소규모 개발자가 대규모 게임을 제작하는데 따른 리스크를 최소화하는 법 – 2부

(Small Developers: Minimizing Risks in Large Productions – Part II)

트로이 더니웨이(Troy Dunniway)

가마수트라 등록일(2009. 11. 19)

http://www.gamasutra.com/view/feature/4180/small_developers_minimizing_risks_.php

[이 기사에서는 마이크로소프트, EA, 인섬니악(Insomniac), 유비소프트(Ubisoft)와 같은 스튜디오에서 경력을 쌓아온 숙련된 게임 개발자이자 매니저인 트로이 더니웨이씨가 소규모 개발팀에서 대규모 개발팀으로 이행하는데 따른 주요 리스크에 대해 서술하고 리스크를 알아내는 방법과 없앨 수 있는 방법에 대해 서술한다. 이번 두 번째 글에서는 서로 다른 영역에서 나타나는 핵심적인 리스크를 다룬다.]

이전에 작성한 가마수트라 기사인 소규모 개발자가 대규모게임을 제작하는데 따른 리스크를 최소화하는 법-1 부 에서는 소규모 회사가 규모를 확장하기 위해 보다 큰 프로젝트를 맡게 될 경우 맞닥뜨리게 되는 일반적인 리스크에 대해 설명했다. 이 기사에서는 게임 개발 비즈니스에서 나타나는 리스크, 적절한 제작 전 과정을 거치지 않을 경우의 리스크, 일하는 방법을 정의한 로드맵이 없을 경우의 게임 개발 프로세스 리스크, 팀 구성원과 직원을 둘러싼 리스크 등에 대해 개괄적으로 살펴보았다.

2 부에서는 계속해서 스케줄링을 제대로 하지 못하는 경우의 스케줄 리스크 및 프로젝트 관리 리스크, 게임 설계, 게임 예술, 테스팅, 프로그래밍 결정이 잘못되었을 경우와 연관된 리스크, 아웃소싱과 연관된 리스크 등에 대해 알아본다. 이 기사는 여러분이 조심해야 할 프로젝트에서의 세세한 문제에 대해 초점을 맞추고자 한다.

1. 스케줄 리스크 및 프로젝트 관리 리스크

프로젝트 스케줄링이 부적절하게 이루어진 경우의 리스크는 다양하게 나타나며 도처에 산재해 있다. 스케줄을 짜는 방법이나 올바른 개발 방법에 대한 정답이나 오답은 없다. 일부 팀에서는 폭포수 방법(waterfall methodology)을 활용하기도 하고 일부에서는 애자일 방법(agile method)을 사용하기도 한다. 대부분은 혼합된 방식을 이용한다.

어떤 방법을 사용하든 프로젝트 스케줄링 작업을 하지 않는다면 감당하기 힘들만큼 많은 리스크가 일어날 것이며 스케줄링이 적절하게 이루어지지 않거나 뒤쳐지거나 스케줄링에서 잘못된 결정을 내리게 될 경우에도 수많은 리스크를 초래하게 될 것이다.

많은 프로듀서들은 예측하지 못한 문제에 대처하기 위해 모든 태스크와 스케줄에 20%에서 50%(또는 그 이상)의 여유분 정도를 추가하기도 한다. 이 방법이 좋긴 하지만 모든 문제를 해명하진 못한다.

게임을 적절하게 스케줄링 하는 방법에 대한 해법과 리스크에 대해 설명하려면 책 1 권을 써야 할 것이다. 명심해야 하는 것은 프로젝트 스케줄링을 하지 않거나 제대로 하지 않는다면 많은 문제가 야기될 것이라는 것이다. 그렇기 때문에 규모가 큰 팀에서는 보통 프로젝트, 나아가 팀이나 기능의 스케줄까지 관리하는 개발 디렉터를 고용하는 경우가 많다.

