[특집 2 _ Part 2]

개발자 입장에서 본 개발툴 연대기

지배할 것인가, 지배될 것인가

이제 개발자에게 필수적인 동반자로 자리잡은 개발툴. 끊임없이 새로운 기술들을 빨아들이며 업계의 다른 어떤 기술들보다도 빠르게 발전하고 있는 개발툴들이 개발자에게 지금 어떤 의미이며, 개발툴 시장에 영향을 미치고 있는 주변 요인들은 어떤 것이 있나. 과연 개발자들은 개발툴을 제대로 지배하고 있는가.

박지훈 **cbuilder1@borlandforum.com**

개발과 가족 외에는 커뮤니티 운영 밖에 모르는 자칭 미련한 개발자. 취미를 묻는 질문에 무심코 개발자 커뮤니티라고 말했다가 인생의 회의를 느끼기도. 현재 듀오에서 개발 책임을 맡고 있으며 '취미로' 개발자 커뮤니티 볼랜드포럼을 운영하고 있다.

1983년에 최초의 상용 **IDE(**통합개발환경**)**인 터보 파스칼이 출시된 이후로, 개발툴들은 컴맨드 라인 컴파일러라는 원시적인 수준을 벗어나 눈부시게 발전해왔다. 이전에는 **IDE**없이 어떻게 개발을 했을까 싶을 정도다. **IDE**의 등장은 일부 천재적인 프로그래머들의 전유물이자 괴롭고 반복적이었던 소프트웨어 개발을 효율적이고 즐겁기까지 한 작업으로 바꿔놓았다. 그러나 동시에 개발자들에게 생산성**(productivity)**의 중요성을 현실적으로 부각시키기도 했다. **IDE**의 등장 이후에야 업계에서 개발자 생산성이 진지하게 받아들여지기 시작한 것이다. 소프트웨어 개발의 제 **2**막은 **IDE**의 등장으로부터 시작됐다.

90년대 초반의 **RAD**에서부터 최근의 **ALM(Application Lifecycle Management)**에 이르기까지, **IDE** 이후에 등장한 모든 굵직굵직한 개발툴 관련 이슈들은 크게 보면 **IDE** 발전사의 연장선상에 있다고 할 수 있다. 에디터와 디버거, 온라인 헬프 등이 통합되어 있었던 **IDE** 초기에는 이 기능들이 개발자에게 큰 부담이 되지 않았다. 하지만 그 이후로 **IDE**는 점점 더 많은 개발 관련 기능들과 툴들을 스폰지처럼 빨아들이면서 엄청나게 비대해졌고 그만큼 개발자들이 익혀야 할 기술과 테크닉도 늘어났다. 그러면 **IDE**의 발전 과정 동안 방대해진 기술 백서만큼 그에 비례하여 개발자들의 대우나 만족도는 더 높아졌을까**?** 바로 이것이 이 글의 화두다.

툴의 발전 vs. 개발자의 대우

컴파일러라고 하지 않고 개발툴이라고 말할 때 일반적으로 의미하는 것은 **IDE**이다. 만약 통합된 개발툴을 생각하지 않고 컴파일러라고만 생각한다면 컴파일러간의 차이는 표준 호환성이나 성능**(**최적화 등**),** 버그 수준 등 주로 품질과 관련된 문제들이 우선된다.

하지만 기반의 컴파일러 코어와 분리하여 바라보면 개발툴은 프로그래밍 언어와 컴파일러 위에 얹혀진 또 하나의 레이어로 볼 수 있다. 프로그래밍 언어가 표준화되어 온 동안 개발툴은 경쟁관계에서 서로의 장단점을 수용하면서도 차별화의 길을 지향해 온 것이다. 실제로 최근에는 아예 컴파일러를 포함하지 않은 개발툴도 종종 볼 수 있다. 자바 개발툴과 마이크로소프트**(**이하 **MS)**의 비주얼 스튜디오 닷넷은 그 자체로는 컴파일러 코어를 가지고 있지 않으며 컴파일러는 **JDK**와 **.NET** 프레임워크에 포함되어 있다. 볼랜드도 **C++** 크로스 플랫폼 개발을 위해 **C++**빌더**X**를 내놓은 바 있는데 이것은 애플리케이션의 타겟 플랫폼 설정에 따라 다양한 컴파일러를 불러 들여 컴파일을 한다.

그러면 왜 컴파일러 자체가 아닌 통합된 개발툴을 사용하는 것일까. 이것은 두말할 것도 없이 생산성 때문이다. 코딩을 위해 최적화된 에디터와**,** 에러 위치를 에디터에서 바로 하이라이트 해주는 컴파일러**,** 실행중인 코드를 따라가면서 상태를 볼 수 있게 해주는 디버거**, F1** 키만 누르면 바로 나타나는 컨텍스트 센시티브 온라인 헬프**,** 이 모든 것이 개발 작업의 생산성을 높여준다. 또한 **GUI RAD** 기능을 갖춘 개발툴에서는 화면 구성을 위한 **GUI** 작업까지도 비주얼하게 할 수 있어 개발 작업을 단순화시켜주며 더 효율적인 자체 프레임워크를 내장하기도 한다.

이처럼 개발툴의 지상 목표는 생산성이다. 프로그래밍 언어와 컴파일러 코어가 성능**,** 호환성**,** 재사용성 등 여러 가지 복잡한 목적을 위해 발전해왔다면 개발툴은 오로지 생산성을 높이기 위해서만 발전해 온 것이다.

혹자는 명필은 붓을 가리지 않고 명장은 도구를 가리지 않는다고 말하면서 개발툴의 자체 가치를 폄하하기도 한다. 하지만 특급 호텔의 요리사라고 하더라도 라면 양은 냄비 하나로 만든 프랑스식 스테이크로 미식가 고객을 만족시킬 수는 없을 것이며**,** 시간도 더 많이 걸릴 수밖에 없다. 양은 냄비 하나로도 요리를 할 수 있고 윈도우 메모장이나 유닉스의 **vi**만으로도 코딩은 가능하지만 문제는 소요 시간**,** 즉 생산성이다. 그리고 설사 그런 명제가 절대적인 진실이라고 하더라도**,** 아직 명장 혹은 흔히 말하는 고수가 되지 못한 다수의 개발자들은 어떻게 해야 하는가.

물론 도구를 가리지 않는 명장도 있을 수 있다. 하지만 항상 치열한 경쟁이 벌어지고 있는 **SW** 업계의 현실에서 남보다 더 빠르게 결과를 내놓는 것은 항상 최우선은 아니라 할지라도 높이 평가받는 요소이며**,** 대부분의 경우에는 필드 개발에서 속도와 생산성은 가장 핵심적인 요소 중 하나다. **SW** 업계의 필드는 학교도**,** 연구기관도 아니며 예술계는 더더욱 아니다. 촛불 끄고 붓글씨를 쓰는 사이에 레이저 프린터로 찍어내 버리면 승부는 끝난다.

물론 개발툴로 인한 생산성의 가치를 높이 평가한다고 해서 생산성 이외의 요소들의 가치가 떨어지는 것은 아니다. 양극단은 언제나 위험한 법이다. 또 투박한 도구로도 뛰어난 제품을 만들 명장이라면 당연히 더 인정받는다.

