embarcadero | Whitepaper

RAD스튜디오

관리자를 위한 안내서

원문 : 스테픈 볼(Stephen Ball) - 2021년 4월

번역 : 데브기어 - 2021년 4월

목차

머리말 **02**

Part 1 - 진화하는 소프트웨어 개발 세상 속의 RAD스튜디오 **03**

RAD와 ASD에서부터 애자일(Agile)까지 03
소프트웨어 개발에 영향을 주는 시장 트렌드 03
RAD스튜디오의 현재 04
인터페이스의 중요성 06
프로세스에 중점을 둔 혁신 - 유닛 테스트(Unit Testing)의 사례 06
제품에 중점을 둔 혁신 - 파이어몽키(FMX)의 능력 07
파트너 관계를 통한 혁신 08
인수를 통한 혁신 08
현대화인가 아니면 재구축인가? 08
안전을 추구하기, 빠르게 진행하여 시간을 벌기 09
UI 테스트를 마이그레이션 방법으로 활용하기 09
윈도우10의 최신 기능 추가하기 10
돈과 시간을 절약할 수 있도록 고안하기 10

Part 2 - 두 세상에서 최고가 되기 - 왜 RAD스튜디오인가 **12**

크로스 플랫폼 개발 도구와 방법의 진화 12
모바일이 비즈니스 프로세스에 미치는 영향 13
멀티 플랫폼 달성 14
순수 네이티브 대 하이브리드 애플리케이션 15
타협 하지 않기 - 크로스 플랫폼과 순수 네이티브 모두에서 최고를 추구하기 16
FMX가 자마린 폼즈 (Xamarin Forms)와 다른 점 17
엔터프라이즈 데이터와 원격 데이터 연결 18
코드가 적은 로우 코드 (Low-Code) 애플리케이션 플랫폼과 RAD 18

Part 3 - RAD스튜디오 현재 - 미래를 위한 투자 **20**

RAD와 데브옵스(DevOps) 20
현재와 미래 모두를 위해 투자하기 22

요약 **22**

이 문서 주제 밖의 참고 사항 **23**

엠바카데로 방식 **24**

머리말

소프트웨어 세상은 오브젝트 지향 프로그래밍(OOP), 애자일 개발 (Agile Development), CI, 데브옵스(DevOps), 로우 코드 (Low-Code), 엔터프라이즈 서비스, 마이크로 서비스, UI/UX 설계 등 혁신적인 개념들에 의해 번성한다.

이 많은 용어들 안에는 공통된 주요 개념이 있는데, 이 개념은 최근에 다시 급부상이며, 오늘날 소프트웨어 개발에 사용되는 도구와 프로세스에 지속적으로 지대한 영향을 끼치고 있으며, 혁신적인 방식을 담은 관련 제품들이 한결같이 강조하는 용어이기도 하다. 그 용어는 다름아닌 **애플리케이션을 빠르게 개발하기 (RAD, Rapid Application Development)**이다.

마이크로소프트, 구글, 애플, 아마존, 세일즈포스 등 큰 소프트웨어 회사들은 모두 RAD 방식을 굳게 믿고 널리 알리고 있다. 이 문서는 RAD가 어떻게 진화해왔는지, 현재 **RAD스튜디오**가 도구와 프레임워크 혁신을 어떻게 지속하고 있으며, 최신의 개발 절차와 현장의 관행을 어떻게 지원하고 있는지를 알아본다.

누가 읽으면 좋은가

이 문서는 CTO 또는 소트프웨어 기술팀을 이끄는 리더들, 특히, 시장 트렌드를 이해하고, 데스크톱과 모바일 모두에 적합한 가능성 있는 해법들을 평가하거나 추진하려는 리더들을 위해 작성되었다.

델파이나 **C++빌더**로 작성한 코드가 있고 오래된 레거시 시스템 안에서 그 코드를 계속 유지하고 있는 곳의 기술 리더라면 이 문서가 중요하다. 왜냐하면, 코딩이 지금까지 어떻게 발전해왔는 지를 폭넓게 이해하고, 이미 보유하고 있는 코드 기반의 가치를 어떻게 평가하고 다룰 것인지에 대한 방향을 잡는데 도움이 되기 때문이다.

진화하는 소프트웨어 개발 세상 속의 RAD스튜디오

RAD와 ASD에서부터 애자일(Agile)까지

RAD는 오랫동안 지속적으로 발전되어 왔는데 최근에는 다시 급부상하고 있다. RAD는 **델파이**(1995)와 **비주얼스튜디오**(1997년) 등 세상을 뒤흔든 개발 도구들이 나오면서 널리 퍼졌다. RAD 도구들은 같은 시기에 유행하던 **적응식으로 소프트웨어를 개발하기 (ASD, Adaptive Software Development) 방식**과 연합하여 **비즈니스 프로세스 재설계 (BPR, Business Process Re-Engineering)**를 급속히 확산시켰다. 이 시기는 마이크로소프트사의 윈도우가 PC 시장을 지배하던 시기와도 일치한다.

초기의 RAD 설파자들이 ASD를 채택한 이유는 제품 생명 주기를 더 짧게함을 통해 얻는 장점을 취하기 위해서였다. 즉 프로토타입을 제공함으로써 고객의 피드백을 조기에 받고 전체 배포의 위험을 낮추고, 버전 초기부터 바로 사용자에게 가치를 제공하고자 했다. RAD에 기반을 둔 개발 프로세스 역시 지난 25년간 계속 진화하여, 소프트웨어 설계, 테스트, 코드 관리 등 개발 프로세스에 관련된 여러 보조 자산들이 출현했다. 이것들이 모이고 커진 결과가 바로 오늘날의 **'애자일 개발(Agile Development)'**이다.

소프트웨어 개발에 영향을 주는 시장 트렌드

최근 15년간, 개발 프로세스와 소프트웨어 개발 분야 모두 큰 변화를 겪었다.

몇 가지를 꼽아보면:

- 모바일 플랫폼과 모바일 하드웨어가 새로 출현함에 따라, 우리의 정보 접근, 처리, 협업 방식이 바뀌게 되었다.
- 인터넷에 연결된 장비를 기준으로 볼 때, 모바일은 데스크톱을 앞질렀다.
- 앱스토어는 모바일 애플리케이션 배포 시장을 장악했다.