스케줄에서 나타나는 리스크를 없앨 수 있는 최선의 방법은 책임자가 맡는 프로세스를 100%가 아닌 50%에서 75%정도로 하면 이들이 여유 있게 스케줄링, 고용, 팀 관리, 상태 보고서 처리, 1:1 미팅 주재와 같은 일을 처리할 수 있다. 많은 팀들에서는 자신의 리더가 다른 구성원과 마찬가지로 프로젝트에 생산성을 높일 수 있는 역할을 하길 바라는 반면 관리자로서의 책임도 함께 요구한다. 이로 인해 책임자는 쉽게 지치게 되어 사태가 악화되는 경우가 생긴다.

팀이 잠시 작업을 멈추고 스케줄링 작업을 할 시간이 없을 만큼 바쁘다면 문제가 있는 것이다.

2. 설계 리스크

완전히 새롭고 혁신적인 것을 만들어야 하는 시기와 방법을 이해하고 기존의 기술을 활용해야 하는 시기를 이해하는 것은 중요한 결정으로 이어진다. 많은 설계자들이 그럴 수 있는 능력이 있다는 이유만으로 일을 다르게 처리하곤 한다. 이는 때때로 좋은 결과를 낳기도 하지만 최대치의 리스크가 내재될 것으로 여겨지는 프로젝트로 끝나는 경우도 많다. 게임 설계가

프로젝트에 주게 될 리스크가 무엇이고 이를 어떻게 최소화해야 하는지 이해하는 것은 중요하다.

설계 리스크는 다양한 형태와 크기로 나타나므로 이들 모두를 분류하는 것은 불가능하다. 팀에서 한번도 해보지 않았거나 어떤 게임도 해보지 않은 것은 위험할 수 있다. 게임을 제대로 만들기 위한 자원이 많이 필요하거나 설계대로 하기 위한 툴이나 프로세스가 없음으로 인해 리스크가 발생하기도 한다. 결국에는 각 기능과 연관된 리스크가 많을지 적을 지만 판단할 수 있으므로 이런 리스크를 최소화하는 방법에 중점을 두게 된다.

초기에는 다듬어지지 않은 기발한 아이디어를 구상하는 것이 중요하지만 아이디어 구상에 어느 정도 한계선을 긋고 가능한 한 빨리 리스크를 줄이려고 노력하는 것도 바람직하다. 이런 노력에는 새로운 일을 시도하는 것을 제한한다든지 (내 경우 가능하면 프로젝트 당 보통 3~5 가지 새로운 기능을 시도한다) 다른 플랜으로 바꾸거나 플랜을 포기하기 전에 새로운 기능이 증명되어야 할 시간을 제한하는 것 등을 포함한다.

설계 파이프라인에서의 리스크도 확인해야 한다. 툴과 프로세스를 살펴보고 설계자가 가능한 한 빨리 콘텐츠를 만들고 수정하며 균형을 잡을 수 있도록 최적화해야 하며 또한 가능한 한 많은 설계자들이 같은 일에 대한 작업을 할 수 있도록 허용되어야 한다. 설계 툴 문제는 특히 개방된 게임, 롤플레잉 게임, 기타 더 복잡한 게임에서 더 크게 나타난다. 툴 파이프라인이 최적화되지 않는다면 한 팀원이 작업을 끝내지 못하면 다른 팀원이 작업을 시작할 수 없게 되며 스케줄에 뒤떨어지고 이를 따라잡지 못하기 때문에 게임 제작이 연기되는 리스크에 처하게 된다.

설계자 역시 프로젝트에 참여한 다른 사람이 뒤쳐져서 자산을 늦게 공급하게 되는 리스크에 처할 수 있다는 것을 기억하는 것이 특히 중요하다. 설계자들은 종종 코드를 맞추는 대부분의 작업을 하기 대문에 전체 팀이 마주치게 될 문제에 훨씬 더 민감하다. 설계자들은 다른 팀원들이 제대로 일을 할 수 있도록 노력하고 이를 보증해야 한다.

결국 설계를 잘못하거나 게임 디렉터가 다른 사람이 어떻게 생각하는지 고려하지 않고 팀을 이끌게 된다면 프로젝트에 큰 리스크가 될 수 있으므로 어떻게든 이를 막아야 한다.