생산성은 1차적으로 개발 작업의 효율성과 관계되지만 프로젝트와는 별개로 개발자의 실생활에도 직접적인 영향을 끼친다. 생산성이 떨어지는 개발툴을 사용한다면 다른 개발자들이 하루면 끝낼 수 있는 일을 며칠씩 붙잡고 노가다에 가까운 작업을 해야 한다. 따라서 프로그래밍 언어와 컴파일러 레벨보다는 개발자에게 더욱 많은 영향을 끼치게 된다. 게다가 개발툴은 필연적으로 수많은 기능을 정리해놓기 위한 복잡한 메뉴 시스템을 가지고 있으며, 그만큼 많은 단축키들도 내장하고 있다.

이 때문에 개발툴은 개발자에게 있어 익히기도 쉽지 않지만 주력 개발툴을 바꾸기도 어렵게 한다. 같은 윈도우 플랫폼을 위한 C++ 개발툴이라도 C++빌더의 GUI 빌딩 기능에 익숙해진 개발자는 비주얼 C++로 바꾸기 어려우며 반대로 비주얼 C++의 코드 에디팅 편의 기능에 익숙해진 개발자는 C++빌더의 부족한 편의 기능에 불평하기 마련이다. 반면, .NET 프레임워크에서의 여러 언어들의 경우나, 쌍둥이 개발툴이라고 할 수 있는 C++빌더/델파이처럼, 개발툴보다 하위 레이어라고 할 수 있는 프로그래밍 언어가 다른 경우라도 개발툴이 유사하고 핵심 프레임워크를 공유하는 경우라면 비교적 쉽게 이동할 수 있다. 따라서 어떤 측면에서는 프로그래밍 언어와 컴파일러보다 핵심에 덜 가까워 보이는 개발툴이 개발자의 생활을 더 강하게 제약하게 된다. 주로는 긍정적인 면에서, 때로는 오히려 부정적인 면에서 말이다.

대혼란기

개발자에게 현재의 SW 업계는 대단히 혼란스럽다. 윈도우와 리눅스가 경쟁하고 닷넷과 자바가 자웅을 겨루고 있다. 프로그래밍 언어에서는 C++과 자바의 양대 산맥에 C#이 뛰어들어 일정한 지위를 노리고 있다. 윈도우 안에서도 MS는 .NET이 대세라고 외치는데 개발자들의 다수는 아직 Win32에 머물러 있고 윈도우 개발툴 시장에서는 MS와 볼랜드의 승부가 아직도 진행 중이다.

자바쪽에서는 자바의 본가라고 할 수 있는 썬과 실질적인 왕좌를 노리는 IBM, 그리고 BEA까지 주도권 다툼을 벌이고 있으며 똑같은 오픈소스 개발툴인 이클립스와 넷빈즈가 각각 IBM과 썬의 영향으로 대결 중이다. 금방이라도 윈도우의 목을 조를 것 같았던 리눅스에서는 SCO가 시작한 저작권 분쟁이 리눅스 확산의 발목을 잡고 있어 현실에서 데스크탑 리눅스는 수년째 제자리걸음인 것으로 보인다.

아마도 개발자에게 IT 역사상 이렇게 혼란스러운 시기는 없었을 것이다. IT의 폭발적인 성장기였던 80년대와 90년대에 PC 레벨에서는 도스와 윈도우가 세상을 장악하고 있었으며 엔터프라이즈 쪽에서는 메인프레임/유닉스와 C/C++이 거의 유일한 선택이었다. 리눅스는 몇몇 애호가들이 취미 삼아 즐겼던 재미거리였으며 자바는 95년에 등장해 90년대 후반에 이르기까지 제대로 관심을 받지 못했다. 리눅스와 자바, 닷넷이 등장하고 관심을 받으면서 지금 대세이면서 앞으로 대세를 이어갈 것이 어떤 것이라고 말하기는 너무나 어려워졌다.

이렇게 복잡하고 혼란스러운 상황에서, 무엇이 대세인지 앞으로 개발자는 어떻게 생존의 길을 찾아야 할지를 한눈에 알아보기는 대단히 어렵다. 현업의 개발자들도 앞으로 어떻게 나아가야 할 지 혼란스러운데 이제 SW 업계로 진입할 꿈을 꾸고 있는 예비 개발자들에게는 차라리 고통에 가까울 것이다. 2002년의 가장 큰 이슈들 중 하나가 닷넷이었고 금방이라도 자바를 따라잡고 신 시장을

열어줄 것 같았지만 아직 시장에서 그다지 호응을 받지 못하고 있다. 2002년의 열풍을 타고 닷넷에 올인했던 대학생들이 C++이나 자바로 전향했다는 얘기도 심심찮게 들린다.

개발자와 이 글의 주제인 개발툴은 이 모든 복잡한 혼란의 중심에 있다. 개발툴은 프로그래밍 언어보다 개발자에게 더 가까이 있기에, 또한 개발자는 애플리케이션을 생산하는 주체이기에, 그리고 애플리케이션이 플랫폼을 대세로 만들어주는 지름길이기에 이 업계의 모든 세력들은 개발자들을 끌어들이려고 혈안이 되어있다. 그만큼 이 업계의 다른 구성원보다 개발자들이 느끼는 혼란은 더욱 클 수밖에 없다. 그래서 당장의 세세한 하나 하나의 사건들이 아니라 전체가 어떻게 돌아가는지를 바라볼 수 있는 냉정한 관전법이 필요하다.

혼란의 중심에 선 개발자

이 글의 주제인 개발툴을 우선적으로 고려하여 개발자의 시각에서 보면, 가장 간단하면서도 명쾌한 관전법은 윈도우 vs. 자바다. 물론 이런 구분에 대한 이견의 여지는 충분하다. 윈도우와 자바 이외에 리눅스나 유닉스, 매킨토시 등 다른 플랫폼들이 있고 또 나누는 기준도 상용 개발툴과 오픈소스(혹은 무료) 개발툴로 나눌 수도 있다. Win32와 닷넷을 나누는 것도 의미가 있다. 혹자는 Win32 개발자는 어차피 닷넷으로 이전할 것이므로 '윈도우 vs. 자바'가 아니라 '닷넷 vs. 자바'가 더 의미가 있다고 주장할 지도 모른다.

하지만 이렇게 윈도우와 자바로 나누는 이유는 무엇보다 윈도우와 자바 플랫폼이 현재 개발자들의 절대 다수를 나누는 경계선이기 때문이며, 두번째는 윈도우와 자바 플랫폼에 비해 다른 플랫폼쪽에는 개발툴이 크게 발전하지 못했기 때문이다(따져보면 이 두 가지 원인은 서로 밀접한 연관이 있다). 예를 들어 무시할 수 없는 중요한 플랫폼인 리눅스의 경우 세계적으로도 윈도우, 자바에 비해 개발자가 적기도 하고 특히 국내에서는 PHP 개발자를 제외하면 리눅스 애플케이션을 개발하는 개발자는 극히 드물다. 또한 리눅스 플랫폼에는 볼랜드의 카일릭스(윈도우의 델파이/C++빌더와 소스 레벨 호환이 가능하다)가 RAD 개발툴로 호평받고 있지만 리눅스 기반 개발이 본격적으로 탄력을 받지 못하고 있는 등 여러 가지 이유로 2003년부터 버전업이 중단된 상태이다. 또 PHP는 LAMP라는 스택 형태의 개발 플랫폼이 세력을 키우고 있지만 아직 다수 개발자들을 끌어들이는 본격적인 상승세에는 이르지 못하고 있다.