- 32비트는 64비트에게 밀려났다. 애플 앱스토어와 구글 플레이는 이제 64비트 앱인 경우에만 신규 배포를 허용한다.
- 자기 소유의 장비을 가져와서 사용하기 (BYOD, Bring Your Own Device) 정책은 '누가 장비를 소유하는가' 라는 문제를 던졌을 뿐만 아니라 데이터에 접근해야하는 장비와 접근 방식까지 변화시켰다.이로 인해 개발자에게는 배포와 보안 측면의 새로운 숙제가 생겼는데, 그 이유는 다루어야하는 환경이 훨씬 광범위해졌고 여러 운영체제와 여러 장비(와 장비마다 다른 기능)들이 섞여있기 때문이다.
- PaaS(Platform-as-a-Service) 클라우드 인프라는 하드웨어 소유와 장애 극복 이슈에 대해 새롭게 다시 생각하게 되는 계기가 되었다.
- PaaS는 SaaS(Software-as-a-Service)가 더 안정적으로 제공될 수 있는 기반이 되었고, 낮은 비용을 정기적으로 지불하는 방식을 제시했다.
- 요즘 사용자들은 시스템끼리 고도로 연결되어 상호작용하기를 기대한다. '공개 기반 혁신'을 이끌기 위해서는 엔터프라이즈 사용자들이 사용하는 여러 다른 시스템들과 공존하고 통합할 수 있도록 하는 API가 필요하다.
- 마이크로서비스는 시스템 안에서 (예를 들어, 언어 번역 서비스, 푸쉬 알리, 날씨 데이터 등과 같은) 새로운 기능을 빠르게 구현하는 대표적인 방법이 되었다. 기본 전송 매커니즘은 HTTPS를, 기본 데이터 구조는 JSON을 사용한다.
- 컨테이너화와 데브옵스(DevOps)는 소프트웨어 개발과 배포 방식을 바꾸어서, 배포를 꾸준히 그리고 빠르게 할 수 있도록하는 틀을 제공한다.
- NoSQL이 정착되고, 전통적인 SQL 데이터 저장소 역시 더욱 발전함에 따라 데이터 저장이 폭발적으로 커지고 있다. 그 결과 데이터는 종종 새로운 원유라고 불린다.

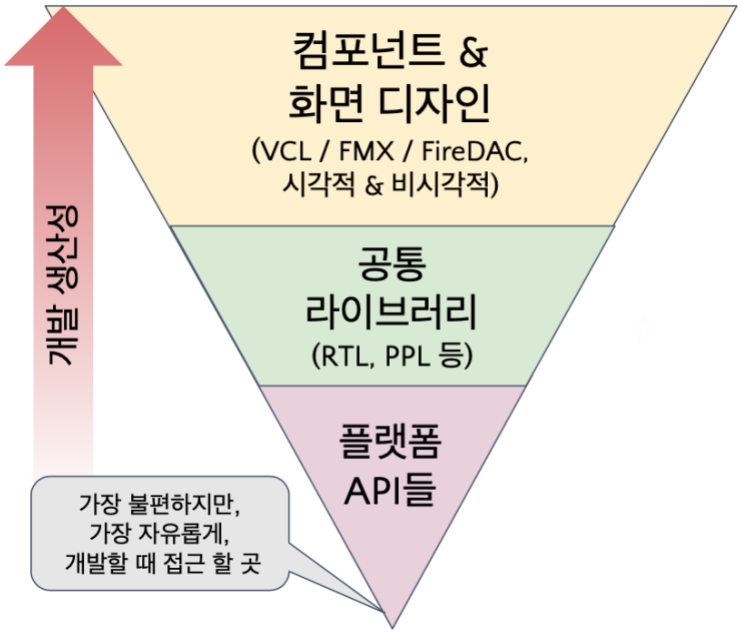
위와 같은 몇가지 주요 시장 트렌드에 따르면, 오늘날의 개발 환경은 RAD가 처음 출현했을 때 즉 마이크로소프트사의 윈도우가 시장의 요구 사항을 지배하던 시대와는 아주 멀리 떨어져있다.

RAD스튜디오의 현재

RAD스튜디오는 단일 코드 베이스를 사용하여 멀티 플랫폼 (윈도우, 리눅스, 맥OS, iOS, 안드로이드)용 순수 네이티브 코드로 컴파일한다. 이 독특한 방식은 매우 빠르고 유연하게 앱을 개발을 개발할 수 있으면서도 실행 속도가 빠른 앱을 만들어 낸다. **RAD스튜디오**가 다른 언어와 프레임워크에 비해 **5배까지 빠르다는 점**은 오랜 기간을 거치면서 증명되었다. 게다가 **RAD스튜디오**는 모든 주요 플랫폼 시장에 보다 빠르게 앱을 출시할 수 있다는 확신을 준다. 단일 코드 기반이라는 특징 덕분에 개발과 테스트 시간 모두 단축되기 때문이다.

델파이에 있는 라이브러리와 컴포넌트에 반영된 최초의 기본 설계 덕분에, 개발자들이 **RAD스튜디오**를 이용하여 25년이 넘도록 자신의 코드에 장기 투자를 할 수 있었다. 여기에 적용된 패턴은 너무나 강력하기 때문에 델파이 이후에 이 방식을 받아들인 (C#과 .NET 등) 많은 언어와 프레임워크에서도 쉽게 알아챌 수 있다.

만약 당신이 이 방식에 익숙하지 않다면, 컴포넌트가 RAD 개발에 기여하는 진정한 가치는 플랫폼의 API를 추상화하여, 쉽게 사용할 수 있는 빌딩 블록을 만들어서 낮은 수준의 시스템 상 호출은 컴포넌트가 알아서 처리해주는 것이라는 정도만 기억해도 좋다. 이 컴포넌트들이 강력할 수 있도록 아래에서 받쳐주는 것이 라이브러리들인데 (파일 시스템, 날짜/시간, 화면 정보 처리 등) 재사용할 수 있는 공통 기능을 제공한다.



(그림1. RAD스튜디오의 설계 기본)

혁신과 변화의 물결이 시장에서 밀려드는 상황에서, **RAD스튜디오**에서 사용된 방식, 즉 인터페이스를 통해 강력하게 지원하는 컴포넌트와 오브젝트 지향 방식은, 새 기능, 새 플랫폼, 새 능력을 쉽게 추가할 수 있으면서도 기존 호환성 역시 수준 높게 유지하는 결과를 만들었다.

인터페이스의 중요성

독특하게 **RAD스튜디오** 만이 **여러 주요 플랫폼에 맞는 순수 네이티브 코드**를 생성하는 능력을 가질 수 있었던 가장 중요한 요소는 **델파이**에 구현된 방식이 인터페이스 중심이기 때문이다.

'차량이 달라도 운전대를 사용하는 방식이 같다'는 점에 비추어 생각해보자. 자신의 차를 운전할 때 뿐만 아니라 처음 운전하는 트럭이나 보트에서도 운전자가 운전대를 사용하는 방식은 같다.

운전자 입장에서 보는 인터페이스는 같지만, 내부에서 작동하는 동작은 크게 다른 것과 마찬가지로, 개발자가 코드를 작성할 때에는 공통 인터페이스를 사용하지만 공통 인터페이스의 내부에는 각 플랫폼 별 고유 구현이 반영되어 있다.

이제 **RAD스튜디오**의 사례를 통해 개발자들이 어떻게 프로세스와 플랫폼 분야에서 밀어닥친 혁신의 물결을 잘 타고 지속할 수 있었는지를 살펴보자.

프로세스에 중점을 둔 혁신 - 유닛 테스트(Unit Testing) 사례

유닛 테스트는 프로그래밍을 통해 코드 테스트 반복을 가능하게 함으로써 코드 변경으로 인한 퇴보하거나 잘못되는 문제, 버그 등을 줄일 수 있도록 도와준다. 유닛 테스트 오픈 소스 프로젝트들은 **RAD스튜디오** IDE에 있는 **오픈툴스 API(Open Tools API)**를 통해 통합되는데 개발자가 직접 코드의 테스트 내용을 실시간으로 볼 수 있도록 한다. 유닛 테스트 실행은 지속적 빌드 프로세스 단계 중 하나가 되기도 하며, 유닛 테스트 결과는 지속적 빌드 프로세스에 연결된 외부 시스템에 기록될 수도 있다.