3. 예술 리스크

설계 팀과 마찬가지로 예술 팀 역시 파이프라인과 연관된 리스크를 많이 지니고 있다. 파이프라인이 최적화되고 증명되지 않았다면 제작과정이 계획보다 훨씬 오래 걸릴 수 있는 리스크가 생길 수 있다. 머리 쓸 일이 없어 보이는 예술적 요소를 게임 엔진에 내보내는 것과 같은 일도 게으르거나 경험이 없는 프로그래머에 의해 쉽게 망가질 수 있다.

예술 팀에서 나타나는 리스크는 프로토타입 제작에서 대부분 증명될 것이다. 가장 큰 리스크는 때때로 엔진이 계속 변하면서 모든 예술 요소가 수정되고 그 때마다 다시 내보내기 해야 한다는 사실로 인해 생겨난다. 이는 새로운 엔진을 구현하는 팀에서 많이 일어나는 경향이 있다. 그렇지 않은 경우에 예술에서 나타나는 대부분의 리스크는 최소화되는 경향이 있다. 이론적으로는 한번 일이 완료되면 일반적으로 수백, 수천 번 "되새기고 반복하기"가 된다.

예술 툴과 파이프라인의 수정과 문제점

팀에서 예술 파이프라인을 제어하는 것은 매우 중요하다. 이런 문제는 특히 대규모 게임, 새로운 엔진을 사용하는 게임, 개발 사이클이 오래 걸리는 게임에서 나타난다. 일부 경우에서는 엔진 내의 예술 영역에 대한 세세한 사항이 변할 수 있어 간단하게 다시 내보내기 작업을 하거나 심하게는 게임 내 예술 부분의 재 작업에 이르기까지 많은 일이 일어나게 된다.

또 다른 문제는 예술가들이 쓰기엔 예술 툴이 너무 어렵게 만들어져 작업을 늦춘다는 것이다. 이는 보통 엔지니어가 툴을 최적화할 시간이 없기 때문에 일어나기도 하지만 다른 이유로 인해 초래되는 경우도 있다

전에 예술적 요소의 내보내기를 하고 엔진에 제대로 삽입하는 작업이 오브젝트 당 반일에서 하루 종일 걸리는 뭔가 잘못된 프로젝트를 진행한 적이 있었다. 오랫동안 문제가 툴이나 엔진과 연관된 기술적인 것이라고 생각했으며 예술가들은 이를 그냥 받아들이고 이로 인해 프로젝트 진행속도가 느려져도 아무 불만 없이 묵묵히 일했었다.

그렇지만 이 문제는 후에 예술 디렉터에게 "불만"이 있는 프로그래머가 일을 방해하기 위해 의도적으로 툴을 어렵게 만들어 그가 느꼈던 울분에 대한 보복을 한 것으로 판명되었다.

예술적 요소의 최적화

예술적 요소 제작에서 나타나는 또 다른 주요한 리스크는 일반적으로 수억 개의 자산을 추적해야 하고 게임을 구현해야 하는 프로젝트 후반에 나타난다. 대부분의 팀은 개발 사이클 초기에 게임이 늦어지는 것(일반적으로 30fps 이내)을 용인하고 프로그래머가 엔진을 최적화하여 보다 빠르게 실행될 수 있게 하고 프로젝트가 끝날 때에는 원하는 프레임 레이트로 실행될 수 있을 것이라고 믿는다.

이론적으로는 가능한 일이지만 실제로는 힘든 일로 마지막에 예술적 요소가 과도하게 최적화되곤 한다. 이렇게 되면 많은 시간을 허비하게 될 뿐만 아니라 예술 팀의 사기가 저하되는 결과를 낳는다.

마지막에 엔지니어링 팀이 예술적 요소를 최적화할 것으로 생각하게 된다면 이는 예술 팀에게 항상 큰 리스크로 다가올 것이다.

일부 팀은 예술 요소를 최적화하는 작업에만 시간을 쓰는 반면 다른 팀은 미리 이런 문제에 대해 고려하고 수많은 자산을 "선택적"으로 설계하여 수준의 견고성을 희생하지 않고도 특정 영역의 예술 요소의 25% 를 포기할 수 있게 한다.

