1. ActiveX란?

ActiveX는 마이크로소프트에서 개발한 응용프로그램과 웹을 연동시키기 위해 제공되는 기술입니다. ActiveX는 웹에서 HTML의 정적인 웹 문서에서 탈피하여, 동적이고 화려한 멀티미디어 기술을 동작 할 수 있도록 하는 일종의 인터넷 익스플로러 전용 플러그인(Plug-in) 기술이라고 할 수 있습니다.

2. ActiveX 관련기술

ActiveX Control

웹 페이지에 내장되어 실행 가능한 객체로 C/C++. 비주얼베이직, 델파이등과 같은 다양한 언어와 개발툴로 작성 할 수 있습니다.

ActiveX Document

웹브라우저를 통해 볼 수있는 HTML과 무관하게 작성된 도큐먼트로, MS워드나 엑셀 파일 등을 의미합니다.

Active Scripting

액티브X 컨트롤이나 자바 애플릿에 포함시킬 수 있는 스크립트 언어로, 자바 스크 립트에 기반한 J스크립트나 비주얼 베이직에 기반한 VB스크립트가 대표적입니다.

ActiveX Server Framework

웹서버에 기반한 기능, 즉 보안이나 데이터베이스 연결을 가능하게 하는 서버 사이드 아키텍처 입니다.

3. 국내 Active X 이 들어오게 된 배경 및 맥락 칼럼

요즈음 ActiveX, 정확히는 'ActiveX 컨트롤'이란 기술이 시끄럽다. 브라우저 밑으로 손을 뻗어 그 밑에 깔린 시스템의 기능을 만지작거릴 수 있게 하는 요물. 웹은 웹이로되 PC가 할 수 있는 모든 일을 하게끔 하는, 웹을 웹 이상으로 조작하기 위한 '만능 컨트롤' 도구, ActiveX. 90년대의 프로그래머들은 ActiveX가 포함된 COM이라는 테크놀로지 조합으로 PC 전성기를 풍미했다.

그런데 새 버전의 인터넷 익스플로러와 새 OS 윈도우 비스타는 자신들의 기술 ActiveX를 유리 상자 안에 가둬 버리고 만다. ActiveX란 뭐든지 만들 수 있지만, 뭐든지 망칠 수도 있는 양날의 검이었다. 새 플랫폼이 ActiveX에 거리를 두는 이유는 '시스템의 기능을 만지작거리는 일'이 악인에 의해서도 자행될 수 있다는 자각 때문이다. ActiveX는 모두가 순박했던 목가적 시절에나 어울리는 기술이었던 것이다.

게다가 이미 업계는 웹을 임의로 '컨트롤'하여 변경하는 일이 그리 바람직한 일도 아님을 공감하고 있다. 웹 표준 운동도 그 일환이다. ActiveX같은 로우레벨 아키텍처에 의존한 플랫폼을 만드는 일이란 플래시 수준의 입지를 지닌 플랫폼 제공자가 아니라면 비즈니스적으로도 별 의미가 없다는 것을 깨달았다. 마치 고급 언어를 배운 이래 어셈블리어를 만질 필요가 없듯, 굳이 웹을 개선한다는 목적만으로는 ActiveX라는 위험한 칼을 만질 이유가 없는 것이다.

물론 아이디어란 표준으로 묶어 놓기에는 너무나 자유분방한 것이기에, 올해도 내년에도 웹의 확장은 일어날 것이다. 그렇기에 웹을 초월한 무언가를 덧붙이려는 확장 욕구는 건전한 것이다. 브라우저로 하지 못하는 일을 새로운 아이디어로 '확장'하려는 욕망은 멈추기 힘들고, 이 점을 강조하기 위해서 일까? 파이어폭스가 ActiveX '컨트롤(Controls)'을 금지하고 대신 파이어폭스 '확장(Extension)'이란 개념을 도입한 의도는 그 용어에 잘 나타나 있다. 마이크로소프트도 이미 닷넷을 중심으로 기술 구조를 재편한지 오래다. ActiveX를 위시한 Win32의 리거시 기술들은 배후로 밀려나고, 웹의 확장 기능도 ActiveX라는 칼을 직접 만지지 않도록 유도하고 있다. 더 편하고 더 쉬운 확장을 할 수 있는 방안과 로드맵이 따로 있는 것이다.

