품질 보증: 다각적 게임 테스팅을 위한 방법론

(Quality Quality Assurance: A Methodology for Wide-Spectrum Game Testing)

데이빗 윌슨(David Wilson)
가마수트라 등록일(2009.04.28)

http://www.gamasutra.com/view/feature/4007/quality_quality_assurance_a_.php

[닌텐도와 마이크로소프트 게임 스튜디오에서 오랜 경력을 쌓은 윌슨은 본지 기사에서 다양한 게임 테스팅의 가치에 대해 언급하고, 버그가 최소화된 상태로 게임을 출시할 수 있는 방식을 제시할 것이다.]

다각적 테스팅(wide-spectrum testing)의 필요성

응용프로그램 개발자이건, 게임 개발자이건 소프트웨어를 개발하다 보면 피해야 하는 함정과 위험 요소들을 계속 직면하게 된다. 개발 주기상 가장 자원을 많이 소모하는 단계 중 하나인 소프트웨어 테스팅 작업은 이러한 문제를 개선하고, 소프트웨어에 실수로 오점을 남기는 것을 막아준다.

일부 개발자들은 소프트웨어 테스팅을 눈에 띄는 오류를 찾아내는 과정이며, 필요악이고 부차적인 것으로 여긴다. 그러나 소비자들의 생각은 소프트웨어 테스팅을 통해 제품에 있는 문제점이 반드시 발견되어야 한다고 것이다. 테스터들은 테스팅 과정을 성급하게 대충 처리해도 된다고 생각하는 경향이 있다.

특정한 테스팅 방식이 다른 것보다 우월하다고 생각하는 것이야말로 소프트웨어 테스팅에 대한 가장 큰 착각이다. 소프트웨어 테스팅은 예술과 과학의 결합체이며, 둘 중 어느 하나도 무시되어서는 안 된다.

아무리 과정이 철저하더라도 엄격한 조건에 따른 테스트 또는 무작위적인 테스트 둘 중 어느 하나만으로는 충분치 않다. 예술과 과학이 둘 다 겸비되어야 가급적 많은 오류를 잡아내어 소프트웨어의 기능을 최대한 높이고 빛을 발하게 할 수 있다.

임의 테스팅(Ad-hoc testing)와 사례 테스트(test cases) 간의 차이점

임의 테스팅

자유 테스팅이라고도 알려진 임의 테스팅은 예술적 측면에 중점을 둔 테스팅이다. 이러한 형식의 테스팅은 게임 테스팅에서 가장 많이 사용되고 있다. 이는 소비자 사용 패널 및 포커스 그룹에서도 나타나고 있는데, 이는 개인들을 불러모아 도달해야 할 매우 광범위한 목표에 새로운 소프트웨어가 부합하는지를 살펴본다.

이 테스팅은 스타일 상 매우 유동적이며 무작위적으로 보이기도 한다. 게임 테스터는 레벨의 중간쯤 진행하다가 개발자가 환경의 틈새로 통과하기를 바라는 곳을 따라 가면, 캐릭터는 환경의 경계선을 떠났다가 꼼짝도 못하고 정상적인 플레이 흐름 속에 돌아올 수 없게 된다.

프레젠테이션 소프트웨어를 테스팅할 때 테스터는 이미지나 동영상을 로드하는 데 시간을 주지 않고 전체 프레젠테이션이 한 번에 나오게 하도록 시도하게 된다. 그러다가 사용 가능한 메모리에 과부하를 주어 소프트웨어가 반응을 못 하거나 서 버리는 문제점이 나올 수도 있다.

장점

테스팅 과정 중에 무작위적으로 이런저런 일을 하는 것처럼 보이지만, 이는 현실 세계에서 얼마든지 있을 수 있는 일들이다. 이 때문에 특정적 테스트를 예술이라고 부르는 것이다. 최종사용자들의 시도 중 개발자가 미처 예상치 못한 부분을 잡아낼 수 있기 때문이다.

이는 매우 간단한 것 같지만 이런 류의 테스팅에는 엄청난 창의력이 필요하다. 이 때문에 테스팅 계획이 지연될 수도 있다.

게임 속에서 던전에 들어갔다가 던전 중간쯤에 눌러앉았다가 다시 마을로 돌아와서 게임을 저장하면 게임이 즉시 서 버릴 수도 있다. 또한 플레이어가 던전 안에 들어가기 전에 게임을 저장해놓지 않았다면 이런 일이 끝없이 벌어질 수도 있다.

