Effective Software Testing: 50 Specific Ways to Improve Your Testing by Elfriede Dustin

Item 47: 테스트 실행 싸이클의 시작과 끝을 명확하게 정의하라.

테스트 단계(phase)에 상관없이, 소프트웨어 테스트 실행 싸이클에 대한 진입 기준(테스팅이 시작될 때)과 종료 기준(테스팅이 완료될 때)를 정의하는 것이 매우 중요하다. (* 역자주: 진입 기준 - entrance criteria, 종료 기준 - exit criteria)

진입 기준은 테스팅 팀이 특정한 빌드의 테스팅을 시작하는 시점을 묘사한다. 시스템 테스팅을 진행하기 위해 소프트웨어 빌드는 다양한 기준들을 만족해야 한다. 예를 들어,

■ 모든 유닛, 통합 테스트가 성공적으로 실행되었다.
■ 소프트웨어가 아무런 이슈없이 빌드(컴파일) 되었다.
■ 빌드는 스모크 테스트 (* 역자주: BVT - Build Verification Test)를 통과했다. (Item 40 에서 논의한 것처럼)
■ 빌드는 무엇이 변경되었는지, 무엇이 새로 추가되었는지를 기술한 문서 (릴리즈 노트)를 포함하고 있다.
■ (이전에 보고된) 결점들이 수정되었고, 재테스트(retesting)를 수행할 준비가 되었다. (결점 추적 라이프싸이클에 대한 논의를 담고 있는 Item 49 를 보라.)
■ 소스 코드는 버전 컨트롤 시스템에 저장되어 있다.

진입 (또는 승인) 기준이 만족될 때만 테스팅 팀은 소프트웨어 빌드를 받아들여 테스팅 싸이클을 시작할 수 있다. 테스팅 단계에 대한 진입 기준을 정의하는 것이 중요한 것처럼, 종료 기준 역시 개발되어야 한다. 종료 기준은 소프트웨어가 적절하게 테스트된 때를 기술한다. 진입 기준처럼, 종료 기준 역시 테스팅 단계에 의존적이다. 테스팅 리소스가 유한하며, 테스트 예산과 테스트 엔지니어의 수에 제한이 있으며, 마감일은 빨리 다가오기 때문에, 테스트 활동의 범위는 제한을 가져야 한다. 테스트 계획은 테스팅이 완료되었을 때를 명확하게 지시해야 한다. 만일 종료 기준이 모호한 용어로 정의되어 있다면, 테스트 팀은 테스트 활동이 완료되는 시점을 결정할 수 없을 것이다.

예를 들어, 테스트 완료 기준은 요구 사항에 근거해 모든 테스트 프로시저를 정의할 것이다. 이것은 심각한 문제 없이 성공적으로 테스트가 완료되며, 개발팀에 의해 높은 중요도를 가지는 결점이 모두 수정되며, 테스트 팀에 의한 회귀 테스팅에 의해 확인되는 것을 의미한다. 이 책 전체에서 논의된 다른 활동들의 맥락에서 이러한 기준을 만족하는 것은 중대한 결함 없이 시스템이 모든 요구 사항들을 만족시킨다는 높은 수준의 확신을 제공한다.

아래의 예들은 어플리케이션의 종료 기준의 일부분으로 고려될 수 있다.

■ 테스트 프로시저는 시스템이 사전 정의된 기능적, 비기능적 요구사항을 만족하는지 결정하기 위해 실행되었다.
■ 레벨 1, 2, 3 (showstopper, urgent, and high-priority)의 소프트웨어 문제가 테스팅을 수행한 결과로 문서화 되었다.
■ 보고된 레벨 1, 2 (showstopper, urgent)의 소프트웨어 문제가 해결되었다.
■ 레벨 1, 2 (showstopper, urgent)의 소프트웨어 문제가 테스팅을 수행한 결과로 문서화 되었고, 보고된 레벨 3 의 문제중 90%가 해결되었다.
■ 소프트웨어는 낮은 중요도의 결점만을 발견한 채로 출시될 수 있다. (아마도 발견되지 못한 결점도 포함되었을 것이다.)

