더티 코딩 트릭(Dirty Coding Trick)

(Dirty Coding Tricks)

Brandon Sheffield

2009. 10. 19

http://www.gamasutra.com/view/feature/4111/dirty_coding_tricks.php

[일정은 꽉 차 있고 게임은 보내야 하는 경우 프로그래머는 게임 프로젝트를 끝내기 위해 비열한 코딩 트릭을 사용할 수 있다.
올해 초 Gamasutra 자매지인 Game Developer 매거진에서 출판한 이 아티클을 통해 이상과 같은 실제 사례 9가지를 소개한다.]

프로그래머는 작업한 코드가 말끔하게 유지되도록 최선을 다하고 규칙적이며 정확한 성격의 소유자이다. 완벽하게 계획한 일정에 차질이 생기고 게임을 보내야 하는 시점이 임박해 오면 "일을 완수해야 한다"는 압박감으로 총력을 기울이게 된다.

이러한 경우 당황한 프로그래머는 베스트 프랙티스를 무시하고 바람직하지 않은 해결책과 임시방편으로 일을 마무리하려고 할 수 있다. 여기서는 납기가 임박한 시점에서 프로젝트를 완수하기 위해 트릭을 사용한 개발자에 관한 9 개 사례를 소개하고 있다.

- Brandon Sheffield

[공유할 더티 코딩 트릭이 있으면 Brandon Sheffield에게 전자메일(bsheffield@gdmag.com)을 보내 주세요. 이 아티클의 삽화는 Jonathan Kim이 제공함.]

예쁘게 찍어 줘

약 4 년 전에 멀티플랫폼의 PlayStation 2, Xbox 및 GameCube 릴리스에 관한 프로그래머로 일한 적이 있었다. 개발이 끝나가려고 할 때 타이틀을 보내기 전 임시 코드(code hack)를 게임에 포함시키는 것은 놀라운 일이 아니었다. 특히 PS2 버전은 추적하기가 어려운 문제점이 있었다. 즉, 게임의 첫 레벨에 있는 플레이어 캐릭터가 장시간 테스트에 노출되면 게임이 얼어 버리는 현상이 발생했다. 불행히도 이것은 디버깅 정보가 없는 디스크 리테일 빌드(retail build)에서만 발생했다. 소니의 TRC(technical requirement check)를 지키지 않아서 그런 것이 아닌가라고 생각하며 열심히 해결책을 찾기 시작했다.

문제의 범위를 좁히기 위해 다른 코드 섹션에 대해 다른 컬러를 사용하면서 화면 가장자리의 경계를 계속 렌더링했다(예: 청색은 렌더링 셋업, 녹색은 플레이어 컨트롤 업데이트를 나타냄). 크래시가 간헐적으로 일어났기 때문에 타이틀 담당 리드 엔지니어와 나는 수동으로 디스크를 굽고 PS2 상에서 게임을 실행했다 (이 업체는 IT 팀 또는 펀시 디스크 버닝 머신이 없는 독립 개발업체임). 코드를 수정하고 디스크를 굽고 배포하고 크래시가 발생한 지점의 근사값을 구하는 데에만 1 시간이 걸렸다. 마감에 임박해 있었으므로 테스트를 해 볼 수 있는 시간도 얼마 되지 않았다.

주말 동안 크래시 때문에 사무실에서 *World of Warcraft* 를 하며 시간을 보내던 중, 누군가가 카메라 각도가 오른쪽 90 도로 잡혀 있을 때는 버그가 발생하지 않는다는 것을 발견했다. 초기에 프로그래밍팀은 일종의 요행수로 버그를 마스킹하여 문제를 해결했었다. 이번은 그런 유형의 문제가 아니라 근본 원인을 해결해야 하는 것이었다. 이로 인해 게임을 출시하지 못하는 경우가 생겨서는 안되었다. 데드라인이 다가옴에 따라 카메라 각도를 오른쪽 90 도로 고정하여 최종 PS2 디스크를 생성했다. 결국, 이 게임은 플랫폼 홀더의 TRC 테스트를 통과하여 제 때에 출시되었다.

이것을 “코딩 트릭”이라고는 할 수 없으나 분명 “올바른” 방법으로 문제를 해결한 것은 아니었다. 문제를 어떻게 해결하게 되었는지는 아직도 미스터리이지만 다행스럽게도 사용자가 클레임을 제기한 적이 없었다.