유닛 테스트가 도입된 2000년 경부터 일부 개발자들은 **MVC(Model View Control)**과 **MVVM(Model View View-Model)** 등 여러 디자인 패턴들을 활용하여 시스템의 논리적인 부분에 대한 테스트를 더 쉽게 할 수 있도록 했다. 비록, 이 디자인 패턴들로 인해 기존 코드나 IDE, 프레임워크가 즉시 바뀌거나 하지는 않았지만, 코드 작성이 어떻게 되어야 하는지에 대한 방향을 제시하고 이끌었으며, 의존성 주입과 같은 보조적 접근을 지원하는 **Spring4D** 등의 부가 프레임워크가 출현하는 바탕이 되었다.

제품에 중점을 둔 혁신 - 파이어몽키(FMX)의 능력

코드가 어떻게 작성되고 어떤 프로세스를 따라야하는 지에 대한 혁신의 사례로 유닛 테스트를 들어보았다. 하지만, 수용해야 할 혁신들 중에는 제품 자체의 기능을 강화해야 가능한 것들도 있다. 2011년 9월, 크로스 플랫폼 프레임워크인 **파이어몽키(FMX)**가 처음 소개되었다. FMX의 첫인상은 윈도우용 RAD 개발을 가능하게 했던 **비주얼 컴포넌트 라이브러리(VCL, Visual Component Library)**와 많은 면에서 닮았다.

새 컴포넌트들로 구성된 FMX는 단일 코드 기반으로 각 플랫폼에서 네이티브로 작동되는 앱을 만드는 크로스 플랫폼 코딩이 가능하도록 기존의 내부 런타임을 업데이트하였다. 이것은 결코 작은 성과가 아니며, FMX 프레임워크는 지난 10년간 계속 진화하여 지금은 **윈도우, 리눅스, 맥OS, iOS, 안드로이드용 네이티브 개발을 단일 코드**로 하는 가장 광범위한 프레임워크가 되었다. 새 컴파일러들이 추가되고, 앱 스토어로 바로 배포하기 등 빌드 구성 역시 더 다양해졌다.

UI를 다루는 FMX 뿐만 아니라, UI를 데이터와 개체 모델에 바인딩하는 **시각적 '라이브 바인딩스'**와 같은 새 기능도 추가되었다. 또한 다양한 엔터프라이즈 데이터를 일관된 방식으로 연결할 수 있도록 강화된 **파이어닥(FireDAC)**은 여러 플랫폼에서 작동된다. **RAD스튜디오** 컴포넌트가 이렇게 진화함에 따라 수십년간 윈도우에서만 데이터베이스를 연결하던 코드가 모바일 플랫폼에서도 작동될 수 있게 되었다.

빠르게 향상하기 위해 **RAD스튜디오**는 **주요 오픈 소스들**도 사용하고 있다. 새 컴파일러는 오픈 프로젝트인 LLVM을 사용해서 만들어졌기 때문에 각 플랫폼 별로 최상의 런타임 성능을 제공한다. IDE에는 **언어 서버 프로토콜(LSP)** 지원이 도입되었다. 이와 같은 표준 기반 접근법은 밝은 미래를 받쳐주는 기반이 된다.

파트너 관계를 통한 혁신

어떤 혁신은 파트너 관계를 통해 실현되었다. **RAD스튜디오**는 IDE 중 세계 최초로 **마이크로소프트사의 데스트톱 브릿지**를 지원함으로써, 마이크로소프트 스토어로 바로 배포하기, 전통적인 윈도우 애플리케이션을 최신 엔터프라이즈 배포 매커니즘을 통해 배포하기 등을 실현했다. 또한 마이크로소프트 **엣지** 브라우저 지원은 이 웹 브라우저가 공식 발표되기도 전에 IDE에 추가 되었다.

인수를 통한 혁신

엠바카데로는 아이데라(Idera)사가 소유하고 있다. 아이데라는 급성장 중인 기술 그룹사이다. 덕분에 작업을 더 빠르게 할 수 있도록 하는 더 많은 개발자 도구들을 채택할 수 있는 문이 열렸다. **라노렉스(Ranorex), 센차(Sencha), 아쿠아 데이터 스튜디오(Aqua Data Studio)** 같은 도구들은 이미 **RAD스튜디오 아키텍트** 에디션에 들어갔다. 그 결과 개발자들은 이제 라노렉스의 강력한 기능을 활용하여 UI 테스트를 할 수 있으며, 광범위한 데이터베이스를 관리하기 위해 접근할 때 아쿠아 데이터 스튜디오로 단순화할 수 있어서 개발을 더 빠르게 할 수 있다. 또한 센차 아키텍트를 사용하여 RAD 방식으로 자바스크립트 개발을 할 수 있다.

게다가 써드파티 컴포넌트 제조사들의 에코시스템 역시 강력하므로, **RAD스튜디오**에서 같은 언어와 같은 코드를 사용하여 강력한 웹 애플리케이션을 만들 수도 있다.

RAD스튜디오는 코드를 강화하는 오픈 프레임워크, 컴포넌트, IDE의 핵심 기능을 지원하는 애드온 등을 제공하는 써드파티 에코시스템을 계속해서 활성화해오면서 함께 발전하고 있으며 지금은 열정적인 개발자 기술 중심 그룹 안에서 새롭게 성장 중이다.

현대화인가 아니면 재구축인가?

델파이 또는 **C++빌더**로 구축한 기존 코드를 가지고 있다면, 지금 있는 코드가 실제로 쓰이는 유스케이스는 무엇이고, 그 유스케이스 지원을 앞으로도 계속 할 수 있을까? 하는 질문을 가장 먼저 해봤을 텐데 그 대답이 만약 '앞으로도 계속 적용된다' 였다면, 지금 가진 코드의 가치는 분명히 있다. 그렇다면, 지금 있는 코드에서 해당 로직을 어떻게 꺼내어 앞으로 사용될 곳에서 적용할 것인지 궁금할 것이다.

앞으로의 계획을 준비할 때, 주안점은 다음과 같다.

- 시스템의 미래 아키텍처를 그려 본다. (예, 원격/모바일 접근 필요 여부 등)
- 위험과 비용을 식별하고 관리한다.
- 일정과 총비용을 검토한다.
- 새 프로젝트가 필요한 때가 언제인지를 파악한다.

이제 이 주안점 중 몇가지를 살펴보자.

안전을 추구하기, 빠르게 진행하여 시간을 벌기

기존 레거시 코드 베이스를 지속적으로 사용하려고 할 때의 장애물 중 하나는, 회사 내의 코딩 관행이 첫 버전 이후 지금까지 너무나 오랫동안 진화해왔다는 점이다. 기존 코드와 프로젝트는 여전히 효력이 있겠지만 프로젝트를 지금 새로 시작한다고 생각하면, 개발팀에서 다른 방식으로 해볼 만한 것들도 일부 있을 것이다.

기존 코드를 현대화할 계획이 있다면 기존 코드를 단계별로 업데이트하는 것이 가장 안전한 방법이다. 고객은 언제나 최신 상태인 제품을 사용할 수 있고, 지속적으로 제품 업데이트를 진행하면서 새 구현을 적용해 갈 수 있다.