사실 다른 플랫폼에 비해 윈도우와 자바 플랫폼이 유난히 세력을 얻고 있는 것은 두 가지 큰 이유가 있다. 하나는 기술 리더십과 자본을 가진 거대 IT 기업들의 영향력이며 다른 하나는 개발자들의 적극적인 호응 혹은 참여다. 윈도우의 경우 MS라는 거대 기업의 영향력에 개발자들이 호응한 경우이며 자바는 처음 등장할 때부터 개발자들이 주도적으로 붐을 일으킨 이후 많은 IT 기업들의 앞을 다투어 지원을 강화하고 있다.

그러면 왜 닷넷이 아니라 윈도우인가. 잘 알려진 것처럼 닷넷은 2000년 6월에 계획이 발표되고 2001년에 닷넷 프레임워크 1.0과 비주얼스튜디오 닷넷의 베타를 내놓음으로써 실체화됐다. 그 이후로 2002년에 비주얼스튜디오닷넷 정식 버전을 출시하고 2003년에도 각각 업데이트, 그리고 이번 달에는 그 최신 버전인 비주얼스튜디오닷넷 2005가 선보이는 등 숨가쁜 행보를 보여 왔다.

하지만 **MS**의 전례 없이 대대적인 홍보와 지원 공세에도 불구하고 현재까지 닷넷은 메인 타겟인 엔터프라이즈 시장의 진입부에 머물렀을 뿐, 윈도우 기반 개발자들의 대다수는 아직 기존 **Win32** 환경에서 개발하고 있으며 자바 개발자들은 요지부동이다.

ALM, 논란의 핵심

이런 업계 자체의 혼란에 대한 시각을 정리하는 것과 함께, 개발툴에 있어 개발자들이 받아들여야 하는 것은 개발툴에 있어 최근 들어 새롭게 도입되는 기술 추세들이다. 대표적인 것이 **ALM(Application Lifecycle Management)**이다. 개발툴 발전사에 있어 가장 최근의 경향인 이것은 이름 그대로 애플리케이션 개발의 전체 주기를 통합 관리한다는 의미다.

이를 위해 당연히 개발 주기 전반에 필요한 각 툴의 유기적인 통합이 필요하다. **ALM**은 볼랜드가 개념 정의에서부터 제품 출시까지 주도하고 있는데 최신 버전의 **J**빌더 **2006**이나 볼랜드 디벨로퍼 스튜디오 **2006**의 경우 개발툴을 중심으로 이들 개발 주기 툴들이 모두 통합되어 있다. 볼랜드의 투게더와 **UML** 모델링 툴의 쌍벽을 이루는 래셔널 로즈를 인수한 **IBM**도 이러한 포트폴리오를 갖춰나가고 있으며, 이 달 발표 예정인 **MS**의 비주얼스튜디오닷넷 **2005**도 모델링 기능을 추가하면서 이런 흐름에 뒤따르고 있다.

개발툴에 **ALM** 구현이 강화되면 될수록 개발툴은 그만큼 복잡하고 무거워질 수밖에 없다. 개발툴이 무거워지는 거야 하드웨어인 머신이 책임질 문제라고 해도 개발자의 입장에서 공부하고 익혀야 할 주제들이 엄청나게 많아지는 것은 그만큼의 부담일 수 있다.

하지만 **ALM**이 구현된 개발툴이 제공하는 것은, 그 모든 광범위한 기능들을 한 사람 한 사람의 개발자가 모두 익혀서 활용하기를 기대하는 것은 아니다. 각 파트의 전문가들이 분화된 개발팀에서 하나의 유기적으로 통합된 툴 셋을 사용함으로써 담당자간의 협력과 소통이 쉬워지고 각 작업들이 연동될 수 있도록 하는 것이다. 사실 한두 사람의 개발자가 진행할 수 있는 정도의 소규모 프로젝트에서라면 **ALM**은 그다지 메리트가 없다고 할 수 있을 것이다.

그럼에도 **ALM**이 개발자들에게 중요한 의미를 가지는 것은, 개발자 역할의 분화가 가속화되어 일반적으로 코더에 가까운 의미였던 개발자가 모델러, 아키텍트, 테스터 등으로 구분되거나 혹은 개발자라는 개념 자체에 변화를 몰고 오기 때문이다. 코딩에 대해 전혀 모르지는 않더라도 전문적이지 않은 모델러가 개발자의 범주로 편입되고 있으며, **1**차 산출물을 무작정 실행해보고 에러를 발견하는 것이 아니라 코더가 사용하는 것과 같은 개발툴에 통합된 테스팅 툴에 전문화된 전문 테스터가 등장하게 된다. 이 과정에서 개발자가 겪을 혼란도 만만치 않을 것이다.

일부 개발자들이 자동화된 개발툴에 대해 거리를 두는 이유들 중 하나는 툴이 개발자를 대신해 코딩을 하는 것처럼 보이기 때문이다. 최근에 **ALM**의 일환으로 **UML** 모델링 툴이 개발툴에 통합되면서 이런 느낌은 더욱 강해진다. 특히 볼랜드의 투게더의 경우 모델로부터 코드를 생성하는 기능뿐만 아니라 역으로 코드로부터 모델을 실시간으로 생성해주는 리버스 엔지니어링 기능까지 실시간으로 제공된다. 이런 지능화된 개발툴의 출현이 개발자들을 위축시키는 한 요인이 되는 것은 사실이다. 하지만 본질적으로 도구는 도구일 뿐 사람을 대신하지는 못한다. 툴이 하는 것은 단지

(어떤 관점에서든) 생산성을 높여주는 것이다. UML 모델이 필요한 모든 코드를 만들어줄 수는 없기 때문에 투게더에서도 이런 문제 때문에 모델이 만들어 낸 코드와 직접 추가한 코드를 구분해 관리한다.

어쨌든 이처럼 끊임없이 쏟아지는 개발 관련 신기술들에 대해 적지않은 개발자들이 피로를 느끼고 있는 것이 현실이다. 하지만 개발툴은 더욱 더 강력해져야 하고 더욱 더 많은 기능들을 흡수해야 한다. 그것은 개발자의 선택의 자유와 관련된 문제이기 때문이다. 개발툴에 어떤 새로운 기능이 등장하든 그것을 채택하고 하지 않는 것은 개발자의 몫이다.

물론 여기에는 개발자가 SW 업계의 주도적인 위치인가 하는 문제가 관련되어 있다. 개발자가 개발자로서의 주체적인 자신을 발견하지 못한다면 새로운 기술은 한낱 쓰레기에 불과하거나 혹은 반대로 등장하는 종종 흡수해야 한다는 강박관념만을 안겨주는 애물단지에 불과할 수 있을 것이다.

플랫폼에 편입되는 개발툴

먼저 앞으로 진행할 논의를 위해 몇 가지 기본적인 단어들을 정의해놓고 넘어가자. SW는 크게 플랫폼(platform)과 애플리케이션(application)으로 구분할 수 있다. 물론 절대적인 것은 아니고 SW의 상호 관계에서 생기는 상대적인 개념이다. 플랫폼이란 다른 소프트웨어를 그 위에서 동작하게 하는 바탕이 되는 소프트웨어/하드웨어를 말한다. 이 플랫폼 위에서 동작하는 SW가 애플리케이션, 즉 그대로 번역하자면 응용 SW이다.