– Mark Cooke

정체성 위기

이 사례는 모든 개발자들에게 친숙한 내용이다. 우리 회사의 Xbox 1 게임의 금메달 후보감을 출시하는 날이었다. 하루 종일 전체 팀이 게임을 플레이테스트하며 모든 것이 제대로 돌아가는지를 확인했다. 이 게임은 재미있고 잘 만들어졌으므로 성공할 것이라고 생각했다.

오후에 마지막으로 게임 밸런싱을 조정한 다음 최종 빌드를 만들고 마지막 플레이스루 세션(playthrough session) 작업을 하고 있을 때 게임에 크래시가 발생했다. 워크스테이션으로 달려가 디버거를 실행하고 무슨 문제인지 알아보려고 노력했다. 이것은 assert 와 같은 사소한 것 또는 divide by zero 같이 추적하기 어려운 것이 아니었다. 가비지 메모리와 같은 것처럼 보였으나 메모리 보고는 문제가 없는 것으로 나왔다. 무슨 일이지?

어느 날 저녁 많은 시간이 지난 후 정시에 출시하려는 우리의 꿈은 사라져갔고 잘못된 데이터가 로딩된 데이터 파일을 추적하게 되었다. 잘못된 데이터라? 어떻게 그게 가능했을까? 리소스 시스템은 게임 데이터를 관리하며, 모든 데이터 파일의 ID 는 64 비트로, 파일명의 CRC32 해시와 실제 데이터 내용의 CRC32 해시의 조합으로 생성된 것이었다. 게임에서는 동일한 리소스 파일을 단일 파일로 통합하고 있었다. 이 시스템으로 수만 개의 파일을 2 년간 관리했으나 지금까지 전혀 문제가 없었다.

그날 오후, 담당 디자이너는 한 텍스트 파일이 다른 리소스 파일과 분명 다른 것임에도 불구하고 동일한 파일명과 데이터 CRC 를 갖고 있다는 것을 발견하게 되었다.

모두 문제의 심각성을 인식하고선 절망하기 시작했다. 그렇게 짧은 시간에 리소스 인덱싱 시스템을 변경할 방법은 없었다. 밤을 샌다고 하더라도 아침까지 모든 것이 정상화될 거라는 확신이 전혀 없었던 것이다.

그러자 절망감이 밀려 왔다. 하지만 게임의 정시 출시를 위해 노력해야 했고 이 과정에서 오류를 수정할 수 있는 방법을 발견하게 되었다. 문제가 된 텍스트 파일을 열어서 파일 마지막에 빈 칸을 하나 넣은 후 다시 저장하였다. 우리는 서로를 쳐다보고 싱긋 웃고는 “출시하자!”라고 말했다.

- Noel Llopis

음주 운전

나는 차량을 다루는 시스템 관리 업무를 맡고 있었는데 이것은 우리 회사 게임의 중요한 부분이었다. 대부분의 코드를 다른 스튜디오로부터 확보했는데, 유감스럽게도, 이 코드는 완전하게 잘 작성된 것이 아니었다. 이 코드가 단지 엔진 루프로부터 변수를 얻으려고 한 것이었고 대부분 백워드 방식으로 작성된 것임을 알게 되었다(Listing 1 참고).

Listing 1: Driving Under the Influence

```
//*************************************************
// Function: AGameVehicle::Debug_GetFrameCount
//
//! A very hacky method to get the current frame count; the variable is protected.
//!
//! ₩return The current frame number.
//*************************************************
UINT AGameVehicle::Debug_GetFrameCount()
{
BYTE* pEngineLoop = (BYTE*)(&GEngineLoop);
pEngineLoop += sizeof( Array<FLOAT> ) + sizeof(
DOUBLE );
INT iFrameCount = *((INT*)pEngineLoop);
return iFrameCount;
}
```

전체 코드에 관해 가장 웃긴 점은 같은 오브젝트에 프레임 카운트를 되돌려주는 함수가 존재하고 있다는 것이었다. 함수가 존재하지 않는다 할지라도 이 코드를 작성한 사람은 간단히 추가할 수가 있었을 것이다. 이것을 발견하자마자 두 말할 필요도 없이 이 코드를 내 게임에도 코드를 가져온 게임에도 추가하지 않았다. 코드 리뷰를 했을 텐데 왜 이런 결과가 나왔는지 모르겠다.

