제4조 【웹사이트 호환성 확보】 ① 행정기관 등의 장은 전자정부 서비스를 위한 웹사이트를 신규 개발하는 경우 다양한 웹 브라우저에서 동등하게 서비스를 제공하여야 한다.
② 행정기관 등의 장은 전자정부 서비스를 위한 웹사이트를 개선, 유지보수 및 운영하는

경우 다양한 웹 브라우저에서 동등하게 서비스를 제공하도록 노력하여야 한다.

③ 제1항 및 제2항에서 웹 브라우저의 종류는 해당 전자정부 서비스를 신규 개발, 개선, 유지보수 및 운영하는 행정기관 등의 장이 정한다.

그뿐만 아니라 전자정부 호환성 진단표도 제정하여 구체적으로 진단 지표, 진단 기준 및 방법까지 규정하고 있다. 표 4.7은 전자정부 서비스 웹 호환성 진단표이다.

표 4.7 **전자정부 서비스 웹 호환성 진단표**

구 분	진단지표	진 단 기 준	진단방법
1. 웹표준 문법 준수	1-1 표준 (X)HTML 문법 준수 여부	• W3C Markup Validator에서 출력된 오류 개수에 따라 감점 ※ HTML5의 경우 Nu Html Checker로 자동전환	W3C Markup Validator
	1-2 표준 CSS 문법 준수 여부	• W3C CSS Validator에서 출력된 오류 개수에 따라 감점 ※ CSS의 경우 Level 3으로 진단	W3C CSS Validator
2. 웹 호환성 확보	2-1 기능 호환성 확보 여부	• 브라우저 부가 기능을 이용해서 해당 페이지 내에 사용된 Javascript 오류 및 DOM 경고 발생 시 감점 • Javascript가 의도한 기능이 정상적으로 동작하는지 점검하여 비정상적 동작에 대해 감점	브라우저 부가 기능 크로스 브라우징 테스트 준용
	2-2 화면표시 호환성 확보 여부	• 다양한 웹 브라우저에서 화면표시가 동등하게 구현되었는지 여부 확인 ※ 웹 브라우저별 특성에 따른 차이(폰트, 픽셀 등)는 예외로 함	크로스 브라우징 테스트 준용
3. 비표준 기술제거	3.1 비표준 기술제거 여부	• 웹 사이트에서 비표준기술 (액티브X 등)사용 여부 점검 ※ 개인정보보호, 보안 등을 위해 사용하는 비표준 기술(액티브X, EXE 등)을 사용하고 있는지 점검 ※ 웹 표준이 아닌 방식으로 메뉴, 동영상 등을 위해 사용하는 멀티미디어(플래시, 실버라이트, 자바 애플릿, 미디어플레이어 등)를 사용하고 있는지 점검	수동 평가
	3-2 최신 웹 표준 기술 사용 여부	• 최신 웹 표준 기술(HTML5) 사용 여부 점검 ※ 최신 웹 표준 기술을 활용하여 웹사이트를 운영하고 있지 않을 경우 감점	수동 평가

표 4.7에서 웹 호환성 확보 진단방법을 보면 크로스 브라우징 테스트를 준용한다고 기술되어 있다. 크로스 브라우징 테스트란 다양한 브라우저에서 또는 동일한 브라우저이지만 버전이 서로 다른 브라우저에서 대상 애플리케이션이 동일하게 동작하는지 테스트하는 것을 의미한다. 이는 상이한 운영체제에서 동작하거나 브라우저에서 채택하는 렌더링 엔진이 다르기 때문에 동일한 애플리케이션이 동작하더라도 브라우저마다 다르게 보일 수 있으므로 크로스 브라우징 테스트를 수행하는 작업은 매우 중요하다.

크로스 브라우징 테스트를 자동화하기 위해 Selenium Grid, QTP, RFP와 같은 도구를 사용할 수 있다. 이 도구들은 테스트 스크립트를 다양한 브라우저와 운영체제 조합에서 수행할 수 있다. 예를 들어, Selenium Grid 도구를 사용하여 크로스 브라우징 테스트를 수행하는 과정을 살펴보자. 그림 4.6은 Selenium Grid 아키텍처를 보여준다.

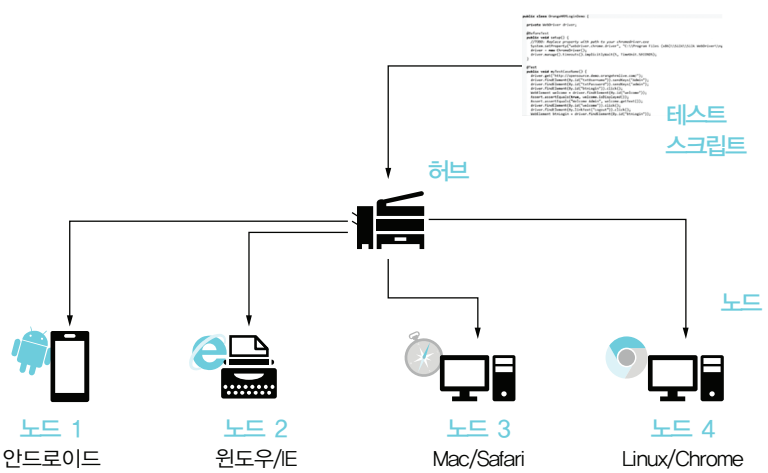

그림 4.6 Selenium Grid 아키텍처

그림에서 볼 수 있듯이 Selenium Grid는 하나의 허브와 여러 개의 노드로 구성되어 있으며, 다양한 운영체제 및 브라우저 환경을 갖춘 여러 노드를 통해 허브에서 제공받은 테스트 스크립트를 동시에 수행할 수 있는 분산 테스트 실행 도구이다. 따라서 크로스 브라우징 테스트를 수행할 수 있고, 만약 노드가 동일한 플랫폼이라면 테스트 스크립트 집합을 여러 노드에 분산 실행함으로써 테스트 실행 시간을 줄일 수 있다.

그러나 이렇게 크로스 브라우징 테스트를 자동화 도구로 수행하면 좋은 점만 있는 것은 아니다. 실제 인간의 눈으로만 확인할 수 있는 부분이 존재하며, 이를 자동화하기란 매우 어렵다. 따라서 전적으로 자동화 도구에만 의존하기보다는 매뉴얼 테스트를 병행적으로 수행하는 것이 좋다. 테스트 자동화는 7장에서 더 자세히 다룬다.