대표적인 플랫폼은 운영체제(OS)로 윈도우 플랫폼에서 동작하는 SW를 윈도우 애플리케이션이라고 한다. 하지만 플랫폼이 OS뿐인 것은 아니다. 잘 알려진 자바나 닷넷도 플랫폼이고 스토어드 프로시저나 클라이언트 프로그램을 운영하는 DBMS도 플랫폼이라고 할 수 있다. 비교적 최근에 등장한 WAS(Web Application Server)도 의심할 필요 없이 플랫폼이다. 플랫폼은 반드시 SW만을 일컫는 것은 아니어서 하드웨어도 그 위에 동작하는 SW에 대해 플랫폼이다. 인텔 플랫폼, SPARC 플랫폼처럼 말이다.

또, 하나의 플랫폼이 다른 더 저수준의 플랫폼의 애플리케이션이 되는 중첩된 구조도 흔하다. 닷넷 플랫폼은 윈도우 플랫폼의 관점에서는 애플리케이션이며 윈도우는 인텔 플랫폼에 대해 애플리케이션이다. 자바는 윈도우와 리눅스, 솔라리스 등 여러 OS 플랫폼에 대한 애플리케이션이면서 자바 애플리케이션에 대해서는 플랫폼이다. 그리고 특정 애플리케이션을 하나의 플랫폼에서 다른 플랫폼으로 옮기는 작업을 포팅(porting) 혹은 마이그레이션(migration)이라고 한다.

플랫폼이 의미가 있는 것은 애플리케이션 때문이다. 그 정의상 플랫폼은 애플리케이션을 구동시키기 위한 바탕일 뿐 그 자체로서 의미를 가지는 것이 아니기 때문에 기본적으로 애플리케이션 없이는 무의미하다. 물론 실제로는 많은 플랫폼이 텍스트 에디터나 기본적인 애플리케이션 몇 가지 정도는 기본적으로 번들하고 있고 더욱이 최근 수년간 윈도우에서 보다시피 미디어 플레이어나 메신저 등 점점 더 많은 애플리케이션들을 기본 번들하면서 조금쯤은 의미가 희석되기도 했지만 어떤 플랫폼도 IT 고객이 원하는 모든 애플리케이션을 모두 번들할 수는 없기 때문에 플랫폼에 있어 애플리케이션의 중요도는 아직도 절대적이다. 원론적으로 말할 때 IT 고객이 필요로 하는 것은

플랫폼이 아니라 애플리케이션이다.

어쩌면 당연한 개념을 장황하게 설명한 것은 개발툴, 더 나아가서 개발자의 위치를 정의함에 있어 플랫폼과의 관계를 빼고는 논할 수가 없기 때문이다. 현재 우리가 사용할 수 있는 개발툴 중에서 기억하는 개발툴의 목록을 작성해보라. 그것이 몇 개가 되든, 그 안에서 다시 플랫폼 벤더들이 판매하는 개발툴의 개수는 몇 개나 되는지 확인해보자. 놀랍게도 몇몇을 제외하면 대부분이 플랫폼 벤더의 제품이다. **MS,** 오라클, **IBM,** 썬, **BEA,** 인텔, 사이베이스 등 이들은 모두 플랫폼 벤더다. 플랫폼 벤더가 아닌 벤더는 수십 년간 개발툴 전문 벤더로 살아남은 볼랜드나 코드워리어로 알려진 메트로웍스 정도 뿐이다. 역으로 생각해보면 **SW** 시장에서 독자적인 영향력을 가지고 있는 플랫폼 벤더치고 개발툴을 판매하지 않는 업체는 거의 없다. 자체적으로 개발할 능력이 모자라면 오라클이 **J**빌더 소스를 볼랜드로부터 사들여 **J**디벨로퍼를 만들었던 것처럼 외부로부터 사들여서라도 어떻게든 개발툴을 자사의 포트폴리오에 포함시킨다.

개발툴과 개발자의 특수한 위치

그렇다면 왜 플랫폼 벤더들이 개발툴에 눈독을 들이는 것일까. 개발툴이 플랫폼 시장과 비교할 만큼 의미가 있는 것일까. 물론 개발툴 시장이 매우 작다. 적어도 주력인 플랫폼에 비하면 미미한 수준이다. **MS** 입장에서 비주얼스튜디오 제품의 매출을 마이크로소프트의 주력인 윈도우나 오피스 시장의 규모와 비교하는 것은 의미가 없다. 그럼에도 불구하고 플랫폼 벤더들이 개발툴이 관심을 갖는 것은 개발툴이 플랫폼과 애플리케이션 사이에서 가지는 특수한 위치 때문이다. 앞에서 말한 것처럼 어떤 플랫폼도 애플리케이션 없이는 무의미하다. 플랫폼이 고객이 원하는 수많은 요구사항을 만족시키는 것은 불가능하기 때문에 서드 파티 애플리케이션은 필수적으로 필요하고 또 많으면 많을수록 좋다.

이 시점에서 개발툴이 정의된다. 개발툴은 애플리케이션을 만들기 위한 도구다. 서드 파티 기업에서 특정 플랫폼을 지원하는 애플리케이션을 개발하는 통로가 개발툴과 개발자다. 따라서 플랫폼 벤더의 입장에서는 다음과 같은 너무나 단순한 공식이 성립된다.

□ 플랫폼 확산이 목표

□ 플랫폼을 지원하는 많은 애플리케이션 필요

□ 플랫폼을 지원할 업체 및 개발자 필요

□ 플랫폼을 지원하는 개발툴 필요

더구나 편하고 더 강력한 개발툴을 내놓아야 더 많은 지원 업체와 개발자들을 끌어들일 수 있다. 모든 유력 플랫폼 벤더들이 개발툴 제품에 힘을 쏟는 이유는 이 때문이다. 개발툴 자체로서의

시장에는 그다지 관심이 없지만 개발툴이 직접적으로 일으키는 파급효과에는 민감할 수밖에 없다. MS가 개발툴 자체의 가격에 비해 턱도 없이 엄청난 재정적 지원을 개발툴과 개발자에 쏟아 붓고 썬과 IBM이 자사가 판매하던 개발툴의 전부 혹은 일부를 오픈소스 개발툴로 내놓는 이면에는 이처럼 명백한 이유가 있다.

그렇다고 개발툴을 내놓는 업체가 플랫폼 벤더뿐인 것은 아니다. 한때 개발툴만을 전문으로 판매하는 전문 개발툴 벤더들의 전성기가 있었다. 퍼스널 컴퓨터가 보급되기 시작하던 70년대 말부터 PC와 DOS의 전성기였던 80~90년대 초까지 개발툴(혹은 컴파일러)를 주력으로 삼았던 많은 SW 벤더들이 우후죽순처럼 생겨났다. 83년에 터보 파스칼을 내놓으며 설립된 볼랜드가 대표적인 성공 케이스라면 Watcom C++로 유명한 Watcom이나 Clipper로 한 시대를 풍미하고 이후에 CA로 합병된 낸터킷도 있었다. 91년에 파워빌더를 내놓았던 파워소프트도 그 중의 하나다.

이들 개발툴 전문 벤더들이 하나 둘씩 사라지기 시작한 것은 91년과 92년 사이에 있었던 큰 사건 하나가 계기가 됐다. 바로 MS 윈도우와 IBM OS/2 간의 PC용 OS 주도권 전쟁이었다. 잘 알려져 있는 것처럼 MS와 IBM은 80년대부터 공동으로 차세대 PC OS를 개발하는 프로젝트를 진행 중이었다. 80년대 중반에 MS는 이 프로젝트와는 별개로 윈도우 개발을 시작했고 IBM 주도의 공동 프로젝트로부터 손을 뗐다. 따라서 윈도우와 OS/2는 기술적으로 한 뿌리였다고도 할 수 있다. 하지만 OS의 커널에 관련된 핵심 기술은 IBM이 주도적이었기 때문에 90년의 윈도우 3.0과 91년의 윈도우 3.1은 대단히 불안하고 자원도 많이 소모하는 OS(사실 도스에 의존하는 '운영환경'이었지만)였다. 당시만 해도 MS는 MS-DOS를 팔아 떼돈을 번 졸부에 가까웠기 때문에 수십 년간 IT 맹주였던 IBM에는 기술적 완성도 면에서나 자금 면에서나 경쟁 상대가 되지 않았다.