이런 상황을 테스트하는 데 필요한 계획을 확실히 만들 수도 있다. 그러나 모든 상황에 다 대처할 수 있는 계획을 짜는 것은 불가능하다. 이 경우 임의 테스팅은 가장 유용하다. 개발 중에 미처 예기치 못한 상황이 출시 후 발생할 수도 있는데, 그런 상황을 미리 테스팅 해본다는 것이다. 훌륭한 테스터는 깜짝 놀랄 정도로 예기치 못한 행동을 많이 한다. 또한 그들은 그만큼 많은 문제를 출시 전에 잡아내어 고칠 수 있다.

단점

이러한 기법은 꽤 근사하게 들리며, 아무도 주의하지 못했던 큰 문제를 잘 잡아낸다. 그러나 여기에도 몇 가지 문제점은 있다. 소프트웨어의 모든 기능을 이런 식으로 테스트하기는 거의 불가능하다. 테스팅 팀이 아무 지향점도 없이 자유 테스트를 하려면 많은 공간이 필요하다.

또한 테스팅 영역 중 대부분의 사람들이 좀 더 신경을 쓰는 영역이 있고, 덜 신경을 쓰는 영역이 있다. 이러한 문제를 해결하기 위해 임의 테스팅 선호자들은 소프트웨어의 특정 영역에 일부러 임의 테스터들을 배정하는 식으로 테스팅 계획을 조절한다. 그러나 이것도 테스팅 방식이 엄격하지 못한 데서 오는 단점을 완전히 해소할 수는 없다.

사례 테스트

이것은 과학적인 테스팅 기법이다. 개발자들과 테스팅 팀 감독은 제품의 기능에 따라 테스트해야 할 항목을 정한다. 예를 들면 어떤 기능이 다른 기능과 상호작용을 하는가? 각 기능마다 매개변수는 어떻게 다른가? 하는 것들이다.

사례 테스트는 임의 테스팅의 정반대 개념이다. 임의 테스팅이 무작위적이라면 사례 테스트는 더욱 엄격하고 통제된 조건이다. 여기서 주로 문제 삼는 것은 기능이다. 여기에는 스프레드 시트 위의 셀 사이를 이동하는 것 같은 간단한 기능, 또는 적의 무리에게 복잡한 주문을 외우는 등의 복잡한 기능까지 다 포함되며 테스트 항목 작성자가 생각해낼 수 있는 모든 관점과 명령 스타일에 따라 테스트가 행해진다.

장점

사례 테스트는 임의 테스팅의 단점을 보완해 줄 수 있다. 소프트웨어의 전 영역에 걸쳐 다양한 방식으로 가장 일반적인 행동을 취했을 때를 테스트해 볼 수 있는 것이다.

이는 테스팅 과정에서 얻을 수 있는 큰 이점이다. 임시 테스터가 테스팅하고 있는 소프트웨어에서 얻게 될 칼라 팔레트가 일정 포맷 스타일 내에서 부정확한 변수 요청에 응하면서 원하는 파란색을 녹색으로 바꿔버릴 수 있는데, 사례 테스트는 각 포맷 마다 색깔을 정확히 점검하면서 이런 문제를 쉽게 잡아낼 수 있다.

단점

한 타이틀에 대한 사례 테스트 범위는 사례를 작성하는 사람에 따라 달라질 수 있다. 다년간의 경험과 테스트 기능에 대한 심도있는 지식이 있는 사람이 작성하면 철저한 테스트가 이루어지겠지만, 그렇다고 해도 최종 사용자가 어떤 시도를 할 지 누구도 예측할 수 없다.

가능한 모든 경우를 다 포괄하려면 계산에 넣어야 할 무작위변수가 너무 많아진다. 또한 일부 테스터들은 테스팅 절차가 너무 까다로울 시 쉽게 지친다는 점도 무시 못한다. 이 때문에 드물게나마 사례 테스트가 제대로 완료되지 못하는 경우도 있다.

사용 도구

화이트박스 테스팅 vs. 블랙박스 테스팅

테스팅 연결망의 두 번째 축을 이루는 것이 바로 이 두 가지 방법이다. 테스터들에게 주어지는 소프트웨어 작동 정보의 양은 어떤 테스팅 방법을 사용하느냐에 따라 정한다.