- Austin McGee

10-Tative 코드

프로젝트가 끝나갈 무렵, 레벨 중의 하나에 숨겨져야 할 오브젝트 하나가 있음을 발견하게 되었다. 우리는 레벨 전체를 뜯어 고치거나 체크섬 이름을 사용하기는 싫었다. 그래서 엔진 코드의 중간에 다음 코드를 사용했고, 게임은 출시되었다.

```
if( level == 10 && object == 56 )
{
```

HideObject();
}

약 1 년 뒤, 우리 엔진을 사용하는 한 아티스트가 10 레벨로 오브젝트를 옮긴 후 레벨에 오브젝트가 안 나타난다고 불평했다. 왜 그런 일이 생겼을까?

-익명

모든 상황이 "아니오"라고 말하다

Raven Software의 새로운*Wolfenstein*를 개발하며 360 에 대한 컨트롤러 지원을 셋업하던 중 문제가 발생했고, 라이브 통합을 위해 어느 컨트롤러가 입력 이벤트를 보내고 있는 중인지를 알고 있어야만 했다. 우리가 사용 중인 *Doom 3* 입력 코드는 *Quake 3*로부터 가져온 것으로, 매운 간단한 시스템이었다.

기존 이벤트 시스템에서 여분의 매개변수가 필요한 경우 각 이벤트를 정수 인수 2 개 및 보이드 포인터로 패스했다. 작업의 목표는 컨트롤러 ID 와 입력 이벤트를 연관시키는 것이었다. 즉, 컨트롤러 ID 를 이벤트의 정수 인수 중의 하나에 삽입하기만 하면 되었다.

또한 프레임 할당 시 조각 없음을 사용하여 일부 메모리를 예비해 두고 컨트롤러 ID 를 지정한 다음 이벤트 포인트 슬롯에서 포인터를 전달하도록 했다.

이벤트 시스템은 이벤트를 처리한 후 이벤트의 보이드 포인터를 해제하는 역할을 하고 있었다. 이것은 우리가 사용 중인 멀티힙(multiheap) 접근법과는 잘 맞지 않았다. 또한 free()를 호출하는 이벤트 코드에 의지하고 있는 레거시 *Doom 3*가 존재했으므로, 코드베이스를 대대적으로 변경하지 않고는 이 호출을 제거할 수가 없었다. 세 번째 정수 매개변수를 추가하는 것은 어떨까? 그렇게 하려면 수백개 함수 호출을 변경해야만 했다.

데드라인이 가까워져서 여기에 더 이상 시간을 할애할 수가 없었다. 그래서 컨트롤러 ID 를 포인터 매개변수에 삽입했다. 대문자로 4 줄 정도 주석을 달아 이 임시 코드를 표시해 두었는데, 잠시 동안 효과가 있었으나 결국 전체 입력 시스템을 좀
더 나은 것으로 교체해야 했다.

- David Dynerman

벨크로잉 발생

초기 PS2 시절의 프로젝트에 관한 이야기이다.

“충돌체” 모델에 사용하기 위해 캐릭터 충돌을 다시 작성하여 막판에 충돌/경계 문제를 해결한 적이 있었다(기울어진 박스의 계층 트리 보다는 범프 대 박스 방식으로 문제를 해결).

우리 게임에는 드문 형태의 버그(벨크로잉이라 불렀음)가 발생했는데 가끔 플레이어 캐릭터가 벽에 부딪혀 미끄러지곤 했다. 또한 충돌체 영역이 벽의 잘못된 쪽에 있거나 플레이어가 벽으로부터 벗어나지 못하는 경우가 존재했다(예: 벽 표면에 붙어 있음).

이것은 아주 간단해 보이는 버그 중의 하나로, 충돌체 영역이 잘못된 쪽에 있게 된 이유를 알아내어 이를 수정하기만 하면 되었다. 그러나 문제는 이 이유를 알아내기 위해서는 모든 충돌 처리에서 발생한 것을 파악해야 한다는 점이었다.

최상의 해답을 찾기 위해 조건부 중단점을 사용하는 경우(부적절한 박스 또는 유사한 어딘가에 있는지 혼란스럽게 만드는) 프레임 속도가 너무 느려지고 상당히 큰 문제가 발생할 수 있었다(실제로 이 버그를 수정하려면 피와 땀과 눈물 어린 노력을 기울여야만 했음).