UI 테스트를 마이그레이션 방법으로 활용하기

코드 테스트로 돌아가면, 코드에 테스트를 추가함으로써 더 빠르고 효과적인 테스트를 보장할 수 있고 이것이 개발 프로세스의 한 부분이 되면 좋다는 생각을 모두가 한다. 이미 유닛 테스트는 코드 테스트에서 이미 한 부분을 차지하고 있으며, 새 코드, 새 함수, 새 클래스에 유닛 테스트를 추가하기 역시 쉽다. 또 다른 방법으로 **UI 테스트**도 인기가 있다. UI 테스트 도구 중에 현재 **VCL** 개발을 잘 보완하고, 향후에도 코드 변경을 확인할 수 있도록 쉽게 기반을 추가할 수 있는 도구는 (아키텍트 에디션에 들어있는) **라노렉스**(Ranorex)이다. RAD 개발자들은 UI 테스트를 만들어서 자신의 애플리케이션이 사용자 수용 테스트(UAT, User Acceptance Testing) 단계로 넘어가기 전에 직접 점검할 수 있다. 라노렉스는 UI층만 다루기 때문에 개발자가 원한다면 얼마든지 내부 코드를 다시 작성할 수 있으며, 사용자 경험 상 바뀌는 것이 없는 지를 확인하는 방식으로 모든 작동을 검증한다.

또한 UI 테스트는 **RAD스튜디오**의 오래된 버전을 새 버전으로 마이그레이션 할 때, 소프트웨어를 확인하는 매우 훌륭한 방식이기도 하다. 라노렉스는 윈도우 통제 핸들 상에서 작동하기 때문에, 개발자가 UI를 재배치 하여도 해당 테스트가 망가지지 않는다. (로그인 행위 등) 반복되는 공통 행위를 저장해 두고, 테스트 여러곳에 넣어서 사용할 수 있으므로, 빠르고 유연하게 테스트를 작성할 수 있다. 또한, 테스트가 생성되고 나서, RAD스튜디오 상에 구축되기 때문에 개발팀 바깥에 있는 기여자들에게 테스트를 만들도록 전달할 수 있어서 인력 부담을 덜 수 있다.

윈도우10의 최신 기능 추가하기

RAD스튜디오의 컴포넌트 설계 원칙 덕분에, 기존 애플리케이션의 어떤 UI도 몇단계만 간단히 거치면 빠르게 강화할 수 있다. 또한 오래된 윈도우 버전에 대한 지원도 유지할 수 있는 **RAD스튜디오** 컴포넌트 만의 특별한 장점은 윈도우 10 이전 버전에 갇혀있는 일부 사용자들까지도 RAD스튜디오로 만든 앱을 계속 업데이트할 수 있도록 해준다.

RAD스튜디오는 윈도우 10에서 제공하고 있는 높은 DPI 모니터와 DPI가 서로 다른 여러 모니터 지원을 구현하기 위해 전통적인 **TImageList**를 대체하는 **TVirtualImageList**와 **TImageCollection**를 제공한다. 새 방식은 높은 DPI 모니터 별로 적용되는 지원 API를 캡슐화해 놓았으므로, 따로 코드를 작성하지 않아도, 각 모니터의 스크린 해상도 별로 알맞은 크기의 이미지가 반영된다.

돈과 시간을 절약할 수 있도록 고안하기

RAD스튜디오의 큰 장점 또 한 가지는 바로 인력 부담을 줄일 수 있다는 점이다. 업데이트 대상인 기술요소와 코드를 더 적게 유지하기 때문에 결국 시간과 비용을 크게 절감할 수 있다. 특히 동시에 여러 플랫폼을 개발하고 배포하려고 할 때는 절감 효과가 더욱 크다. 이런 이유로 **RAD스튜디오**에 대한 관심 역시 급격하게 커졌다. 2017년에 일주일간 온라인 컨퍼런스 형태로 열린 사물인터넷 부트 캠프에는 180 여개국의 개발자들이 등록했다. 온라인 자료들 역시 **LearnDelphi.org** 와 **EmbarcaderoAcademy.com** 등 많이 있다.

전세계적으로 **RAD스튜디오** 개발자 커뮤니티가 크긴 하지만, 특정 영역에서는 개발자가 가끔 모자라기도 한다. 이런 상황을 해소하기 위해 점점 더 많은 회사들이 C# 개발자를 고용하고 **RAD스튜디오**를 익히도록 하여 향상시키고 있다. C#개발자라면 일반적으로 2주에서 4주 정도면 **RAD스튜디오**와 해당 프레임워크를 매우 편안하게 사용할 수 있게 된다 (물론 개발자마다 차이가 있긴 하다). C#은 **델파이**를 만든 앤더스 헤즐스버그가 만들었는데, C# 방식이 **델파이**를 빼닮는 데 큰 영향을 끼쳤다. 이런 점이 고려하면, 상당히 현실성이 높은 방안이다.

RAD스튜디오의 아키텍처는 오랜 시간 동안 증명되었으며 혁신적인 크로스 플랫폼 솔루션 분야에서 계속 앞서가고 있다. OOP를 가장 잘 반영한 검증된 여러 방식을 기반으로 컴포넌트 모델이 설계된 덕분에 코드를 현대화하기가 쉽고, 최소한의 노력으로 시장 변화의 장점을 취할 수 있어서 장기 투자 측면에서 ROI가 매우 탁월하다. 기존 개발자가 **델파이**를 익히는 것이 매우 쉽고, **RAD 개발자는 다른 프레임워크 개발자에 비해 5배 이상 생산적**이기 때문에 RAD스튜디오는 팀의 규모와 관계없이 장기 생산성 면에서도 탁월하다.

두 세상에서 최고가 되기 – 왜 RAD스튜디오인가

크로스 플랫폼 개발 도구와 방법의 진화

1부에서는 지난 25년간 소프트웨어 개발에 영향을 끼친 혁신들을 살펴보았다. 새로 떠오른 기술은 대체로 초기 성장기의 지속적인 변화를 겪고 표준으로 자리잡고 성숙하기까지 대략 10년 정도가 걸린다. 어떤 기술이든 초기에는 높은 위험을 감수해야 하므로, 장기적인 솔루션을 제공하려는 입장에서 처음에는 일단 위험이 낮은 방식을 선호하게 된다. 이런 초기 시장 마인드는 새 장비를 커버하기 쉽다는 장점이 있는 웹 중심 솔루션의 물결을 일으켰다. 왠만한 장비에 이미 있는 웹 브라우저를 이용하면 빠르게 시장을 테스트하고 니즈를 확립할 수 있기 때문이었다.

RAD스튜디오가 웹 개발을 지원하기 시작한 것은 1990년대에 인트라웹(IntraWeb)이 추가되면서 부터이다. 개발자들이 인트라웹으로 전화기와 PDA에 있던 초창기 브라우저 등 여러 웹 형식을 개발할 수 있었다. (혹시 WAP를 기억하는 사람이 있는가?) 그 후 웹 표준이 향상되고, 자바스크립트의 인기가 높아지면서, 데스크톱 브라우저의 코드 표준에 훨씬 더 가깝고, 더 빠르게 개발할 수 있고, 멀티 디바이스 지원을 제공하는 라이브러리들이 출현했다.