그러나 우리는 유난히 ActiveX라는 날카로운 칼을 좋아했다. 그리고 무척이나 잘 드는 이 칼로 웹을 확장한 것이 아니라 오히려 웹의 여기저기를 도려내며 우리만의 아키텍처를 만들었다. 대한민국의 웹을 서핑하다 만나게 되는 수 없는 경고창들, 칼을 조심하라는 시스템의 경고지만 개의치 않는다. 수저가 필요한 곳에 칼이 놓이고 있다. 손잡이가 필요한 곳에 날이 서 있다.

칼날이 난무한다. 특히 은행 일이라도 한번 보려면 여러 개의 컨트롤을 일단 깔아댄다. 뭐가 뭔지 도무지 모르겠지만, 여하튼 설치하지 않으면 아무 것도 못하니 방법이 없다. 게다가 왜 이렇게 회사마다 종류가 골고루인지. 그렇게 내 PC를 유린하듯 설치되는 컨트롤의 면모는 살펴 보니 하나 같이 '보안 모듈'.

여기에서 의문이 생긴다. 왜 보안을 웹의 외부 기능에 의존해야 하는 것인가? 사실을 말하자면, 한국 수준의 보안은 모르겠으나 적어도 세계 수준의 보안은 브라우저 만으로도 얼마든지 확보할 수 있다. 외국 굴지의 은행들은 브라우저만으로 인터넷 뱅킹을 무리 없이 수행하고 있다. IE와 파이어폭스 모두 필요 충분한 수준의 암호화 기능은 물론 인증서 관리 기능도 들어 있다.

그런데 한국은 세계에서 통용되는 이러한 표준 기능은 활용하지 않은 채, 보안을 웹의 외부 기능으로 빼내어 독자적으로 처리하고 있다. 놀라운 기술 독립이다. 마이크로소프트도 모질라 재단도 놀라고 있는 일이다. 그들은 이해를 못하는 일이다.

왜? 도대체 왜 이 상황이 된 것일까?

여러 가지 도시 전설이 횡행하지만, ① 당시 미국의 128비트 암호화 수출 금지 조항에 맞선 독자 기술(SEED)의 개발과 적용 지도, ② 한국의 특수 상황이 발생시킨 정보 기관의 지침(보안 적합성 검증), ③ 독자적 최상위 인증 기관 운영 욕구, ④ 해킹 피해 발생 보도에 대한 과민 반응. 등이 복합적으로 작용했다는 설이다. 인터넷이 너무 일찍 퍼진 한국은 너무 급했고 너무 불안했던 것이다.

이 과정에서 얻은 일도 있을 것이다. 내수 보안 산업이 자생적 생태계를 꾸릴 수 있었다. 척박한 국내 IT 시장에서 나름대로 고용을 창출하고 기술을 연마해 온 그들에게 과연 “당신들의 존재 자체가 틀렸어!”라고 감히 말할 수 있을까? 누구도 그럴 용기가 없다. 완전한 기술 쇄국을 이끈 정부도 금융권도 IT 업계도 국민도 어느 누구도.

그러나 잠시 스스로를 돌아 볼 때다. 우리는 정말 세계 어느 나라보다 안전할까? 인증서 파일을 PC에서 PC로 옮겨 들고 다니는 일이 과연 최고의 보안 솔루션일까? 다른 나라처럼 암호 발생 카드나 암호 발생 열쇠고리를 사용하는 것이 차라리 안전하지 않을까? 전세계적으로 테스트되고 사용되고 있는 브라우저 들의 내부 보안 기능보다, 버그가 있을 수 있는 개별 기업의 외부 보안 솔루션이 더 안전하다고 우리는 진정 믿을 수 있을까? 우리에게는 잠시 쉬어가며 백지에서 다시 생각해 볼 여유가 필요한 것이다.