게임을 제출하기 위해 노력하면서 이 문제를 블로킹 버그와 관련된 것으로 간주하고 이에 따라 해결책을 찾아 보았다. 한 가지 해결책은 문제가 발생한 지점의 경계를 버퍼링하는 것이었다. 이렇게 할 경우 다른 문제가 발생하지는 않지만 올바른 방법으로 수정할 수 있는 것이 아니었다. 우리는 버그를 아트 팀으로 보냈고 이 미묘하고 심각한 문제를 추적하기 위해 총 처리 시간을 감안하여 시간을 할애했다.

테스트 담당자는 “플레이어가 보이지 않는 벽에 부딪혔는데, 이 벽에 부딪혀서는 안되는” 심각한 버그라는 답변을 보내왔다. 결국 게임의 나머지 부분을 정리하면서 피와 땀과 눈물 어린 노력을 기울여 이 버그를 수정했다.

이것은 자랑할만한 정도의 버그 처리 시간은 아니었으나 최선을 다했던 하나의 사례라고 생각된다. 우리는 “아니오, 어제 테스트를 위해 제출했어요”라고 말한 후 지독하고 넌더리나는 문제를 서둘러 해결하기 위해 온 힘을 기울였기 때문이다.

-익명

땜질로 내 강아지 구하기

다음과 같은 한물 간 농담이 있다.

환자: "선생님, 이것을 하면 아파요."
의사: "그럼, 그것을 하지 마세요."

웃기지만, 적절한 상황에 적용한다면 현명한 말일 수 있다. 나는 PC 용 3D 3 인칭 슈팅 게임을 PS1 으로 변환하는 작업을 한 적이 있다.

ps1 은 부동소수점수 지원이 없었으므로 PC 코드를 다시 컴파일하고 모든 부동소수점 데이터를 고정소수점으로 오버로딩하여 변환하는 과정을 거쳤다. 이것은 상당히 효과가 있었으나 이로 인해 충돌점 측정이 완전히 망가져버렸다.

우리에게 제공된 평면 구조는 PC 버전의 게임에서는 잘 실행되었으나 고정 소수점으로 변환될 때 고정소수점과 부동소수점간의 값 차이로 인해 이음새(seam), T 자 접합부(T-Junction) 등과 관련된 문제가 나타나기 시작했다. 이 문제로 인해 아주 작은 구멍들이 생겼고 메인 캐릭터인 Damp 가 이 구멍에 빠져서 평면 아래의 나락으로 떨어지는 사태가 발생했다.

우리는 Damp 가 더 이상 빠져 나가지 못할 때까지 발견한 구멍을 임시 방편으로 메웠다. 그런 다음 유통업체에 테스트를 위해 게임을 보냈는데 갑자기 무수한 버그가 발생했다는 보고가 들어왔다. 매일 프레시 배치 위치에 Damp 가 빠져 나갈 수 있는 작은 구멍이 발견되었다. 이 구멍을 메워서 다음날 업체로 보내면 10 개 이상의 구멍이 또 발견된 채로 돌아왔다. 며칠 간 계속 이러한 작업을 수행했다. 유통업체의 테스트 부서는 Damp 가 빠져 나갈 수 있는 장소가 없는지 찾아보면서 하루 10 시간 동안 게임 전체를 둘러볼 사람을 고용했다.

평면 구조 자체에 문제가 있었다. 즉 이음새 없이 촘촘한 구조가 아니었다.게임은 PC 상에서는 잘 실행되나 PS1 에서는 그렇지 못했는데, 고정 소수점 연산방식으로 인해 문제가 더 커졌다. 이상적인 해결책은 이 평면 구조가 이음새 없이 연결되도록 수정하는 것이었다.

그러나 이것은 방대한 작업이었고 우리의 제한된 리소스와 시간으로는 불가능했으므로 테스트 부서에서 문제 영역을 찾아 주는 것에만 의지하고 있었다.

이런 식으로는 문제가 결코 해결될 것 같지 않았다. 또한 새로운 형태의 동일한 버그가 매일 보내져 왔고 이것은 도저히 언제 끝날 지 가늠할 수 없는 성질의 것이었다.

그러나 결국 무엇이 문제인지를 깨닫게 되었다. 즉, 문제는 평면 구조에 작은 구멍이 생겼다는 것이 아니라 Damp 가 그 구멍에 빠진다는 거였다. 이러한 생각이 떠올라 신속하게 코딩하여 다음과 같이 버그를 수정하였다.

IF (Damp will fall through a hole()) THEN
Don't do it

실제 코드는 이것에 비해 그다지 복잡하지 않았다(Listing 2 참고).