블랙박스 테스팅에서는 소프트웨어 내부 작동에 대한 정보가 테스터들에게 거의 주어지지 않는다. 테스터는 최종사용자와 완전히 동일한 시각에서 소프트웨어를 보게 된다. 게임의 베타 버전을 테스트용 콘솔에서 플레이하는 것이 그 좋은 사례이다.

반면 화이트박스 테스팅은 소프트웨어의 작동을 알 수 있는 내부 기능이나 두 번째 소프트웨어 스위트(보통 디버거)를 사용하여 정보를 테스터 또는 테스트 로그에 필요할 때마다 보낸다.

이러한 테스팅의 좋은 사례는 프로그래밍 플랫폼의 개발 환경 내의 소프트웨어 빌드를 테스팅하는 것 또는 자동화 소프트웨어를 사용하여 특정 동작을 계속 반복했을 때 나타나는 미묘한 차이를 점검하면서 디버거를 배경에서 작동시키는 등이다.

이렇게 보면 화이트박스 테스팅이 훨씬 더 좋을 것 같지만, 배경에 또 다른 소프트웨어가 동작해야 하며 소프트웨어 내를 흐르는 데이터를 가로채 테스팅 해야 한다는 것은 중요하게 생각해 볼 문제이다.

이러한 방식은 소프트웨어의 정상적인 작동을 방해하여, 원래는 일어날 일이 없는 문제를 발생시키거나, 일어날 수 있는 문제 발생을 막기도 한다. 그러므로 화이트박스와 블랙박스 테스팅 간에 균형을 이루어야 소프트웨어에 대해 적절한 테스트가 이루어질 수 있다.

테스터

이들의 주특기는 소프트웨어 분해하기이다. 선호하는 방식은 소프트웨어 분해이며, 이들 중 다양한 테스트 방법에 정통한 사람이 많다. 이들이 접할 수 있는 자원이 많고, 개발자와 높은 유대관계를 유지해야 적절한 테스팅을 수행할 수 있다.

타사 테스터

흔히 말하는 가장 낮은 단계의 테스터이다. 이들은 테스팅 과정의 주축을 형성하기도 한다. 타사 테스터는 프로젝트별로 계약을 맺는 경우가 많다. 특정한 방식에 특별한 재능을 보이는 사람이라면 특별한 테스트 장비를 사용할 수 있는 경우도 있지만, 보통 그렇지 못한 경우가 많다. 타사 테스터는 주로 블랙박스 테스팅에 많이 동원된다.

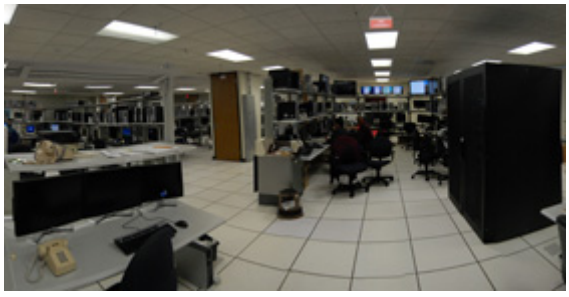
대규모로 계약직 테스터를 고용하면 옥석을 구분할 수 없는 경우가 많아진다. 그러나 기술적 지식이 거의 없는 테스터도 적절한 교육을 받아 좋은 결과를 얻어내면 임의 테스팅은 물론 사례 테스트 모두에 투입할 수 있다.

자회사 테스터

대기업의 자회사 또는 자매 회사의 테스팅 그룹에서 일하는 테스터이다. 이들은 계약제 또는 전일제로 근무할 수 있다. 개발자들과 관계가 비교적 긴밀하므로 더욱 진보된 도구를 사용하는 경우가 많다. 이 때문에 사례 테스트와 화이트박스 테스팅에 더 큰 중점을 두는 경우가 많다. 대부분의 자회사 테스터는 테스팅 과정에 대한 어느 정도의 경험이 있다.

자사 테스터

개발자와 직접 함께 일하거나 직접 의사소통을 하는 이들은 개발사 소속의 전일제 근로자들인 경우가 많지만, 숙련된 계약직 직원들도 이들의 임무를 맡는 경우가 있다. 이들은 테스팅 도구를 가장 많이 사용할 수 있으며 업무에 임하는 다른 테스터 그룹을 감독하기도 한다. 대부분의 자사 테스터들은 테스팅 과정과 다양한 개발주기를 잘 알고 있다.