하지만 OS/2의 승리를 점쳤던 많은 전문가들이 무색하게도 이 경쟁은 불과 1~2년 만에 MS 윈도우의 완승으로 싱겁게 끝나버렸다. 그 사이 MS 오피스를 비롯한 윈도우를 위한 다양한 애플리케이션들이 쏟아져 나온 반면, IBM의 OS/2를 지원하는 애플리케이션은 손에 꼽을 정도였다. 대부분의 사용자들이 외면할 수 밖에 없었던 것이다. 물론 이런 많은 윈도우 지원 애플리케이션들 중 오피스는 MS 제품이었지만 그 외 절대 다수의 나머지 애플리케이션들은 서드 파티에서 나온 것들이었다.

IBM을 누른 MS의 승리비결, 개발툴

이렇게 지원 애플리케이션 숫자에서 엄청난 차이를 가져온 원인은 여러 가지가 있겠지만, 가장 핵심적인 원인들 중 하나가 바로 91년에 첫 버전이 출시되었던 비주얼 베이직이었다. 베이직은 MS의 고향과 같은 프로그래밍 언어다(파스칼이 볼랜드의 고향인 것과 마찬가지다). 75년에 MS가 설립되면서 처음으로 내놓았던 제품이 알테어용 베이직 인터프리터(빌 게이츠가 직접 개발한 처음이자 마지막 SW로 알려져 있다)였으며, 그 이후로도 DOS에 GW-BASIC과 QBASIC을 계속 번들하는 한편 베이직 컴파일러인 Quick Basic을 별도의 제품으로 팔기도 했다. 비주얼 베이직은 Quick Basic의 계보를 잇는 개밭툴이다.

비주얼 베이직의 가장 큰 특징은 두 가지로 요약할 수 있다. 최초로 실용적으로 도입한 GUI 빌더

통합 개발툴이었으며 또 한 가지는 당시의 C++ 개발툴과 달리 윈도우 프로그래밍의 복잡한 과정을 아예 숨겨버렸다는 것이다. 이러한 특성들은 윈도우 개발을 대단히 빠르고 단순하게 만들었다. BASIC이라는 언어 자체의 '배우기 쉽다'는 특성도 이런 과정에 한몫을 했다. 적어도 비주얼 베이직이 없었다면 윈도우가 OS/2를 그렇게 단기간에, 그렇게 허무하도록 쉽게 물리치지는 못했을 것이라는 데는 이견이 없을 것이다. 그래서 이 사건은 개발자들을 끌어들이는 개발툴이 플랫폼 확산에 얼마나 필수적인지에 대한 대표적인 사례로 남게 되었다.

이 사건 이후로 플랫폼 벤더들의 개발툴 전문업체 인수 합병은 하나의 경향이 되었다. MS는 91년에 FoxPro로 잘 알려진 Fox Soft를 인수했고, 사이베이스는 94년에 이미 Watcom을 인수했던 파워소프트를 인수했다. BEA는 99년에 시만텍으로부터 비주얼 카페를 인수해 WebGain이라는 자회사를 만들었다. 오라클은 볼랜드로부터 J빌더 소스코드를 라이선스하여 J디벨로퍼를 내놓았고 썬은 2000년에 자바 개발툴인 넷빈즈(NetBeans)를 인수했다. 이런 인수 합병의 사례는 일일이 꼽기도 어려울 정도로 많으며 지금도 이런 인수합병의 행렬은 계속되고 있다. 가장 가까운 사례로는 지난 9월에 BEA가 또 다른 개발툴 회사 M7을 인수하기도 했다.

한 차례 인수 합병의 회오리 바람이 몰아친 후 살아남은 개발툴 전문 벤더는 볼랜드 정도였다. 볼랜드가 살아남을 수 있었던 이유는 인수하기에는 덩치가 너무 커서 재정적으로 부담이 되고(실제로 코렐이 인수하려다가 발표 직후의 주가 폭락 때문에 철회한 바 있다), 윈도우와 자바, 리눅스 등 경쟁관계에 있는 여러 플랫폼을 동시에 지원하는 제품 포트폴리오 때문이다(IBM이 래셔널을 합병한 직후에 MS가 볼랜드를 인수하려고 고려하다가 상용 자바 개발툴 점유율 1위인 J빌더의 존재 때문에 접었다는 소문이 파다했었다).

결국 현재에 있어서는 개발툴을 주력으로 하는 업체는 볼랜드 외에는 극소수만이 남았을 뿐이다. 개발툴 전문 벤더라는 말보다는 독립(independent) 개발툴 벤더라는 말이 일반적이 될 정도로 플랫폼 벤더 주도의 개발툴 시장이 일반적인 인식이 되었다.

플랫폼 밴더 주도의 개발툴 시장 그리고 개발자

하지만 이런 현재의 개발툴 시장이 과연 개발툴 자체의 시장인가 하는 문제는 생각해볼 문제다. 혹시 개발툴이 더 큰 기업에 인수되어 더 많은 자금과 기술, 인력을 투자하면 개발자는 결국 더 좋은 개발툴을 쓸 수 있으니 좋은 것이 아닐까. 개발툴이 플랫폼과 그만큼 더 밀접하게 연결되면 쓰기 쉬워지고 아무래도 OS를 만든 벤더의 개발툴이 더 낫지 않겠나. 개발툴 전문 업체든 플랫폼 업체든 어차피 돈 벌려고 하는 장사니까 똑같지 않을까. 당장 개발할 일거리가 산더미인데 그런 걸 생각해볼 필요가 있나.

물론 이러한 반문들은 대부분 사실이거나 사실에 가깝다. 개발툴 전문업체보다 플랫폼 업체가 훨씬 많은 자금과 인력을 가지고 있으며 그만큼 개발툴을 더 개선시켜왔다. 물론 기술의 면에서는 예외가 있을 수 있다. IT 업계의 역사에서 작은 벤처 기업이 대기업이 가지지 못한 뛰어난 기술과 아이디어로 혁신적인 제품을 만들어내는 걸 우리는 종종 봐왔으니까.

하지만 이런 시각에는 중요한 한 가지가 빠져있다. 개발툴 시장을 거의 장악하고 있는 플랫폼

벤더들이 과연 개발툴 자체의 시장을 인정하는가의 문제다. MS의 실적 발표에는 MS가 적지 않은 투자를 하고 있는 개발툴이 윈도우 부문으로 통합되어 별도의 집계조차 되지 않는다. 다른 플랫폼 업체의 경우도 비슷하다. 이것이 뜻하는 바는 무엇일까. 플랫폼 업체에게 개발툴은 아웃풋이 나오는 시장으로 인정되는 것이 아니라 플랫폼을 위한 투자일 뿐이라는 것이다. 즉 플랫폼 벤더에게 개발툴은 플랫폼을 위한 것일 뿐 개발툴 자체로서는 의미가 없다.