Listing 2: Meet My Dog, Patches

```
damp_old = damp_loc;
move_damp();
if (NoCollision())
{
damp_loc = damp_old;
}
```

단번에 천 개의 버그를 수정했다. 그 후 Damp 는 더 이상 평면 구멍에 빠지지 않았고 그 대신 구멍 쪽으로 걸어갈 때 조금 떨리는 편이다. 우리를 성가시게 하는 것이 무엇인지를 발견하고 이것을 차단해 버렸다. 유통업체는 매일 게임 전체를 둘러 보게 한 테스터를 그만두게 하고 게임을 출시했다.

"if A==bad then NOT A"의 성공에 힘입어 나는 이 툴을 사용하여 많은 버그를 수정했다(충돌 코드와 거의 관계가 없음). 그러나 개발이 끝나갈 무렵 버그가 점점 더 많아졌고 그에 따라 수정 횟수도 더 많아졌다.

Listing 3: Meet My Dog, Patches

```
if (damp_aliencoll != old_aliencoll &&
strcmpi("X4DOOR",damp_aliencoll->enemy->ename)==0 &&
StartArena == 6 && damp_loc.y<13370)
{
damp_loc.y = damp_old.y; // don't let damp ever
touch the door.. (move away in the x and y)
damp_loc.x = damp_old.x;
damp_aliencoll = NULL; // and say thusly!!!
}
```

이 코드가 무엇이 문제지? 기본적으로 Damp 가 특정 위치에서 특정 평면에 있는 특정 유형의 문을 터치하는 경우 문제가 발생했다. 문제의 근본 원인을 해결하기 보다는 Damp 가 문을 터치하면 그를 멀리 이동시켜 더 이상 구멍에 빠지지 않을 것 보이게 했을 뿐이었다.

돌이켜 보면 이 코드가 아주 끔찍하게 느껴진다. 나는 버그를 완전히 해결하기 보다는 임시로 잠깐 손을 봤을 뿐이었다. 불행하게도, 버그를 수정하려면 PS1 고정 소수점으로 전체 게임 구조와 충돌 시스템 모두를 다시 작업해야만 했다. 일정은 빡빡했고 프로젝트 마감일이 가까워졌기 때문에 포괄적이고 비용이 많이 드는 것 보다는 임시로 수정하는 쪽을 선택했다.

그러나 결국 이러한 방법은 좋지 않은 결과를 낳았다. 이 후, 수 백 개의 패치를 필요로 하게 되었고 패치 자체로 인해 문제가 발생하기 시작했으며 이 문제를 해결하기 위해 더 많은 패치를 추가해야만 했다. 버그는 계속 발생했고 패치로 이것을 계속 수정해 나갔다. 그러나 결국 일정보다 몇 개월이나 늦게 게임을 출시하게 되었고 이 기간 동안 하루 14 시간씩 작업을 수행해야 했었다.

이러한 경험으로 인해 "패치"에 대한 경각심을 갖게 되었다. 현재는, 패치가 간단하고 안전하고 이를 사용할 수 있다 하더라도 버그의 근본 원인을 먼저 알아보려고 노력한다. 내 코드에 버그가 생기지 않았으면 좋겠다. 우리는 대개 의사에게 가서 "이것을 하면 아파요."라고 말한 다음 의사가 왜 아픈지를 찾아내서 치료해 주기를 바란다. 신체의 고통과 코드의 버그는 훨씬 더 심각한 증세일 수 있다.

우리가 의사에게서 바라는 그대로 코드의 버그도 수정해야 함을 깨닫게 되었다.

- Mick West

내가 화났을 때 그렇게 하지마

이전에 THQ studio Relic Entertainment에서 *The Outfit*(Xbox 360 용 초기 게임 중의 하나)에 관한 작업을 한 적이 있다. 우리는 PC 엔진(싱글 쓰레드)으로 시작했고 이것을 차세대 멀티 코어 콘솔 상에서 18 개월 내에 완벽한 게임으로 변환해야 했다. 출시 약 3 개월 전 게임이 360 상에서 약 5 fps의 속도로 돌고 있었는데, 최적화가 상당히 필요하다는 생각이 들었다.

몇 가지 성능 테스트를 했을 때 코드가 느린 만큼 "PC"도 느리다는 것을 알게 되었고 더 큰 문제는 게임의 콘텐츠였다. 예를 들어, 모델이 너무 복잡하고 셰이더에 계산이 너무 많이 필요하며 일부 임무로 인해 너무 많은 캐릭터가 한 스테이지에 몰리게 되는 것 등이었다.