그렇긴 하지만 나 역시 예술적 요소에 너무 비용이 많이 소요되어 문제가 된 프로젝트를 많이 보아왔다. 어떤 경우는 그것이 의도된 것이었으며 어떤 경우는 우연히 그렇게 된 것이었다. 단지 예술가가 잘 알지 못해서 그럴 경우도 있었다.

화면보다 10 픽셀쯤 위에 있어야 했던 요소가 잘못되어서 50,000 개의 다면체로 모델링되는 3D 장면의 실행속도가 느려지는 이유를 팀 내에서 알아내지 못했던 적이 있었다. 예술가가 특정한 가이드라인과 한계선을 갖지 못하는 것 역시 매우 위험한 일이다.

일반적으로 일어나는 또 다른 리스크는 게임을 경영진에게 "팔아야"하는 경우가 많기 때문에 일어나며 예술적 요소는 그들을 감명시킬 수 있는 가장 좋은 도구가 되는 경우가 많다는 것으로 인해 일어난다. 정말 놀랄 정도로 외관이 훌륭한 수많은 데모들을 보아왔다. 그렇지만 불행히도 게임플레이 시스템의 많은 부분은 아직도 기능을 하고 있지 않는데 그 부분에서 시스템 자원의 100%를 모두 사용해 버리는 경우가 많다. 결국 데모는 실제 게임과는 차이가 있어 팀에서는 관리부서에게 이것이 망쳐졌다는 것을 설명하는데 집착하거나 엔진을 더 빠르게 만들기 위한 최적화 작업을 할 수 있다고 생각하게 된다. 이 두 가지 해법은 주요한 리스크를 초래하므로 피해야 한다.

예술적 요소를 최적화하기 전에 예술적 요소가 정말로 게임 속도를 늦추게 만들고 있는지 확인해야 한다. 대부분의 3D 그래픽 칩과 엔진은 다면체나 텍스처를 만들어내는데 능하므로 (첨단적인 콘솔 게임에서)몇 개의 다면체나 픽셀을 최적화하기 위해 많은 시간을 소비하고 있다면 아마 다른 문제가 있을 것이다. 프로그래머가 속도가 느려진 것을 예술적 요소 탓으로

돌리는 경우가 여러 번 있었지만 실제 원인은 인공지능이나 물리, 또는 기타 시스템의 책임이었음이 밝혀진 경우도 여러 번 겪어 왔다.

4. 아웃소싱 리스크

“아웃소싱”이라는 단어는 일반적으로 다른 곳, 주로 중국, 인도, 동유럽, 기타 노동력이 저렴한 곳에 있는 팀의 힘을 빌려 예술 작업을 하는데 적용된다. 이런 팀의 기술력 수준은 편차가 많아 위험하게 여기는 사람들이 많다.

경험이 많지 않은 저렴한 팀들은 특히 더 위험하지만 이런 팀들에 대한 최선의 방법과 적절한 관리를 통해 많은 리스크를 완화할 수 있다.

외부 예술 팀과 일한 경험이 없다면 관리 업무가 힘들 수 있다. 시간차의 문제, 시간차, 또는 언어소통의 문제 등의 의사소통 문제, 기술적 문제, 예술적 요소의 품질 문제 등 다루어야 할 문제가 대단히 많다.

그렇지만 외부 팀을 고용하기 전에 이들을 제대로 검증하여 이들이 기술적 요구사항을 제대로 이해하고 실행할 수 있다는 것을 확인하고 이들이 요구되는 품질의 작업을 할 수 있으며 분명한 방향성을 지니고 시간을 엄수하며 피드백을 통해 적절하게 의사소통을 할 수 있다면 이러한 리스크는 줄어들 수 있다.

예술 작업을 아웃소싱하는 것은 힘들 일이겠지만 대규모 예술 팀을 고용하고 이들에게 계속 일을 주어야 할 리스크를 최소화하면서 많은 시간을 절감할 수도 있다. 이렇듯 많은 팀을 활용하여 수행 작업의 양을 늘릴 수 있으며 또한 외부 팀들의 강점과 약점을 활용할 수 있다.