ActiveX의 문제란 결국 독자 기술의 꿈이 불러 온 기술 쇄국의 딜레마였던 것이다.

사실 아무 일도 아닐 수도 있다. 쇄국의 아키텍처를 끝까지 고수하며 업체를 압박한다면 어떻게든 솔루션은 생길지 모른다. 그러나 언제까지 그렇게 아슬아슬한 아키텍처를 우리는 가져갈 수 있을까? 새로운 OS가 등장할 때마다, 새로운 브라우저가 등장할 때마다 우리는 '우리의 실정'을 부르짖어야 할 테니까.

기술은 도구인 이상, 양날의 검이다. 잘 쓰면 유용한 도구이지만 목적을 잊은 채 수없이 주머니에 품고 있기에는 거북한 존재인 것이다. 잘못 들어가 있는 칼은 서서히 걷어내야 한다. 그리고 그 칼의 사용은, 그리고 더군다나 민생에 직결되는 서비스에서의 사용은 더 신중히 논의되어야 하는 것이다.

칼을 드는 순간, 내 스스로 누군가를 소외시키지는 않는지, 그리고 그 칼을 드는 순간 내가 세상으로부터 소외되지는 않은지 생각해 봐야 한다. 도구의 의미를 생각하지 않은 채, 용도를 숙고하지 않은 채, 도구의 방향을 관찰하지 않은 채, 도구를 본래의 취지와 맞지 않게 남용하는 것이 얼마나 무모한지 우리 사회는 그리고 업계는 어쩌면 매우 비싼 값을 치르며 배우고 있는 것인지도 모른다.

3. Active X 가 한국에서만 사라지지 않는 이유

가장 큰 문제, 공인인증서, 개인에게 보안 문제 덮어씌우기

해킹등 금융사기 사건 발생시, 보안 문제의 책임이 개인에게 돌아간다는 것.

실제로 국내 여러 금융사기 사건들을 살펴보면
피해를 입은 개인이 해당 금액을 모두 보상 받거나 기업 측에서 피해를 책임 진 사례가 거의 없다고 한다.
금융회사의 입장에서 보면, 거래에서 공인인증서가 사용된 사실만 확인되면 문제가 생겨도 기업 책임은 아니라는 것이다.

국내의 금융거래 보안 기술은 돈을 보내는 사람(sender)의 신원확인에만 집착하고 있으나, 외국의 보안기술 경향은 돈을 받는 사람(beneficiary)의 신용도를 중요하게 고려하고 있음.
송금자의 신원(identity)만 확인되면, 나머지는 만사OK라는 식의 후진적 보안기술은 금융소비자의 피해로 직결되고 있음. 돈을 받는 사람의 신용도 체크를 당장에 할 수 있는 은행이 그것도 안하고 대포통장으로 마구 이체해놓고, 고객에게 책임을 떠넘기려는 것은 옳지 않음.

출처- 고려대학교 김기창 교수 인터뷰

4. 생각 및 정리

유저가 알아서 간수하라..라는 부담을 고객에게 준다.
금융회사의 이익을 위한 것이라는 결론.
보내는 사람의 신원이 중요한 것이 아니라, 어떤 패턴 과거를 가지고 해 왔는가.. 은행이 다 분석하는
외국의 보안트렌드가 진화해 왔다. 한국은 금융 소비자에게만 책임을 묻고 있다. otp나 공인인증서 activex로만 확인하면 금융회사 책임은 아니라는 것이다.
너무 후진적인 보안기술.. 정부가 특정한 보안기술을 만들어서 금융회사에게 책임을 물을 수 있도록 해야한다.
외국에서 사용되는 선진 보안 기술의 대표적인 예 - FDS (Fraud Detection System, 이상거래 탐지 시스템)
무엇을 설치하라는 것이 계속 나온다는 것은 매우 후진적인 보안방식. 15년 전에는 별다른 보안 아이디어가 없어서 대박 아이디어로 사용했지만, 인증서 재발급에 필요한 과정이 activeX때문에 허술하기 때문에 보이스피싱이 계속 이루어진다. 단순히 불편하다를 넘어서 매우 위험하다 라는 결론.