종료 기준을 만족했음에도 불구하고, 소프트웨어는 고객에게 유용할 때만 성공적일 것이다. 따라서, 유저 승인 테스팅 (User Acceptance Testing)을 테스팅 계획의 요소로 고려하는 것이 중요하다.

개발자들도 시스템 승인 기준에 대해서 인식하고 있어야 한다. 테스트 팀은 승인을 위해서 테스트 계획을 제출하기 전, 초기에 개발팀과 진입과 종료 기준의 리스트에 대해서 의사소통 해야 한다. 조직 내의 진입과 종료 기준은 몇몇 프로젝트에서 증명된 기준들을 기반으로 해서 표준화되어야 한다. (Entrance and exit testing criteria for the organization should be standardized where possible, and based upon criteria that have been proven in several projects.)

이후에 릴리즈나 패치에서 다룰 수 있는 몇몇 결점을 포함한 채로 출시해도 될 것인지를 결정해야 할 것이다. 생산 과정을 거치기 전에, 테스트 결과는 어떤 결점은 즉시 수정해야 하는지 어떤 것은 수정을 연기해도 되는지 확인하는데 도움을 얻기 위해 분석할 수 있다. 예를 들어, 어떤 "결점"들은 기능 개선으로 재분류하여, 이후의 릴리즈에서 처리되게 할 수도 있다. 프로젝트/소프트웨어 개발 매니저는 변경 제어 위원회의 다른 멤버들과 함께, 결점을 지닌 채로 출시할 것인지, 즉시 수정을 할 것인지를 결정한다.

종료 기준의 일부로서 평가되어야 하는 부가적인 메트릭의 예는 다음과 같다.

■ 이전에 작동하던 기능에 대해서 회귀테스트에서 발견된 결점의 비율은 어떠한가? 다시 말하면, 이전에 작동하던 기능의 오작동을 얼마나 자주 수정하는가?
■ 결점 수정의 실패가 얼마나 잦은가? 즉, 수정되었을 것으로 기대한 결점이 실제 그렇지 않은 것을 의미한다.
■ 이 단계의 테스팅이 진행되면서 새로 발견되는 결점 비율의 추이가 어떠한가? 테스팅이 진행되면서 새로 오픈되는 결점은 감소해야 한다.

테스팅은 어플리케이션이 출시하기에 적합한 상태일 때, 종료 기준에 부합할 때 종료를 고려할 수 있다. 심지어 아직 발견되지 않은 결점들이 잔존할 것으로 생각되어도 말이다.

제한된 예산과 스케줄의 환경에서는 테스팅을 중단하고, 제품이 배포되어야 하는 시점이 존재한다. 아마도 소프트웨어 테스팅에 있어서는 언제 테스팅을 중단할 것인가가 가장 어려운 결정일 것이다. 소프트웨어 릴리즈와 완결을 위한 품질 가이드라인을 수립하면 테스트 팀은 그러한 결정을 내릴 수 있을 것이다.

Item 48: 개발 환경과 테스트 환경의 분리

테스트 환경은 테스팅 팀이 테스트 전략을 수행할 준비가 되는 시점에 셋업되는 것이 중요하다.

테스트 환경은 개발 환경과는 분리되어야 하는데, 이는 테스팅 하는 중에 소프트웨어가 추적되지 않게 변경되거나 과도한 실수를 피하기 위해서이다. 하지만 아주 자주, 이것은 정례화되지 않으며, 비용을 줄이기 위해서 테스팅 팀이 별도의 분리된 테스트 환경을 얻지 못한다.

별도의 독립된 테스트 환경이 없다면, 테스팅 활동은 아래와 같은 문제에 직면하게 된다.