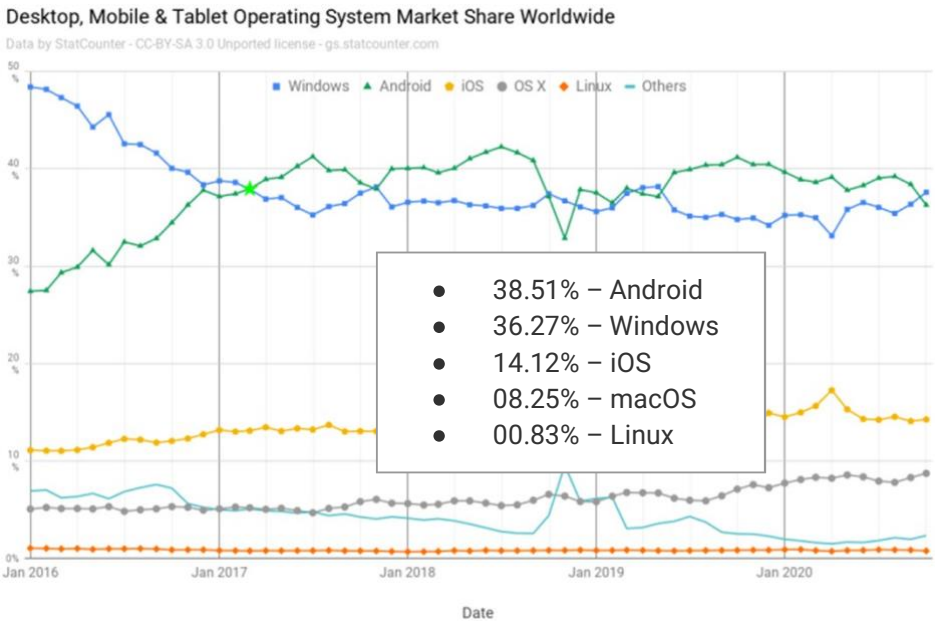
RAD스튜디오에서 **인트라웹**, **TMS 웹코어**와 같이 **델파이**에서 사용할 수 있는 웹용 컴포넌트들이 다양해지는 한편, HTML과 자바스크립트용 독립 프레임워크들 역시 많이 생겨났다. 자바스크립트를 기반으로 RAD에로 접근하는 선두 사례는 **센차(Sencha)**이다. 센차는 **자바스크립트 기반의 웹 컴포넌트 중 최고급**에 속한다. 속성을 지정하고 연결하는 RAD 방식을 사용하여, 기능이 풍부한 웹 애플리케이션을 빠르게 개발할 수 있다. 센차는 오라클 등 전세계의 많은 솔루션에서 사용되거나 내장되어 있으며, 아이데라 그룹에 합류된 이후, 현재 **RAD스튜디오 아키텍트** 에디션에도 들어있다.

RAD스튜디오 사용자는 웹 언어로 개발할 수도 있고 (웹 코드보다 더 빠르게 실행되는 앱을 만드는) 컴파일 되는 네이티브 코드로도 개발할 수 있는 유연성을 가질 수 있다. 하지만, 시장에는 다른 대안들도 많다. 각 방식별로 차이점을 정리해보고,아울러 필수 플랫폼 별로 각 시장을 살펴보자.

모바일이 비즈니스 프로세스에 미치는 영향

모바일은 비즈니스 애플리케이션 용으로는 메인이 아니긴 하지만 원격 데이터 활용하는 능력의 장점을 취하려는 **BPR**의 새 트렌드로 인해 집중 조명을 받고 있다. 자동화가 더욱 증가하는 추세 속에서 전통적으로 본사에서 완료되던 작업들이 이제는 현장 근무자에게로 분산되어 문서 작업은 줄어들고 프로세스 흐름은 짧아졌다. 예를 들어, 현장에 방문한 엔지니어가 작업 내역과 사진 기록까지 그자리에서 모두 완료한다. 게다가 모바일 서비스는 위치 정보와 같은 중요한 데이터를 추가로 확보할 수 있도록 해준다.

BPR의 이런한 새 트렌드를 실현하기 위해 채택된 BYOD 같은 정책은 사용자가 원하는 장비를 쓸 수 있도록 하는 것이 주된 목적이었다. 하지만 다른 한편으로, 작업에 필요한 장비를 제공하는데 걸리는 시간과 비용 등을 줄이고자 하는 회사들 역시 채택했는데, 스마트폰이 시장에서 널리 퍼지면서 가능해졌다.



Operating System Market Share as of November 2020

Source: https://gs.statcounter.com/os-market-share

(그림2. 주요 운영체제의 시장 점유율, 2020년 11월 기준)

현재 안드로이드 사용자가 iOS 사용자 보다 훨씬 많지만, 이 두 플랫폼 모두 널리 사용되고 있다. 여기에 BYOD가 맞물린 결과, 소프트웨어를 성공적으로 출시하려면 두 플랫폼 중 어느 하나도 버릴 수 없는 것이 현실다. 게다가 이 두 플랫폼의 디자인 가이드가 서로 다르기 때문에 문제는 더 복잡해 진다. 소프트웨어가 성공하려면 각 플랫폼의 디자인 가이드 지켜야 한다. 그렇지 않을 경우 해당 플랫폼 사용자는 바로 어색함 느끼고 거부감을 가지게되므로 새 소프트웨어가 성공할 활률은 낮아진다.

요컨데, 오늘날 모바일 중심 BPR 능력을 확보하려면, 소프트웨어 엔지니어가 **iOS와 안드로이드를 모두 동시에 다룰 수 있어야 한다.** 그리고 앱의 룩앤필은 각 플랫폼의 특성에 맞아야 한다.

멀티 플랫폼 달성

웹 기술을 이용하면 새로운 장비와 새로운 플랫폼을 빠르게 수용할 수 있긴하지만, 이 방법은 웹 브라우저 안의 기능 수준까지만 구현할 수 있다는 제약이 있기 때문에 결국 BPR에서 요구하는 능력을 확보하는데 한계가 있다. 웹 페이지를 제공하는 방식은 속도 또한 느리기 때문에 사용자 경험(UX) 수준이 낮다. 최상의 신뢰성, 속도, UX를 제공하고, (특히 보안 권한이 요구되는 곳에서도) 장비의 기능에 접근할 수 있으려면 디바이스에 설치되는 애플리케이션이 필요하다.

설치형 모바일 애플리케이션을 만드는 방식은 두 가지이다. 하나는 완전히 컴파일된 네이티브 방식이고 다른 하나는 장비의 기능에 액세스할 수 있는 브라우저 셸(Shell) 안에서 웹앱을 실행하는 하이브리드 앱 방식이다.