이것은 개발자에게 중요한 의미를 가진다. 개발툴 전문 벤더에 있어서는 개발툴의 판매 매출과 이익만이 개발툴의 향방을 결정한다. 다시 말해 개발자들이 많이 채택해주면 개발툴은 계속 업그레이드되며 매출이 나오지 않으면 단종을 고려하게 된다. 이것은 시장의 원리와 정확히 일치한다. 수요에 의해 공급이 결정되는 것이다. 반면 플랫폼 벤더의 경우 개발툴의 매출에는 거의 관심이 없다. 만약 플랫폼 전략에 개발툴이 도움이 되지 않게 되는 경우나 플랫폼 전략과 개발툴 전략이 충돌하는 경우에는 어떻게 될까. 대표적인 사례는 다시 비주얼 베이직이다.

지난 3월에 일제히 IT 관련 언론들에 기사로 실리면서 잘 알려진 사례를 보자. MS는 지난 3월 31일부로 비주얼 베이직 6 버전에 대한 일반 지원을 종료했다. 일반 지원을 종료했다는 의미는 단순히 무료 기술지원의 중단만이 아니라 버그 패치나 업데이트 등도 더 이상 제공되지 않는다는 의미이다. 하지만 그 이면에 실린 의미는 그 이상이다. 비주얼 베이직 구버전에 대한 지원 종료는 현실적으로는 비주얼 베이직 6로 작성된 애플리케이션이 미래 버전의 윈도우에서 문제가 생기든 말든 마이크로소프트로서는 알 바가 아니라는 것이며 닷넷 이전 버전을 사용하지 말라는 의미와 다를 바 없다. 비주얼 베이직 개발자들 상당수가 온라인에서 닷넷이 아닌 비주얼 베이직 6의 계보를 잇는 새 버전을 요구하는 대규모 서명운동을 벌이는 이유도 이 때문이다(**http://classicvb.org/petition/**).

현재 이 서명운동에 참여한 개발자는 9000명을 넘었으며 여기에 동참한 MVP만도 260여명이나 된다. MVP는 MS가 자사의 제품 확산에 핵심적인 고객군으로 공인한 엔지니어들이다. 전세계 MVP가 2600여명이고 그 중 개발자 MVP가 1000여명인데 여기에 서명한 MVP가 260여명이라는 것은 1/4을 훨씬 넘는 것이다. 물론 1000여명의 MVP 중에는 비주얼 C++ 등 비주얼베이직 개발자가 아닌 경우도 많이 포함돼 있을 것이므로 이것까지 고려하면 MVP의 반발은 전체의 절반에 육박한다.

이런 집단적인 반발에 대해 MS의 대응은 어땠을까. MS는 기존 VB의 계보를 잇는 새 VB 버전을 만들어달라는 성난 개발자들을 위해 VB.NET으로의 마이그레이션 방법에 대한 아티클들, 그것도 새로운 것도 아닌 몇 년 전부터 MSDN에 쌓여있던 아티클들을 링크한 사이트를 하나 만들어 'VB6 리소스 센터'라는 재미있는 제목을 달아줬다. (**http://msdn.microsoft.com/vbrun/**)

필자는 91년부터 10년도 훨씬 넘게 MS가 개발자 제국을 이루는데 가장 핵심적인 역할을 했던 비주얼 베이직 개발자에게 이렇게 대응하는 것은 심각한 문제가 있다고 본다. 그것도 구버전의 비주얼 베이직의 매출에 문제가 있었거나 개발자들이 외면했거나 했던 문제도 아니다. 만약 비주얼 베이직이 개발툴 전문업체의 제품이었다면 이런 일은 절대로 발생하지 않았을 것이다. 비주얼 베이직을 이전 버전의 언어 그대로 닷넷에 올리는 것은 OOP 및 CLI와의 연동 문제 때문에 곤란할 수도 있겠지만 비주얼스튜디오에서 비주얼 베이직의 Win32 환경 기반의 개발을 계속 지원하고

비주얼 베이직 닷넷과 함께 탑재하면 그만이다. 그런데도 **MS**가 이같은 탄원에 대해 조삼모사식의 조치만으로 외면하고 있는 것은 쉽게 짐작할 수 있듯이 **MS**의 개발툴을 사용하는 개발자들을 **Win32** 환경에서 닷넷으로 전환시키기 위해서다. 개발툴 자체의 시장은 계속 살아있음에도 불구하고 닷넷이라는 플랫폼 전략을 위해 개발자에게 변화를 강요하는 것과 다를 바 없다.

그렇다면 개발자들은 어떻게 대응해야 할까. 플랫폼 벤더의 개발툴은 피하고 개발툴 전문 벤더의 개발툴이나 오픈소스 개발툴만을 이용해야 할까. 물론 이것은 현실적이지도 않고 옳지도 않다. 그러나 개발자들은 앞으로도 플랫폼 벤더들의 입맛에 따라 휘둘리지 않으려면 한번은 깊이 심사숙고해야 할 문제임에 틀림없다.

자바 진영의 개발자 지원과 개발툴 현황

과거 수년간 자바 개발을 처음 접했던 윈도우 개발자들이 흔하게 놀라는 사실들 중 하나는 자바의 개발환경이 윈도우 기반에 비해 '원시적'이라고 할 정도로 단순하고 시간 소모적이라는 것이다. 비교적 최근까지 자바 개발자들 사이에서는 통합된 개발툴이 아닌 텍스트 에디터 기반으로 개발하는 사람들이 많았다. 텍스트 에디터를 이용하는 개발만이 진정한 개발이라고 믿어 의심치 않는 개발자도 많다. 이런 자바 개발자들의 성향 때문에 자바용 개발툴은 윈도우 개발툴 시장보다는 느리게 성장해왔다. 비주얼카페가 볼랜드에 인수되고 사이베이스가 파워**J**를 포기한 후 상용 자바 개발툴 시장은 사실상 **J**빌더가 독점에 가까운 상태가 되긴 했으나 에디터 기반 개발에 익숙한 자바 개발자들을 통합 개발툴로 끌어들이는 데는 그다지 영향을 미치지 못했다.

이런 상황을 혁신적으로 바꿔가고 있는 것이 이클립스다. 이클립스는 **2001**년말 **IBM**을 비롯하여 볼랜드, 머랜트, 레드햇 등 굵직한 **IT** 업체들이 참여하여 오픈소스로 진행하고 있는 개발툴 프로젝트로서, 오라클과 인텔 등 새로운 기업들이 잇따라 참여하고 있다.

이클립스 자체는 그렇게 대단하다고 말할 정도로 뛰어난 개발툴은 아니다. 하지만 가장 큰 장점은 기본 기능이 약한 대신 강력한 플러그인**(plug-in)** 지원으로 개발툴의 확장에 대단히 유리하게 만들어져 있다는 것이다. 이클립스 자체도 플러그인 구조로 구성되어 있으며 서드 파티의 플러그인을 추가함으로써 거의 무한대로 확장할 수 있다. 이런 이유 때문에 서드 파티 플러그인 중에는 오픈소스 외에도 상용 개발툴 기능도 많다. **IBM**의 상용 자바 개발툴인 웹스피어 스튜디오도 이클립스 기반이며 볼랜드는 최근 **J**빌더에서 독자적인 **IDE**를 포기하고 이클립스 기반에 **J**빌더를 올리는 것으로 전략을 수정하는 등 자바 진영에서 모든 개발 관련 도구들이 이클립스로 대통합을 이루는 분위기다. 이클립스라는 개발툴 자체가 또 하나의 플랫폼이 되는 것이다.