5. 프로그래밍 리스크

프로그래머 역시 프로젝트를 수행하는 동안 수많은 리스크에 노출된다. 적합한 기술을 선택하는 것 외에 프로그래머에게 가장 큰 리스크는 아마도 설계자가 될 것이다. 끊임없이 마음을 바꾸고 적절한 세부사항을 규정하지 않고 즉흥적으로 설계를 하며 불가능한 게임플레이 아이디어를 설계하고 기술적으로 할 수 있는 것을 설계하려고 하지 않는 설계 팀으로 인해 프로그래밍 팀이 궤도를 벗어나게 되는 경우도 종종 있다.

그렇지만 프로그래밍 팀 역시 소스 제어 시스템을 이용하고 올바른 코딩 표준을 따르며 적절한 시간을 툴에 투자하는 등과 같은 자신들의 업무를 제대로 해야 한다는 것은 당연한 일이다.

엔진 기술 리스크

게임 프로그래밍 팀은 알맞은 엔진과 기술을 선택하는 데서 가장 큰 리스크를 만나게 된다. 팀에서 새로운 엔진을 찾기로 했다면 정말 거대한 리스크를 떠안게 되는 것이다. 팀에서 엔진에 대한 라이선스를 가지려는 것 또한 많은 리스크를 낳게 하는데 특히 팀 내 몇 명이 이전에 해당 엔진을 사용해 본 경우에는 더욱 위험하다. 그러므로 리스크를 최소화하기 위해서 프로그래밍 팀은 어떤 일을 결정하기 전에 적절한 시간을 투자하여 가능한 모든 선택사항과 기술을 배우고 평가해야 할 필요가 있다.

이는 매우 어렵고 논란을 일으키는 리스크이다. 일부 팀은 엔진을 라이선스 하는 것을 선택하는 반면 일부 팀에서는 자신만의 엔진을 선택할 것이다. 만일 라이선스 하려는 엔진을 사용해 본 적이 없고 팀 내에 아무도 그것을 사용해 보지 않았다면 이에 대해 알아보기 위해 오랜 시간을 들여야 하는 것과 연관된 많은 리스크가 발생한다. 어떤 엔진을 사용할지 조사할 경우에 팀은 엔진의 성능만을 보는 것이 아니라 어떤 기능을 사용할 수 있는지, 툴은 어떻게 작동하는지 등 여러 측면을 살펴봐야 한다.

예를 들어 캐주얼 게임을 만드는데 주로 이용되는 저렴한 게임 엔진들은 대규모 게임을 실행할 수 있는 성능을 갖추지 못했을 뿐만 아니라 대규모 팀이 효율적으로 사용하기에는 사용하는 툴이 매우 적절하지 않은 경우가 많다.

일부 엔진은 유명한 게임에서 이용되었다고 광고를 하지만 개발자와 이야기를 나눠보면 엔진에서 꽤 많은 부분을 다시 작성했으며 이들이 제대로 작동하려면 대부분의 자원을 할당해야 한다는 것을 알 수 있을 것이다. 그러므로 해당 엔진을 사용했던 작품과 똑같은 모방 작품을 만든다고 해도 실제로 이를 구현하기 위해 얼마나 작업을 해야 하는지 알 수 없다.

예산 금액을 대충 알아보려면 해당 엔진을 사용하는 프로젝트를 위한 팀의 변동 내역을 살펴보고 그 프로젝트에 얼마나 많은 엔지니어가 투입되었는지 확인하면 된다. 게임 개발에 오랜 시간이

걸렸거나 내역에 추가적으로 수많은 엔진 프로그래머가 투입되었다면 그 이유에 대해 알아봐야 한다.

이제까지 나는 사용할 엔진을 잘못 결정해서 프로젝트가 취소되거나 크게 지연되는(1 년 이상) 경우를 많이 목격해왔다.

미들웨어 리스크

미들웨어를 사용하는 것은 매우 훌륭한 결정일 수도 있지만 이 결정이 악몽으로 이어질 수도 있다. 미들웨어는 광고대로 동작하기도 하지만 믿기 힘들 정도로 느리거나 엉성할 수도 있다. 완전한 게임의 엔진을 고를 때와 마찬가지로 미들웨어를 고를 때도 주의 깊게 모든 기능과 성능과 관련된 리스크를 확인해야 한다.