*선진국사례 - 금융회사(은행, 카드사) 책임

반면 선진국에서는 서버(은행, 카드사)측이 빅데이터 분석 및 인공지능(AI)을 기반으로한 FDS 솔루션을 운용함으로써 보안을 철통같이 유지하고 있습니다.
FDS 솔루션 자체가 서버쪽에서 클라이언트들의 거래 패턴을 분석해서 고객의 신원을 정확히 파악(인증, Authentication)해야 하는 개념이기 때문에, 만약 전자금융사고가 발생하면 '고객의 과실' 여부는 따지지도 않고 해당 금융업체가 고객에게 철저히 손해배상을 해주게 됩니다. < Zero-Liability Protection 정책 >

따라서 선진국의 소비자들은 "액티브 X" 와 같은 잡다한 프로그램들의 설치 없이, 웹 브라우저만으로도 안전하게 온라인 서비스를 이용할 수 있는 것.

5. 기술적 부채

1992년 Ward Cunningham이 비 기술자들에게 문제를 전달하기 위해 사용하기 시작한 단어로, 간단히 말해 꼭 해 두지 않아도 또는 프로젝트가 끝날 때 까지(심지어는 끝나고 나서도) 티가 잘 나지 않는 작업들을 지칭한다.

일반적으로 잘 알려진 기술적 부채로는 악취 나는 코드, 테스트 커버리지, 적절하지 못한 객체 모델링과 같은 작업들이 있다.

기술적 부채는 당장 눈 앞에 놓은 이익으로 인해 급한 불만 꺼나가는 방식의 잘못된 의사결정이 지속되어 누적된 결과이죠. 따라서 기술적 부채는 잘못된 의사결정의 결과라고 볼 수 있습니다

기술적 부채를 둘러싼 순환구조
Ward Cunningham이 부채라는 표현을 쓴 이유는 복리이자가 발생하기 때문으로 이 문제를 제대로 다루기 위해서는 문제를 순환구조로서 파악하지 않으면 안된다.

기술적 부채가 지닌 대표적인 특징은 빌린 사람과 갚는 사람이 일치 하지 않는다는 점인데, 빌린 사람은 최초 개발자에 해당하고 갚아야 하는 사람은 표면적으로는 유지 보수 개발자이고 본질적으로는 시스템 소유자(프로젝트 오너)라 볼 수 있다. 기술 부채는 소프트웨어를 사용하는 앤드유저에게는 직접적인 가치를 제공하지 않기때문에 관리업무에서 흔히 소외되는 요소중 하나로, 좀 더 직설적으로 이야기 하자면 코드 가독성을 높이기 위해 고객에게 릴리즈 일정을 조정해 달라고 할 수 없다는 뜻이다.

부채는 도덕적 해이 (morale hazard)를 불러오기 쉬운것처럼, 기술적 결함도 누적되어가면서 조직이 결함에 관대해지고 개발자들이 책임을 지지 않는 태도를 보이는 경우가 있습니다. 심각한 결함이 아니라면 그냥 방치해 버리거나 QA조직에서 사정을 해야만 손을 대는 것.

금융 부채와 마찬가지로 적절하게 컨트롤 되기만 한다면 기술적 부채가 꼭 나쁘기만 한 것은 아니다. 적절한 기술적 부채의 운용은 프로젝트 진행에 상당한 융통성을 가져다 주기 때문이다. 채무를 예로 들자면 단기 상환채무를 장기 상환채무로 옮긴다던가 보다 많은 이자가 발생하는 부채를 적은 이자가 발생하는 부채로 옮기는 것과 같은것이다.

하지만 이자는 어딘가에 몰아넣고 잊어버릴 수 있는것이 아니다. 시간과 함께 착실히 증가하게 되며 이를 제대로 인지하고 관리해 두지 않는 다면 프로젝트나 비지니스를 망치는 심각한 문제로 커 질 수도 있다. 이 점 때문에 기술적 부채는 이상과 현실 사이에서 영원히 고통 받는 프로젝트 매니저들에게 중요한 숙제거리이다.