그러나 100 명의 팀원들 하나하나를 설득하면서, 프로그래머가 엔진 성능을 고칠 수 없으며 익숙해진 작업 방식을 바꾸어야 한다고 말하기는 힘들었다. 게임의 성능은 모든 사람들이 관심을 갖고 있는 문제라는 점을 이해시켜야만 했다. 이를 위한 최상의 방법은 이면에 진실이 숨겨진 유머를 이용하는 것이었다.

솔루션을 만들어내는 데는 1 시간이 걸렸다. 동료 프로그래머는 내 얼굴을 찍은 사진 4 장(웃는 얼굴, 보통 얼굴, 약간 화난 얼굴 및 분노로 머리를 쥐어 뜯는 얼굴 등)을 갖고 있었다. 이 이미지를 화면 코너에 집어넣고 프레임 속도와 링크시켰다. 즉, 게임이 30 fps 이상으로 돌고 있으면 웃는 얼굴이 뜨고 20 fps 이하로 돌면 화난 얼굴로 바뀌었다.

이로 인해 속도 문제에 대한 사람들의 생각이 "프로그래머들이 고치겠지. 뭐"에서부터 "음, 이 모델을 넣으면 닉이 화를 내겠지! 우선 최적화를 해 보는 게 좋겠어."로 변했다. 사람들은 그들이 노력한 바가 프레임 속도에 미치는 영향을 즉각적으로 확인할 수 있었고 결국 게임을 30fps 의 속도로 출시할 수 있었다.

- Nick Waanders

프로그래밍 반영웅

대학을 갓 졸업한 풋내기였을 때 베타 단계의 프로페셔널 게임 프로젝트(90 년대 후반 PC 타이틀 작업)에 처음으로 참여했었다. 이것은 롤러 코스터를 타듯 아주 즐거운 일로, 모든 콘텐츠가 제대로 갖추어진 멋진 게임을 개발하는 것이었다. 그러나 한 가지 문제가 발생했는데, 메모리 용량이 초과되고 있었다.

모델과 텍스처가 대부분의 메모리를 차지하고 있었기 때문에 우리는 아티스트와 협력하여 게임의 메모리 사용량을 가능한 많이 줄이려고 노력했다. 즉, 이미지 크기를 줄이고 모델 수를 줄이고 텍스처를 압축했다. 어떤 때는 아티스트의 지원을 받아 이러한 작업을 진행했고 또 어떤 때는 할 수 있는 최대의 선까지 메모리 사용량을 줄이려 애를 썼다.

몇 일 동안 미친 듯이 몇 메가바이트씩 줄여나갔지만 더 이상 어떻게 할 수가 없는 지점에 도달했다. 주요 콘텐츠를 삭제하지 않는 한 더 이상 메모리를 줄일 방법이 없었다. 모든 방법으로 노력해 본 다음 현재 메모리 사용량을 측정해 보았는데 여전히 1.5 MB 정도가 초과하고 있었다.

이런 상황에서 팀 내 가장 경험 많은 프로그래머 중 한 명(수년 간 개발 작업에 참여하였고 옛날에 잘 나가던 프로그래머임)이 직접 문제를 해결해 보겠다고 했다. 그가 사무실로 불렀고 나는 메모리를 줄이기 위한 힘든 시간이 또 시작될 것이라고 상상했다.

그러나 그는 한 소스 파일을 열고 다음 코드를 보여 주었다.

static char buffer[1024*1024*2];

“이거 보이지?”라고 하더니 키를 쳐서 이 줄을 지워 버렸다.

나의 놀란 눈을 보고는 개발 초기에 2MB 의 메모리를 만들어 두었다고 설명해 주었다. 그는 다수 경험을 통해 프로젝트 말기에 메모리를 줄이는 것은 거의 불가능하다는 것을 알고 있었으므로 항상 필요한 경우에 대비해 예비 메모리를 확보해 두었던 것이다.

그는 사무실 밖으로 걸어나가 제한된 예산 내에서 메모리 사용량을 줄이겠다고 말했고 프로젝트의 영웅 대접을 받게 되었다.

이러한 야만적인 행동에 놀란 것은 내가 아직 초보라서 그런 것인지도 모르겠다. 이상과 같은 행동에는 아직 동감이 가질 않지만 만일의 경우를 대비해 뭔가를 미리 준비해 두는 일은 중요하다고 생각한다. 시간과 경험이 모든 것을 변하게 만든다.

- *Noel Llopis*