메모리 맵이 만들어져야 하며 미들웨어가 예상 범위 안에서만 실행된다는 것을 확인해야 한다. 많은 팀들이 미들웨어를 완벽한 환경 하에서 테스트하기 때문에 수많은 기타 프로그램과 같이 수행되는 곳에서 게임을 할 때 실제로 무엇이 일어나는지 파악하지 못하고 있다.

미들웨어 회사의 마케팅 문구를 그대로 믿지 않는 것도 중요하다. 프로젝트에서 미들웨어를 사용하기로 정한다고 해서 팀에서 해당 미들웨어를 그대로 사용한다는 것을 의미하지는 않는다. 개발자는 미들웨어의 일부만을 사용할 수도 있으며 팀이 실제로 미들웨어의 문제점을 고쳐 실제로 잘 돌아가게 하기 위해 미들웨어를 많이 수정했었을 수도 있다.

6. 테스팅 리스크

프로젝트에 버그가 너무 많게 되면 큰 문제가 된다. 일부 팀에서는 버그가 쌓이게 내버려 두고 프로젝트를 진행하면서 눈에 띄는 것만 고치는 반면 매주마다 문제점을 고치고 코드를 최적화하기 위해 많은 시간을 쏟는 팀도 있다.

버그를 고치는 시간이 길어질수록 버그를 바로잡기도 힘들고 리스크가 증가할 수 있다. 그렇지만 초기에 버그를 고치는데 시간을 너무 많이 소모하게 되면 그만큼 일이 진척되지 않기 대문에 스케줄을 지키지 못하거나 다른 문제가 생길 리스크가 나타난다.

게임 개발이 최종단계로 들어설수록 버그 숫자는 점점 더 중요한 의미를 지니게 된다. 스케줄에 따라서 버그 숫자는 베타 출시 때나 그 이후에 가장 집중적으로 관리된다.

그렇지만 일반적으로 프로젝트에서 3~4 개월이 남았고(표준적으로 24 개월 개발 기간을 고려할 때) 데이터베이스에 수천 개의 버그가 있다면 제 시간에 출시하지 못할 것이다.

그러므로 프로젝트 후반기, 또는 일정이 1/3 정도 남았을 때는 끊임없이 버그 수를 모니터하고 리스크를 평가하는 것이 중요하다. 그렇지 않으면 프로그래밍 팀이 날마다 추가되는 새로운 기능이나 기존 버그 수정 결과, 또는 테스팅 팀이 날마다 발견해내는 새로운 버그 수정 등의 작업을 제 시간에 해내지 못할 것이다.

리스크 없애기

이제 여러분은 리스크를 파악하고 초기에 리스크를 없애는 방법을 이해한다는 것이 얼마나 중요한지 알게 되었을 것이다. 리스크를 없앤다는 것은 자기 자신과 다른 사람의 실수로부터 배워야 함을 의미한다. 리스크를 없애기 위해서는 현명해야 하며 항상 새로운 문제를 인식하는 한편 예전 문제점을 해결하여 다시는 일어나지 않도록 해야 한다.

여러분이 인식하고 인정해야 할 가장 중요한 사항은 무엇을 알지 못하는가를 아는 것이다. 수많은 회사에서 자신이 모든 것을 알고 있다고 착각하는 재능이 뛰어난 사람들로 팀이 구성되는 경우가 많은데 이들은 결국 예전 팀에서도 지식이 풍부한 전문가들이 많았지만 지금은 작업하지 않는다는 것을 깨닫곤 한다.

현재의 문제점을 파악할 수 있는 직원을 계속 고용할 수는 없겠지만 이런 직원은 꼭 필요하다. 충고를 해주고 방향을 제시해줄 수 있는 숙련된 계약직원이나 컨설턴트가 많이 있다. 만일 리스크를 없애고자 한다면 여러분이 내리는 의사결정은 추측이 아닌 실제 지식에 기반을 두어야 한다.