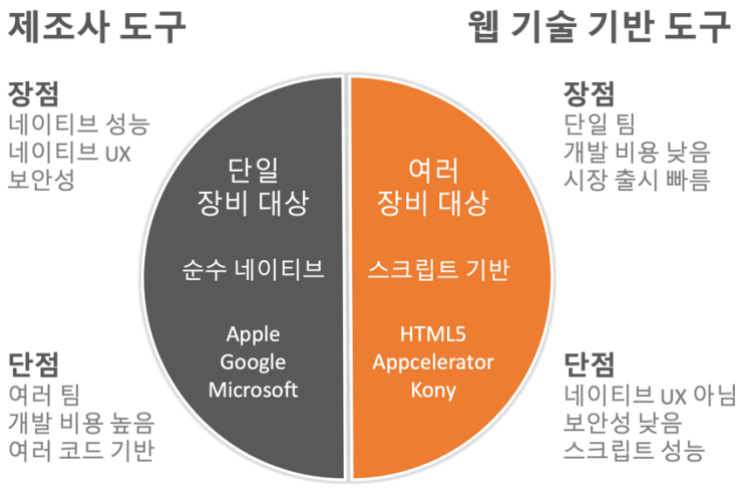
순수 네이티브 대 하이브리드 애플리케이션

순수 네이티브 코드는 전형적으로 플랫폼 제조사에서 제공하는 Xcode, 비주얼스튜디오, 안드로이드 스튜디오 등의 도구를 사용한다. 순수 네이티브 개발은 네이티브한 UX, 네이티브한 속도와 성능, 코드가 완전히 바이너리로 컴파일 됨에 따른 수준높은 보안이 장점이다. 하지만, 이러한 네이티브의 장점을 취하고자 제조사 도구를 사용하면 각 플랫폼 별로 개발과 관리를 따로 해야하기 때문 코드 기반이 여러개가 되고, 개발자와 기술요소 역시 더 많아지게 된다. 이 모두는 더 높은 개발 비용으로 이어진다.

시간, 공정, 비용 부담으로 인해 많은 개발자들이 스크립트 방식 즉 하이브리드 앱을 추구한다. 하이브리드 방식은 단일 팀에서 애플리케이션 하나를 개발하면 되기 때문에 더 낮은 비용으로 여러 플랫폼을 동시에 커버할 수 있다. 하지만, 보안, 성능, 네이티브 UX 등을 희생해야 한다.

스크립트 방식 즉 하이브리드 앱 방식이 **성능이 낮고 보안이 취약한 이유**는 앱 실행 중에 스크립트 언어가 해석되어야 하기 때문이다. 실행 중에 코드를 해석하는 방식은 성능 상 병목이 되기도 하고, 해커가 장난칠 수 있는 보안 상 잠재적인 약점을 가지고 있다. 게다가 기본적으로 웹 애플리케이션이므로 네이티브 애플리케이션보다 훨씬 많은 메모리가 필요하다.

마지막으로, 닷넷이나 자바스크립트 런타임 웹키트 또는 자바런타임(JavaRuntime) 등의 런타임을 사용하는 애플리케이션도 있다. 이 경우에는 사용하는 런타임 자체로 인한 추가적인 보안 위협이 있다. 게다가 이들 런타임의 메모리 부하도 상당하여 스스로 메모리를 관리하기 위해 가비지 컬렉터가 들어 있다.

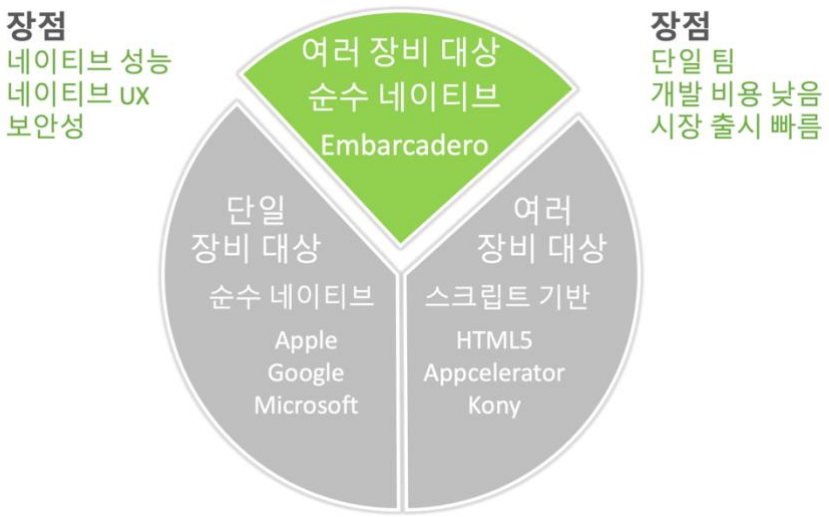


(그림3. 제조사 도구와 웹 기술 기반 도구의 장단점)

타협 하지 않기 - 크로스 플랫폼과 순수 네이티브 모두에서 최고를 추구하기

RAD스튜디오 만의 방식은 크로스 플랫폼 개발 시 타협할 필요가 없다. 개발팀 하나로 모든 플랫폼을 동시에 커버하면서도, 이와 동시에, 완전하게 컴파일되는 순수 네이티브 애플리케이션을 개발하여 자연스러운 UX, 매우 빠른 성능, 회고 수준의 보안을 제공한다.

RAD스튜디오에서 LLVM을 사용하여 컴파일하면, 각 플랫폼에 최적화된 애플리케이션이 생성된다. RAD스튜디오는 iOS와 맥OS용 앱을 만들 때에는 Xcode에서 사용되는 것과 동일한 컴파일러를 사용한다. 안드로이드용 앱을 만들 때에는 자바보다 더 낮은 수준으로 컴파일하여 게임 개발자과 마찬가지로 CPU와 GPU를 바로 접근한다. 가비지 컬렉터를 이용하여 메모리를 부풀릴 필요도 물론 없다.



(그림4. 멀티 디바이스 개발과 순수 네이티브 개발의 모든 장점을 취하기)

FMX 프레임워크는 2011년 9월, 델파이 XE2에서 처음 발표된 이후 계속 성숙하여, 단일 코드 기반으로 순수 네이티브 UX의 성능과 보안을 제공한다. 콘트롤에 있는 Platform Default 속성을 사용하면 손쉽게 해당 플랫폼의 고유한 룩앤필을 바로 만들 수 있다. 물론 개발자는 필요한 곳을 얼마든지 커스터마이징할 수도 있다. **FireUI**와 같은 생산성 기능을 활용하면 개발자가 UI 디자인과 작동을 (어떤 장비에서도) 즉시 볼 수 있어서 개발 시간이 훨씬 줄어든다.

컴파일러의 공통 아키텍처를 사용하는 덕분에, 언어와 프레임워크의 능력 전체를 모든 플랫폼에서 지원할 수 있다. **RAD스튜디오**의 FMX 코드는 윈도우, 맥OS, 리눅스, 안드로이드, iOS에서 실행되면서도 개발 생산성 수준은 더 높다.

또한 **RAD스튜디오**는 IDE 안에서 바로 앱스토어 용 앱을 공증하고 배포할 수 있어서 전반적인 빌드 절차가 단순화된다.

FMX가 자마린 폼즈 (Xamarin Forms)와 다른 점

2014년 5월에 발표된 **Xamarin Forms**는 FMX와 유사한 시도를 했는데, 언어는 XAML을 사용했다. Xamarin은 크로스 플랫폼 개발용 콘트롤의 하위 집합을 개발자에게 제공한다. 하지만, **RAD스튜디오**와 비교하면 Xamarin은 2가지 중요한 영역에서 뒤떨어진다.

첫째, **플랫폼에 따라 언어의 능력이 제한적**이다. (예를 들어 iOS에서는 제네릭스를 사용할 수 없다. 즉 각 플랫폼 별로 코드 작성을 따로해야 하는 경우가 종종 발생한다). 둘째, 자마린은 닷넷 기반이므로 **비효율적인 메모리 관리**로 고생한다. 둘째, Xamarin은 닷넷을 기반으로 하기 때문에, iOS에 닷넷 가비지 컬렉터가 추가된다. 안드로이드에서는 (닷넷 및 자바 둘 다) 있으면서 자바 가비지 컬렉터가 실행된다. 이처럼 Xamarin의 메모리 관리는 효율적이지 않다.