자바 진영에서 이클립스와는 조금 다른 길을 걷고 있는 것을 꼽으라면 역시 오픈소스 프로젝트인 썬의 넷빈즈다. 수년동안 새 버전의 공개가 중단되어 있던 넷빈즈는 이클립스의 부상에 자극받아 최근 새 버전을 연달아 내놓고 있다. 자바 진영에서 썬과 **IBM**은 대표적인 경쟁 관계인데 이클립스는 **IBM**이, 넷빈즈는 썬이 주도하고 있는 것이다. 사실 이클립스와 넷빈즈는 기능적으로 우열을 가리는 것은 어려울 정도로 세부 기능별로 장단점이 있다.

하지만 넷빈즈의 역공에도 불구하고 이클립스는 자바 개발툴의 대세가 되어가고 있고 넷빈즈가 이런

전세를 뒤집을 수 있는 가능성은 많지 않을 것으로 보인다. 프로그래밍 언어와는 달리 개발툴에 있어서는 한 개발자에게 두 개 이상의 툴이 상호보완적인 관계로 사용되는 경우가 드물기 때문이다. 이클립스와 넷빈즈를 통합하려는 **IBM**과 썬 간의 협의가 진행되기도 했으나 **2003**년말 썬이 이클립스 참여를 최종 거부함에 따라 이클립스-넷빈즈라는 초유의 오픈소스 통합은 무산된 바 있다.

그렇다면 왜 자바 개발자에게 이클립스라는 개발툴의 존재가 중요할까. 단지 공짜이기 때문만은 아니다. 유무료 여부는 개발자가 속한 개발회사의 자금 사정과 관계가 있을 뿐 그 이상의 의미는 크지 않다**(**이클립스 프로젝트를 이끌고 있는 **IBM**에게도 이클립스가 오픈소스이건 상용제품이건 그다지 중요한 문제가 아닐 것이다**).** 오히려 더 중요한 의미는 다시 생산성이다. 이클립스의 보급으로 텍스트 에디터 기반의 시간 소모적인 작업으로 일관해왔던 많은 자바 개발자들의 생산성을 더 높일 수 있는 기회를 보게 된 것이다.

'뛰어난 개발툴'은 후발 주자인 **MS**의 닷넷이 자바를 따라잡기 위해 내세우는 단골 메뉴 캐치프레이즈 중 하나이기도 하다. 자바 개발자들 사이에 이클립스가 확산되고 이클립스를 지원하는 다양한 강력한 플러그인들이 늘어나면서 **MS**는 더 이상 개발툴과 생산성으로 자바 개발자들을 유혹하기는 어렵게 됐다. 이클립스의 등장은 윈도우와 자바 진영의 경쟁 구도에 또 다른 이정표가 되고 있는 것이다.

닷넷 대세론의 허와 실, 그리고 개발자

앞에서도 말했듯이 닷넷은 많은 개발자들에게 혼란의 주요 이슈들 중 하나다. 기존의 윈도우 개발자들은 대부분 지금도 **Win32** 환경에서 개발하고 있으며 비주얼 스튜디오 구버전이나 델파이**,** **C++**빌더의 구버전을 더 많이 사용하고 있다. 자바 개발자들은 닷넷이 언제쯤 대세가 될지 여부에 큰 관심이 없으며 '닷넷이 실제로 대세가 되면 그때 넘어가면 되지' 정도의 분위기가 지배적이다. 닷넷으로 대세가 넘어갈지 아닐지에 관심이 있는 것은 윈도우 개발자들 뿐이라고 해도 과언이 아니다.

개발자의 입장에서 봤을 때 닷넷 자체는 개발자들이 요구하지 않은 기술이라고 할 수 있다. 닷넷이 필요한 핵심적인 이유 중의 하나라고 자주 언급되는 웹 서비스는 물론 좋은 기술임에 틀림없다. 그러나 닷넷을 통하지 않고도 웹 서비스를 개발하는 데는 별 문제가 없다. 예를 들어 볼랜드는 이미 **2001**년부터 **Win32** 버전의 델파이와 **C++**빌더에서 웹 서비스 개발을 지원하고 있다. 물론 **MS**도 충분히 할 수 있는 일이겠지만 일부러 **Win32**에서 지원하지 않고 있다고 볼 수 있다.

오히려 많은 개발자들이 닷넷이 성공할 것이라고**(**혹은 적어도 실패하지 않을 것이라고**)** 쉽게 가정하는 이유 중 가장 큰 것은 영향력과 자본력 등 **MS**가 가진 사상 초유의 **IT** 권력 때문이다. 영화 '고질라'에서 내세웠던 단 두 단어 짜리의 홍보 문구 '**Size Matters!**'와 같은 것이다. 크더라도 이만저만 큰 것이 아니니까 문제**(matter)**가 되는 것이다. 이러한 필자의 생각에는 몇 가지 무시할 수 없는 징후와 이유들이 있다.

무엇보다 닷넷의 확산속도다. 닷넷이 실체화된 것이 벌써 만으로 **4**년이 넘었지만 닷넷으로 마이그레이션한 윈도우 프로젝트는 그리 많지 않다. 추상적인 대세론을 체감하기에는 마이그레이션의

속도가 너무 늦다. 윈도우 3.1도 윈도우 95가 나오기 전 4년 만에 상당한 인기를 끌며 DOS를 몰아냈고 윈도우 95는 불과 1~2년 만에 IT 업계를 휩쓸며 엄청난 반향을 일으켰다. MS는 내년에 출시될 예정인 코드네임 롱혼 윈도우 비스타의 출시가 이 같은 효과를 낼 것으로 기대하고 있지만 윈도우 98 이후 윈도우 업그레이드 비율은 매년 지속적으로 하락하는 것으로 알려져 있다. 게다가 윈도우 비스타에서 닷넷이 기반 프레임워크가 될 것이 당연하다고 믿어왔던 예측도 깨지면서 닷넷에 기대를 거는 개발자들을 더욱 당혹스럽게 하고 있다.

기능면에서 비스타에서 추가된 기능들이 (엔지니어의 입장이 아닌) 일반 소비자 시장에서 얼마나 매혹적일지도 아직은 미지수다. 윈도우 라이선스 매출 중 높은 비율을 점하는 기업 시장의 업무용 PC에서 비스타의 화려한 UI는 사실 큰 의미가 없다(이는 대대적인 홍보 공세가 있었던 윈도우 XP의 사례와도 겹치는 부분이다. 98에 비해 XP의 탁월한 안정성과 예쁘장한 UI가 그다지 효과를 거두지 못했는데, 그보다 더 멋진 UI와 편의성으로 소비자들의 선택을 이끌어 낼 수 있다는 논리는 너무 낙관적인 전망이 아닐까).

물론 현실에서 분명히 닷넷 관련 프로젝트는 여러 건이 성사되었고 또 다른 여러 건은 진행 중이다. 이렇게 하나씩 늘어가다 보면 닷넷의 확산에도 당연히 가속도가 붙을 것이고 어쩌면 자바를 따라잡거나 넘어설지도 모른다. 그러나 여기서 문제는 속도, 더 엄밀하게는 상대적인 속도이다. 이미 자바가 엔터프라이즈 시장의 대세로 자리잡았지만 더 넓힐 수 없을 정도로 절대적인 점유율을 가진 것이 아니라 지금보다 충분히 더 성장할 여지가 있는 상태에서 과연 지금의 확산속도로 이러한 경향을 뒤집을 수 있느냐 하는 것이다.