결국은 방어적인 개발 모델을 구축해야 한다. 이는 마치 방어 운전과 같다. 길에서 속도를 늦출 줄도 알아야 하고 교통상황이 변하면서 달라지는 주변상황의 분명하고 잠재적인 문제를 항상 인식해야 한다. 코스를 바꾸거나 길에 있는 장애물에 부딪히는 대신 피해서 갈 수 있는 여유가 있어야 한다.

방어운전에서와 마찬가지로 프로젝트를 개발할 때의 “주변을 살피는 시각”과 관심은 다가오는 문제를 항상 조심하고 문제가 실제로 일어나기 전에 이를 어떻게 해결할 수 있을지에 초점을 맞춘다. 현실에서 이는 대단히 어려운 일로 불가능할 때도 있지만 리스크를 없애려는 노력을 하지 않는다면 결국 그로 인해 프로젝트가 취소될 수도 있다.

리스크 줄이기

여기서 적은 리스트는 프로젝트에 존재하는 리스크의 완전한 리스트에는 전혀 미치지 못한다. 각 프로젝트마다 리스크가 너무 다양하게 나타나기 때문에 모든 리스크를 적은 리스트는 있을 수 없다. 결국 잘못될 수 있는 사항을 찾아내고 분석 대상으로 만들기 위해서는 두뇌와 스케줄, 기능 리스트, 과거 경험, 그리고 본능을 이용해야 한다.

리스크를 줄일 수 있는 최선의 방법은 그것에 대해 언급하고 이를 없앨 수 있는 계획을 세우는 것이다. 팀에 속하지 않은 외부인과 이야기를 나누면서 그들이 비슷한 문제를 겪은 적이 있는지 묻고 충고를 얻도록 한다. 가마수트라나 게임 개발자(Game Developer)의 프로젝트 후기를 읽는 것도 도움이 될 것이다. 이런 글에서는 프로젝트에서 나타나는 많은 문제점들을 지적하고 있어 기본적인 리스크 분석만 해도 리스크를 없앨 수 있는 경우도 많다.

리스크에 대처하려면 리스크를 확인하는 일 외에 리스크를 없애는 계획과 더불어 어쩔 수 없이 일어나야 한다면 어떻게 리스크를 받아들이는가에 대한 계획을 가져야 한다. 리스크가 확인될 경우에는 다음 중 한가지 방법을 취할 수 있다. 확인되자마자 이를 제거하려고 노력하는 방법, 또는 언제 리스크를 받아들이지 않고 더욱 노력하여 제대로 된 기능을 구현할 것인지를 결정할 계획을 세우는 것이다.

리스크가 전혀 없다면 프로젝트를 혁신하려고 노력하지 않고 최대한 게임 규모를 키울 것이기 때문에 프로젝트에 나타나는 일부 리스크는 이로운 역할을 하기도 한다. 게임에 어떤 리스크도 없다고 말할 수 있다면 정말 운이 좋거나 아니면 현실을 자각하지 못하거나 둘 중의 하나이다.

대규모 프로젝트에서는 소규모 게임과 매우 다른 리스크나 문제가 나타나기 때문에 소규모 게임에서 사용했던 것과 동일한 리스크 평가 방법을 사용할 수 없다는 사실을 기억해야 한다.

모든 게임의 요구사항은 서로 완전히 다르다. 자기만족에 빠져서 익숙한 것이라는 이유로 예전에 사용했던 것과 동일한 프로세스를 계속 사용하지 않아야 한다는 것을 명심해야 한다. 각각의 프로젝트에서의 개발 프로세스를 계속 개선하고 리스크가 실제로 최소화하고 있다는 것을 검증하는 것이 중요하다.

무슨 일이 일어나든 끊임없이 전체 팀과 이야기를 나누고, 함께 작업하며, 숨기지 않고 개방적으로 작업을 한다면 문제가 훨씬 쉬워진다는 것을 기억해야 한다.

사진: 스콧 맥스웰(Scott Maxwel), 오픈데모크래시(openDemocracy), wwworks의 작품을 CCL(Creative Commons license) 에 따라 사용함.