엔터프라이즈 데이터와 원격 데이터 연결

RAD스튜디오가 제공하는 강력한 데스크톱과 모바일 앱을 개발 능력은 훌륭하지만, 완전한 개발 환경이 되려면 원격 장비로 데이터를 제공할 수 있어야 한다.

해를 거듭하면서 **RAD스튜디오**에는 데이터를 원격에서 연결하는 여러 방식들이 도입되었다. 현재 파이어닥 컴포넌트는 데이터 연결 시 로컬과 원격 모두를 지원하며, 전형적인 SQL과 NoSQL 데이터베이스 역시 15 종류 이상을 지원한다. (직접 지원하지 않는 경우를 위해 ODBC도 들어있다). 또한 파트너사에 의해 확장되면서 (Jira, Salesforce, Microsoft Teams, Google Drive, eBay, Facebook, Slack, Twitter, Amazon Marketplace 등등) **엔터프라이즈 시스템과 빅데이터 시스템 180종 이상**을 표준 SQL을 통해 직접 연결하는 기능이 제공되는데, 이 중 상당수는 엔터프라이즈 에디션에 들어있다.

여러 데이터베이스와 많은 엔터프라이즈 데이터 소스에 빠르게 연결할 수 있는 능력 덕분에, **RAD스튜디오**를 이용하여 이상적인 미들 티어를 만들 수 있다**. 웹서비스**, (마이다스를 기반으로 세션의 기초 연결을 실현한) **데이터스냅**, (도커, Docker, 구성을 제공하고, 사용성 분석과 리포트 기능이 내장되어 있으며, 전적으로 RESTful 기반 MEAP인) **RAD 서버** 중 알맞은 방식을 사용하여 여러 시스템을 통합하고, 데이터를 빠르게 노출할 수 있다. 종종 코드 작성을 전혀 할 필요가 없기도 하다.

코드가 적은 로우 코드 (Low-Code) 애플리케이션 플랫폼과 RAD

가트너(Gartner)의 정의에 따르면, “로우 코드 애플리케이션 플랫폼 (LCAP)은 애플리케이션 개발, 실행, 관리를 빠르게 할 수 있도록 지원하는 애플리케이션 플랫폼으로써, 모델 중심이나 메타데이터 기반 프로그래밍 언어와 같이 선언적이고, 매우 높은 수준의 프로그래밍 추상화를 사용하고, 배포는 원스텝으로 완료된다. LCAP은 UI, 비즈니스 프로세스, 데이터 서비스를 제공하고 지원한다.”

LCAP에 대해 관심이 커진 주된 이유는 비즈니스 프로세스 검토 (BPR, Business Process Review)와 변화하는 비즈니스를 지원하기 위한 앱 제작에 집중하게 되면서, 소프트웨어를 통한 비즈니스 자동화 요구가 증가했기 때문이다. 신속한 배포와 크로스-플랫폼 지원을 실현하기 위해 LCAP은 주로 웹 기술 기반인 하이브리드 앱 형태를 사용하여 소프트웨어를 제공한다.

로우 코드가 시장에서 팔릴 때는 종종 “시민 개발자”가 가능하다는 약속을 기반으로 한다. 즉 조금만 교육받으면 누구나 앱을 만들 수 있다는 약속이다. 이론 상 좋기는 하지만, 현실에서는 맞춤이 필요한 경우 **개발 범위가 제한적**이고 **학습곡선도 매우 높다**. 그리고 써드 파티와 연동하고 관리하려면 **LCAP 제조사에 크게 의존**해야 한다. 몇몇 시스템은 단순 작업 이상을 할 수 있도록 더욱 숙련된 개발자용으로 웹 기반 고급 RAD 도구를 제공하기도 하는데 일단 해당 LCAP 전용 프레임워크와 모델을 익혀야 사용할 수 있다. 주의 사항! 명확한 애플리케이션 전략이 없이 시민 개발자 방식을 적용하면 비즈니스 소프트웨어가 조각조각 나누어지기 쉽기 때문에 관리나 변경이 어려운 장기 기술 채무가 발생한다.

많은 로우 코드 플랫폼의 경우, 앱을 사용하는 사용자당 **구독비용**을 지속적으로 내는 방식이므로 **비용이 결국 상당히 커진다**. 게다가 가트너의 주의 사항에 따르면, 실제로 많은 로우 코드 제조사들은 전문 서비스를 통해 상당한 매출을 올린다. 즉 해당 도구를 지원할 때 필요한 인력은 시민 개발자가 아니라 해당 프레임워크를 잘 아는 전문 개발자일 수도 있다는 의미이다. 결국 LCAP 총소유비용이 최초 예상보다 현격하게 더 높은 경우가 대부분이다.

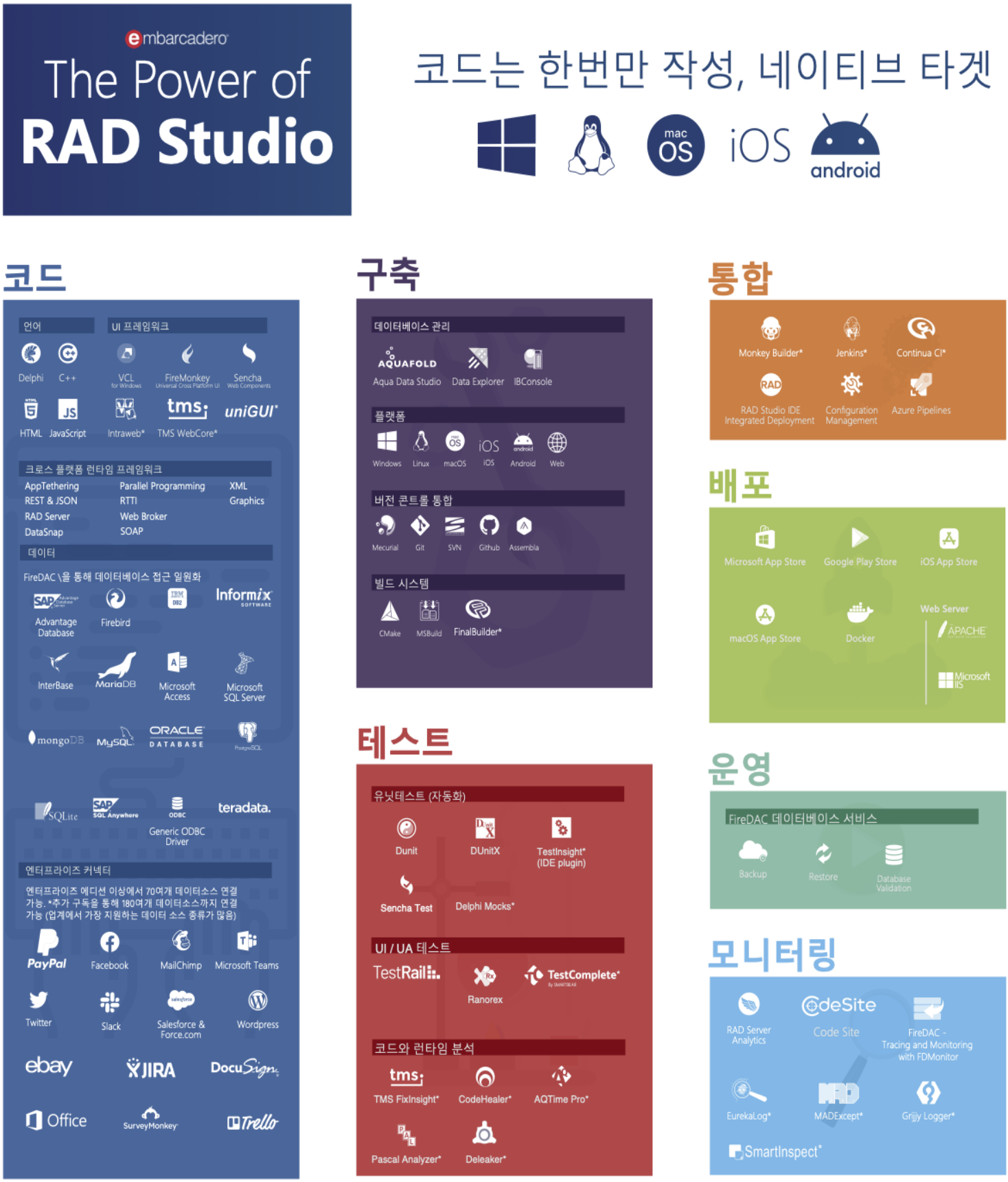
지난 수십년 동안 **RAD스튜디오**는 비즈니스 프로세스 검토, 신속한 애플리케이션 프로토타입, 필요한 것이면 무엇이든 붙일 수 있는 오픈 에코시스템 등을 위한 로우 코드 개발과 지원을 제공해왔다. LCAP를 찾고 있다면 투자 대상으로 어느 것이 더 좋을 지 판단해보기 바란다. 특정 LCAP 제공자에게만 속한 틈새 기술을 추가로 익히는 것과 기존의 팀이 **델파이**를 쓸 수 있도록 하는 것 중 어느 것이 더 좋은 투자일까? 델파이는 더 보안성이 높은 네이티브 애플리케이션을 더 빠르게 제공하고 모든 주요 플랫폼을 커버할 수 있다. 많은 사람들이 **RAD스튜디오**를 고려하게 된 이유이기도 하다.