테스터와 개발자와의 역학관계

앞서 말한 유대관계 수준이 중요한 이유는 바로 이 역학관계 때문이다. 개발자와 테스팅 그룹 간에 다양한 문제에 대한 의견충돌을 일으켜 분열을 일으키는 경우는 너무나도 흔하다.

두 그룹 사이가 소원해질수록 이러한 문제가 발생할 확률은 더욱 높아진다.

타사 테스터의 경우 도구나 자원이 없다는 불평을 가장 많이 할 것이다. 또한 어떤 테스터이건 버그 처리에 대해 이견이 발생할 수 있다. 버그를 인정하거나 또는 없앨 때는

비용 대 효율성, 시간제한, 타당성 등을 모두 고려해야 한다. 모든 직원들은 자신들이 동일한 목표, 즉 멋지고 잘 작동하는 제품을 만들기 위해 일하고 있다는 사실을 명심해야 한다.

이 때문에 개발자들은 많은 테스터들이 최종사용자 중 핵심집단을 이룬다는 사실을 알고, 그들의 의견에 귀를 기울여야 한다.

이와 마찬가지로 테스터들도 개발자들이 프로젝트에 대해 테스터들이 모르는 많은 정보를 알고 있으며 그 정보에 기반해 결정을 내린다는 사실을 알아야 한다. 이러한 상황은 두 그룹 간 정보 및 유용한 도구 공유로 해소될 수 있다.

관리팀

업무 분산의 필요성

개인이나 부서에 업무를 할당할 때 그들이 그 일을 해낼 수 있는지 미리 따져보는 것이 중요하다.

그 이유는 두 가지이다. 우선 그렇게 해야만 업무를 할당받은 개인이나 부서가 자신들이 신뢰받고 있음을 알게 되면서 사기가 오를 것이고, 두 번째로 노력의 중복 투입을 막을 수 있기 때문이다.

테스트를 지켜보는 눈이 많을수록 좋다. 그러나 먼저 끝내야 하는 테스트를 위해 기다려주는 것도 필요하다. 업무 분담의 목표는 팀 전체가 힘을 합쳐 테스트를 완벽히 끝내기 위해서이다.

이 때문에 업무는 가장 잘 할 수 있는 부서에 적절히 분담되어야 한다. 예를 들어 게임플레이 테스팅, 텍스트 체크, 인증 테스팅을 해야 하는 게임의 경우 이 세 가지 업무를 가장 잘 할 수 있는 팀에 분산해서 맡겨야 한다.

한 팀이 주어진 업무를 종료하면 이미 완료된 다른 팀의 업무를 중복 체크해야 한다. 특정한 자격을 가진 인원에 의해서만 수행되는 인증 테스팅의 경우는 예외로 하자. 이는 인증 팀의 다른 멤버만이 실시해야 한다. 이는 단 두 사람만 있어도 소프트웨어의 각 부분을 모두 교차 검증할 수 있는 최상의 방식이다.

팀의 위계구조와 의사소통

어떤 프로젝트이건 여러 팀의 활동을 조정하는 사람이 하나 있어야 한다. 또한 자신이 속한 팀의 활동을 조정하고 프로젝트 리더에게 상황을 보고하는 사람이 팀마다 한 명씩은 있어야 한다. 이는 여러 팀 간에 문제가 발생했을 때 신속히 따를 수 있는 명령 체계를 구성한다.

이는 한 팀이 다른 팀에게서 도움을 받아야 할 때 중요하다. 이런 명령 체계가 있으면 한 팀의 리더가 프로젝트 리더 및 다른 팀의 리더와 동시에 의사소통이 가능하기 때문이다.

팀이 그리 바쁘지 않다면 별 문제가 안 될 수도 있다. 그리고 각 팀의 리더가 쉽게 도움을 주고받을 수도 있다. 그러나 개발주기의 막바지에는 업무가 폭주하므로 시간이 없어 쉽게 남을 도와주기 힘들 수 있다.

이런 상황에서 과연 한 팀이 다른 팀을 도와야 할지 말아야 할지를 정하는 것은 프로젝트 리더의 몫이다.