■ 환경의 변경. 개발자는 테스팅 팀에 통보없이 수정 사항을 적용할 수 있거나, 개발 환경 설정에 또 다른 변경을 가하고, 새로운 데이터를 추가할 수 있다. 만일 테스터가 그 환경에만 근거해서 결점을 보고 했다면, 그 결점은 이제 재현 가능하지 않게 될 것이다. (If a tester has just written up a defect based on the environment, that defect may now not be reproducible.) 그 결점 리포트는 아마도 "종료 - 재현되지 않음"으로 마킹될 것이다. 결과적으로 개발과 테스팅 리소스가 낭비되는 것이다. 추가적으로, 개발자가 작업한 내용인 기존 코드의 변경 사항에 대해서 테스터에게 알리지 않는다면, 새로운 결점은 놓치게 될 것이다. (* 역자주: 새 기능에 대한 테스팅을 하지 않았으므로) 테스터는 변경된 코드에 대한 재테스트도 모르게 될 것이다.

■ 버전 관리. 개발자와 테스터가 동일한 환경을 공유하는 경우 버전을 관리하기가 어렵다. 개발자들은 소프트웨어에 새로운 기능을 통합하려고 할 것이고, 반면에 테스터들은 테스팅을 완료하기 위해서 안정 버전(stable version)을 필요로 할 것이다. 반대로, 개발자들은 자신들의 데스크 탑 머신 이외의 또 다른 최신 빌드의 소프트웨어를 실행시킬 곳을 필요로 한다. 만일 테스터들이 개발 환경을 공유한다면, 개발자에 의해서 그 환경의 사용에 제약이 있을 것이다.

■ 운영 환경의 변경. 테스팅은 다양한 조건 하에서 어플리케이션을 테스팅하기 위해서 환경의 재설정(reconfiguration)을 필요로 한다. 이것은 개발 활동과 충돌할 수 있는 대규모 환경의 혼란을 유발한다. 예를 들어, 머신이 재부팅되거나, 며칠동안 오프 라인 상태로 만들어 설치된 소프트웨어나 하드웨어에 변경을 일으키는 것이 그것이다.

예산 상의 제약은 종종 독립된 테스팅 환경을 구성하지 못하게 하지만, 몇가지 비용 절감의 방법을 사용하면, 독립된 테스트 환경을 적절히 만들 수 있다. 예를 들면,

■ 성능과 용량을 적게한다. 테스팅 환경으로 대용량의 디스크나 메모리 용량을 가진 고성능의 머신을 필요로 하진 않는다. 더 작은 머신들은 대 부분의 테스팅 활동을 가능하게 하며, 그 결과를 가지고 더 큰 하드웨어 플랫폼에서의 성능을 추정할 필요가 있을 때 대응할 수 있다. 이것은 예산이 제품 사이즈의 테스트 머신을 허락하지 않는 경우에만 권장된다. 하지만, 이러한 추정은 성능을 측정하는 부정확한 방법이므로, 실제 운용 환경에서의 실제 성능을 단순 추정만하게 된다. 실제 수치가 필요한 경우, 성능 지표는 실제 제품 클래스의 하드웨어에서 도출되어야 한다.

■ 이동식 디스크. 다수의 환경에 대한 테스팅에 반드시 다수의 머신이나 각각의 환경에 대한 완전한 재인스톨을 필요로 하는 것은 아니다. 다중 파티션을 가진 이동식 디스크로 비용과 시간을 절약하면서, 다수의 소프트웨어 환경을 꾸밀 수 있는 단일 머신을 운용할 수 있다. 호환성 테스팅을 위한 이동식 디스크의 사용에 대해서는 Item 46 을 더 참고하라.

■ 공유 테스트 랩. 다수의 소프트웨어 프로젝트에 대해서 테스트 랩을 사용함으로써 테스팅 환경을 꾸미는 비용을 공유할 수 있다. 이러한 랩은 단일 프로젝트를 위해서만이 아닌, 다양한 테스팅 요구를 위해 쉽게 재설정이 가능하도록 디자인되어야 한다.

{중략}