RAD스튜디오 현재 – 미래를 위한 투자

RAD와 데브옵스(DevOps)

현재 개발 프로세스 강화 부문에서 업계 트렌드는 데브옵스(DevOps)인데, 성공적인 결과를 만들려면 개발팀과 배포팀이 함께 작업할 필요가 있다는 점을 이해하게 되면서 시작된 배포 구성과 설정을 둘러싼 혁신이다.

데브옵스의 초점은 그 자체가 기술백서가 되는 것이다. 하지만 가끔은 그림 한장이 훨씬 많은 말을 대신하기도 한다. 이 문서 앞부분에 이미 다룬 바와 같이, **RAD스튜디오**와 연결되는 오픈 에코시스템이 잘 형성되어 있기 때문에, 개발 전 과정에 걸쳐 다양한 개발자 생산성 도구들이 사용될 수 있다. 다음 페이지에 있는 **"Power of RAD Studio"** 인포그래픽에서는 이런 도구들 중 몇 가지를 뽑아서 한눈에 보여준다.



(그림5. RAD스튜디오와 데브옵스 인포그래픽)

[인포그래픽] The Power of RAD-Studio PDF파일로 다운로드하기

현재와 미래 모두를 위해 투자하기

어떤 개발 도구 안에 코드를 넣는 것은 일종의 투자이다. 장기적인 투자 대비 수익 면에서 **RAD스튜디오**는 최고라고 할 수 있으며, RAD스튜디오는 지금껏 오래된 버전에서 최신 버전으로의 코드 이식 보장에 대한 집중을 계속 지켜오고 있다.

정기적으로 업데이트되어 **엠바카데로** 커뮤니티에 공유되고 있는 향후 로드맵에서는 애플 실리콘, 마이크로소프트 MSIX 패키징, 앱스토어 변화를 둘러싼 최신 혁신들을 수용하고 있는 것을 확인할 수 있다.

요약

이 문서에서 살펴본 바와 같이, 애플리케이션을 빠르게 개발하고자 하는 바램은 더욱 커지고 있고, 하이브리드와 네이티브 개발 시장에는 많은 선택지가 있다. **RAD스튜디오** 만의 순수 네이티브 방식은 네이티브 전용 도구 제조사들이 사용하는 것과 동일한 컴파일러 기술을 이용하여 가장 빠른 속도, 성능, 보안성을 제공하면서도 크로스 플랫폼용 단일 코드 기반이라는 가치를 동시에 제공한다.

RAD스튜디오는 시장 트렌드를 지속적으로 수용하고 개발해왔다. (요즘 가장 중요하게 여겨지는) 코드 보안을 제공하고 소프트웨어 개발 투자에 대해서 가능한 최고의 ROI를 보장한다.

주요 트렌드를 파악하고 대응하면서, **RAD스튜디오**는 미들 티어와 마이크로서비스를 RAD 방식으로 개발하기, 주요 데스크톱과 모바일 앱 스토어에 빠르게 배포하기, 주요 엔터프라이즈 시스템들 뿐만 아니라 각종 SQL과 NoSQL에 광범위하게 연결하기 등을 실현했다.

윈도우, 맥OS, iOS, 안드로이드, 리눅스로 확장하기 위해 다른 것을 잃을 필요가 없다는 점이 이미 증명된 프레임워크를 제공하는, **RAD스튜디오** 만이 시장에 제공할 수 있는 개발 능력은 **RAD스튜디오**로 만든 코드의 미래가 밝고 안전하다는 것을 확신할 수 있다.

이 문서 주제 밖의 참고 사항

이 문서의 주제에 딱 맞지는 않지만, RAD스튜디오에 최근 몇년 간 추가된 개발자의 생산성 향상 기능들도 살펴볼만한 가치가 있다.

RAD스튜디오에 현재 들어있는 컴파일러들로는 윈도우(32비트, 64비트), 리눅스(64비트), 안드로이드 (32비트, 64비트), iOS(64비트), 맥OS(64비트)용 네이티브 앱을 개발할 수 있다. 모든 컴파일은 단일 소스 코드에서 이루어지므로 모든 주요 플랫폼에서 테스트와 관리 비용이 절감된다.

최신 IDE를 사용 중이라면, 다음과 같은 많은 개발자 생산성 업데이트의 혜택을 받을 수 있다.

- **소스 코드 저장소 연결 지원 내장**: 깃, SVN, Mercurial
- **현대화된 IDE 레이아웃 구성**과 다크 모드
- **더 큰 메모리 지원**: 훨씬 더 큰 프로젝트를 컴파일
- **LSP (Language Server Protocol) 지원**: 코드 자동 완성, 에러 표시 작업 등이 백그라운드에서 실행되므로 더 빠르게 코딩
- 등등…

더 자세한 내용을 알고 싶으면 언제든 데브기어로 문의

엠바카데로 방식

단일 소스, 네이티브 멀티 플랫폼 소프트웨어 개발

RAD스튜디오의 IDE와 프레임워크는 현대식 델파이와 현대식 C++ 언어를 사용하여 코드를 작성하고, 작성된 단일 코드를 기반으로 컴파일하여 윈도우, 리눅스, 맥OS, iOS, 안드로이드용 네이티브 앱을 생성한다.

Design it, Build it, Run it!