누락된 자동화 테스트

제때에 작성해 두지 않은, 또는 부실하게 작성되어 실행 패턴에 대한 커버리지가 부족한 자동화 테스트는 대표적인 기술 부채이다. 멀쩡하게 동작 하는 것 처럼 보였던 코드들은 지름신 에게 휘둘린 후 애써 모른 척한 신용카드 청구서처럼 때가 되면 어김 없이 당신을 찾아와 비용 지불을 요구할 것이다. 버그의 처리 비용은 생성 후에 얼마나 빠른 시간 내에 처리 되느냐에 따라 달려 있다는 것은 이미 여러 연구에 의해 밝혀진 사실이다.

이에 반해 테스트 케이스를 작성하는 과정 자체는 테스트 대상이 되는 실행 코드에 대한 통찰력을 늘리는 데에도 많은 도움을 준다. 이것은 채무의 반대되는 의미로 기술 자산이라 불러도 좋을듯 하다.

자동화 테스트는 젠킨스와 같은 CI툴과 만나 더욱더 놀라운 시너지 효과를 낸다. 프로젝트의 누군가가 새로운 소스를 커밋 한다면 CI서버는 이를 감지하여 자동화 테스트를 실행하고 그 결과 최단 시간에 새로운 코드가 발생 시킬 수 있는 문제점들을 알려준다.

자동화된 테스트 코드의 가장 큰 장점은 코드가 사양 변경에 매우 견고해 진다는 점 이다. 제대로 작성된 테스트 코드를 가지고 있다면 추가되는 사양 변경에 대해 두려움 없이 작업을 할 수 있을것이다. 테스트 결과는 수정한 내용이 다른 코드들에 부작용 없이 잘 움직인다는 것을 보증해 줄 것이며 만약 예상치 못한 문제가 생기더라도 즉각 CI를 통해 이를 인지 하여 대처 할 수 있을 것이다.

성실하게 작성된 테스트 코드들은 부채가 아닌 착실하게 신뢰성이라는 이자를 불려가며 당신의 코드를 더욱더 가치있게 만들어 줄 것이다.
만약 프로젝트 일정중에 잠시잠시 여유시간이 생긴다면 언제나 자동화 테스트를 추가하고 또 추가하라. 언제나 녹색을 유지하는 테스트 결과 그래프는 당신의 작업을 더욱 즐겁게 만들어 줄 것이다.

결론

프로젝트 내부에서 기술 채무는 빨리 제거 하는 것 만이 능사는 아니다. 서론에서 기술한 바와 같이 기술적 채무는 개발자가 게을러서 발생하는 것이 아니며, 스케줄 진행상 거의 필수 불가결하게 발생하게 되므로 적절하게 관리만 된다면 프로젝트 진행에 상당한 융통성을 가져다 줄 수 있다. 물론 처음부터 기술 부채가 발생하지 않도록 노력 하는 것은 중요하지만 사안에 따라서는 프로젝트 이해 관계자들의 승인 하에 과감한 이월(유지 보수 관리자에게 떠 넘기기)을 선택 하거나 아예 디폴트를 선언하는것도 한가지 방법이 될 것이다. 디폴트 선언(냄새나는 코드 무시하기)에 대해서는 Eric Evans가 제안한 전략적 설계가 좋은 지침이 될 것이다.

기술 부채는 금융 부채와 마찬가지로 한꺼번에 상환 하기 어려운 경우가 대부분이기 때문에 프로젝트 매니저는 스케줄 관리에 기술 부채 상환에 대한 부분을 반영하고 전략적으로 유지 되도록 관심을 기울여야 할 것이다. 작업 관리에 Trac이나 Redmine과 같은 이슈 트랙커를 사용하고 있다면 이러한 기술 부채를 다루기 위한 별도 카테고리를 만들어 관리하는 것도 좋은 방법이다.