이는 팀간 의사소통이 잘 되어야 하는 이유이기도 하다. 목표 시간보다 빨리 일을 진행하는 팀이 있다면, 의사소통이 잘 되어야 그 팀이 업무진척이 부진한 팀을 도울 수 있다. 또한 팀 전체가 자신들의 전반적 진행상황을 알고 팀 리더와 프로젝트 리더가 팀의 노동력을 전반적으로 더 잘 관리하는 데 도움이 된다.

주간보고 또는 일간 보고는 이러한 의사소통에 매우 큰 도움이 된다. 또는 리더 간의 정기적인 구두대화만으로도 큰 도움이 된다. 약간의 의사소통으로도 오랫동안 일을 제대로 무리 없이 처리할 수 있다.

적절한 테스터를 적절한 팀에 배치하라

특정 능력에서 남들을 능가하는 실력을 가진 사람들이 있다. 따라서 능력과 직무를 잘 연결시켜주는 것이 필요하다. 직원의 실력을 알아내는 데는 시간이 걸리지만, 그 만큼의 가치가 있다.

게임 클리어에 뛰어난 인원은 게임 플레이, 특히 게임 클리어가 필요한 쪽에 배치해야 한다. 이 경우 이 인원의 능력은 게임 엔딩은 물론 전반적인 플레이 점검과 연관된 업무를 보는 데 도움이 될 것이다.

이와 마찬가지로 언어에 뛰어난 직원은 게임 내 텍스트 체크에 배속시켜 게임에 텍스트가 제대로 나오는지를 살피게 해야 한다. 그래픽 문제를 잘 파악하는 인원은 그래픽과 애니메이션을 체크하게 해야 한다. 이렇게 하면 팀은 자신들의 실력을 최대한으로 발휘하여 주어진 업무를 가장 효율적으로 해낼 수 있다.

적절한 조화

이제 일반적 개념은 다 나왔는데, 이 모든 것을 어떻게 하나로 합칠 것인가? 최초의 목표는 테스팅 기술을 적절히 배분하는 것이었다.

처음에는 사례 테스트를 40%, 임의 테스팅을 30%, 나머지 30%는 테스터 실력에 따라 유동적으로 정하는 것이 좋다.

마지막 30%를 가장 쉽게 정하려면 여기에 우선도가 낮은 사례 테스트를 배정한다. 그래야 조직에 이런 임무에 어울리는 사람이 적더라도 피해가 적다. 이러한 각 그룹의 절반에는 화이트박스 테스팅을, 절반에는 블랙박스 테스팅을 배정해야 한다.

처음 1~2주 동안에 어느 테스터가 어떤 임무에 가장 적합한지를 판단해야 한다. 이것이 일단 정해지면 테스터들을 적합한 팀으로 배치한다.

이 시점에서 테스팅은 사례 테스트 60%, 임의 테스팅 40% 정도로 비율을 조정한다. 사례 테스팅에 시간과 인력이 더 들기 때문이다. 팀이 구성되면 각 팀 별로 테스트 계획을 작성하여 업무수행방식과 속도를 알려야 한다.

처음 1~2주는 신인 테스터의 실력을 알아보는 중요한 시기이다. 테스터가 특정 임무에 적합하다는 것을 알면 그를 적절한 팀으로 보내 일을 시작하게 할 수 있다.

즉 최종적으로 배치하기 이전에 테스터의 재능을 알아내야 팀이 오래도록 무사히 테스팅을 하는데 지장이 없다는 뜻이다.

마지막으로 특히 개발자들에게 몇 마디 조언하자면, 다양한 메카니즘과 기능에 대해 자료화를 해두면 언제 어디서건 유용하게 쓸 수 있다. 테스팅 팀에게 많은 자료를 줄수록 테스트가 더욱 잘 이루어진다.

또한 게임 내에서 음성화되는 텍스트가 있을 경우 우선 편집 과정을 거쳐라. 이래야 오디오에서 생길 수 있는 오류를 줄일 수 있다. 또한 그런 오류에 텍스트를 맞출 수는 없기 때문이다.

마지막으로 언제라도 테스터들의 의견을 물어라. 이들의 의견을 이용하면 개발자들이 생각지 못한 방향으로 소프트웨어를 개선할 수 있다. 우리 모두는 프로젝트의 성공을 원한다. 약간의 팀워크만 있다면 가급적 오류가 없고 멋진 제품을 출시할 수 있을 것이다.