아이러니하게도 MS가 닷넷에 올인을 선언한 2000년 이후, 매년 비주얼 베이직 개발자들은 줄어들고 있다. MS는 이렇게 줄어드는 비주얼 베이직 개발자들이 C#으로 이동해주기를 간절히 바랄 것이며 최소한 비주얼베이직닷넷으로 전환하기를 기대하겠지만 에반스 데이터의 조사 결과와 다른 통계들을 보면 아직도 C# 개발자의 숫자는 미미하고 비주얼베이직닷넷 개발자와 구버전 비주얼베이직 개발자를 모두 합해도 전성기 때의 비주얼베이직 개발자 숫자의 절반 이하인 것으로 나타나고 있다. **http://www.evansdata.com/n2/pr/releases/EDCEMEA04_02.shtml**, **http://www.tiobe.com/tpci.htm**

한때 점유율 1위였던 비주얼 베이직 개발자가 현재 크게 내려앉고 그 사이 다른 언어는 큰 변동이 없는데 자바 개발자만이 크게 늘어난 것을 보면, 이렇게 줄어든 비주얼 베이직 개발자들의 상당수가 자바 진영으로 이동했다고 추측된다. MS가 야심차게 밀어붙이고 있는 닷넷 전략으로 인해 오히려 MS의 가장 든든한 우군이던 비주얼 베이직 개발자들이 이탈하는 상황일 수도 있다. MS는 이런 상황을 역전시키기 위해 혼신의 힘을 다하겠지만 이런 상태가 몇 년 지속된다면 이후 MS는 어떤 선택을 하게 될 지 자못 궁금하다.

그럼에도 불구하고 막강한 자본력 및 압도적인 홍보 공세 외에도, MS가 닷넷의 엔터프라이즈 시장으로의 진입에 자신감을 가질 수 있게 해준 것은 기존의 수많은 Win32 개발자들이라는 막강한 배경에 대한 기대도 컸을 것이다. MS가 Win32 개발자들에게 기대를 걸었던 것은 이들이 닷넷으로 빠르게 넘어갈 것으로 예상했기 때문인데 아직까지 이런 시나리오는 현실화되지 못했다. 또 MS는 닷넷이 성능면에서 자바보다 월등하다는 주장도 자바 개발자들 사이의 주요 이슈가 되기를 바랬겠지만, 엔터프라이즈 컴퓨팅에서 성능이 항상 최우선 이슈는 아니며, 자바와 닷넷 사이의 이런

성능 논쟁은 자바 개발자들 사이에서 이미 시들한 주제가 되어버린 듯하다.

MS의 닷넷 전략에서 핵심적인 역할을 할 비주얼스튜디오닷넷은 언매니지드 **C++**이라는 일부 외에는 오로지 닷넷만을 위한 개발툴이다. 아직 **Win32**에 머물러있는 개발자들의 결단을 강요하고 있는 셈이다. 이와는 반대로 개발툴 전문 벤더인 볼랜드가 지난 달 발표한 볼랜드 디벨로퍼 스튜디오 **2006**은 **Win32**와 닷넷을 함께 지원한다. 개발툴의 우열을 떠나 소비자이자 주권자인 개발자 입장에서는 역시 플랫폼 벤더와 개발툴 전문 벤더의 차이를 실감할 수밖에 없는 대목이다. **MS**의 입장에서 닷넷은 이미 돌이킬 수 없는 선택이 됐겠지만 **MS**가 아닌 별개의 소비자로서 대다수 개발자들에게는 그렇지 않을 수 있다. 현재 **Win32** 환경과 닷넷 환경 사이에서 개발자들의 선택의 방법과 시기가 중요한 것은 이 때문이다.

마무리

꽤 장황하게 얘기를 풀었다. 이제 처음 제기했던 의문에 대한 결론을 생각해보자. 수많은 기술들을 흡수하며 발전해온 개발툴의 방대한 규모만큼 개발자가 해피해졌는가**?** 플랫폼 벤더들이 서로 자사의 울타리로 개발자들을 끌어들이기 위해 쟁탈전을 벌이는 동안, '몸값'이 높아져야 할 개발자의 대우는 오히려 거꾸로 하락해온 것이 아닌지 의문스럽다. 혹시 개발자들을 각자 고유의 차별화된 보유 기술과 가치를 인정받기보다는 윈도우 개발자 몇 명, 자바 개발자 몇 명이라는 식으로 플랫폼 벤더들의 세력 셈법의 숫자로 카운트되는 데에 더 고 있는 것은 아닐까. 또 플랫폼 벤더들이 앞다투어 각자 평준화된 기술을 수없이 뿌려대는 동안, **(**실제는 그렇지 않더라도**)** 적어도 외견상으로는 개발자들이 평준화되어 보이게 되는 부작용이 생겨버린 것은 아닐까. 개발자가 실무에서 고민할 웬만한 문제들은 대부분 플랫폼 벤더에서 공개한 문서들에서 찾아볼 수 있는 현실에서, 엔지니어가 아닌 독자들의 고용주들은 과연 많은 웹 페이지를 링크해둔 개발자와 진정 문제 해결 능력을 가진 고급 개발자를 구분할 수 있을까. 만약 이런 구분이 쉽지 않다면, 앞으로 **5**년후, **10**년후 여러분의 대우는 과연 쌓인 경력과 실력만큼 상승할 수 있을까. 또, 비주얼베이직의 예에서 본 것처럼 개발자들의 의사와는 반대로 단지 플랫폼 벤더의 전략 때문에 여러분이 오랫동안 사용해오던 언어와 개발툴이 사라지는 경우가 다시 발생하지는 않을까. 그리고 이런 모든 면들을 생각했을 때 과연 거대 플랫폼 벤더 주도의 개발툴 시장이 과연 개발자들에게 마냥 좋은 방향이기만 한가.

어떻게 판단을 내리든 그것은 필자의 몫이 아니라 독자인 개발자 여러분 각자의 몫이다. 필자는 우리 개발자들이 개발툴들을 사용하면서 한번은 돌아봐야 할 화두를 던져주었을 뿐이다. 많은 반론도 있을 수 있고, 반대로 이 연장선상에서 논의를 더 확장시킬 수도 있을 것이다.

이번 특집 기사를 청탁 받았을 때 걱정부터 앞섰던 것이 사실이다. 개발툴은 프로그래밍 언어보다 습관성, 나쁘게 얘기하자면 중독성이 강해서, 개발자에 따라서는 개발툴에 대한 선호가 오히려 메인이라고 할 수 있는 프로그래밍 언어 자체보다도 더 강한 경우도 적지 않다. 그래서 개발자마다 가진 특정 개발툴에 대한 강한 선호도를 최대한 비켜가면서 개발자 입장에서 전체 개발툴 세계에

대해 조망할 수 있는 시각을 제공하는 것이 가능할까 하는 것이 가장 큰 고민거리였다.

이 글을 탈고하기 며칠 전 이미 경험이 있는 한 지인이 필자에게 반 농담으로 웃으면서 충고했다. 글을 그런 식으로 쓰면 엄청난 항의 메일 폭탄을 얻어맞게 될 거라고. 필자도 그럴 거라고 생각한다. 그래서 웃으면서 대답했다. 개발툴과 개발자의 관계 같은 민감한 주제로 글을 쓰려면 꽤 뻔뻔해져야 하지 않겠느냐고. 개발자는 그 모습 그대로 개발자여야 하고 개발자가 하는 일, 즉 **SW** 개발 이상의 다른 목적의 수단으로 휘둘려서는 안되지 않을까.

정리 | 박상훈 | **nanugi@imaso.co.kr**