developer Works

프레임워크 설계 및 구축

애플리케이션과 프레임워크 동시 개발

프레임워크가 좋다는 건 알지만 언제 어떨 때 만들어야 할까? 애플리케이션은 프레임워크가 있으면 만들기가 한결 수월하다. 프레임워크도 애플리케이션이 이미 만들어져 있으면 좀 더 쉽게 접근할 수 있다. 하지만 둘 다 없으면 어떻게 해야 될까? 이 글에서는 이에 대한 고민을 해결해 본다.

2

고 상 원 , 장 진 호 , 전 제 민 , 손 영 수
http://fdg.springnote.com | IT 생태계의 먹이사슬 구조를 벗어난 개발자가 되고 싶어 한다. 함께 프레임워크를 공부하고 있으며 현재 「Framework Guide Lines 2nd Edition」을 번역하고 있다. 프레임워크를 공부하면서 정리한 자료를 웹에서 공유하고 있다.

우리 생활을 편하게 하는 놀라운 물건이 나오면, 그걸 본 사람들이 농담 삼아서 인간이 게을러서 발명한다고 얘기하곤 한다. 오랜 시간 동안 인류는 조금이라도 쉽고 편하게 일을 할 수 있는 방법을 찾아서 적용하고 발전시키려 노력했었다.

프로그램을 개발하는 일도 예외일 수 없다. 재사용이라는 강력한 무기를 들고 나타난 객체지향이라는 패러다임은 많은 개발자들을 유혹했다. 프로그램을 만들 때마다 이전에 만들었던 모듈을 다시 만들거나, 기존의 소스 코드를 수정해 사용해야 했던 일이 많았던 개발자들이 재사용이란 달콤한 열매에 열광하는 건 '쉽고 편하게'를 추구하는 게으른 인간의 입장을 생각하면 당연한 일이었다. 이후로도 재사용을 돕는 새로운 기술들이 나오고 시도되고 사라져갔다.

요즘 개발자들이 재사용이란 달콤한 열매를 맛볼 수 있는 방법 중 하나는 프레임워크를 이용하는 것이다. 프레임워크는 단순히 소스 코드를 재사용하는 것에 그치지 않고 프레임워크의 구조와 추상화 수준을 개발자들이 그대로 이어받도록 한다. 프레임워크를 구축해 두면 비슷한 애플리케이션들을 만들 때마다 재사용할 수 있다. 프레임워크 덕분에 애플리케이션을 만드는 데 들어가는 비용도 줄어들게 된다.

값 비싸고 어려운 프레임워크

매번 비슷한 애플리케이션을 개발하다 보면 누구나 한 번쯤은 프레임워크를 떠올리게 된다. 프레임워크가 주는 재사용이란 혜택은 애플리케이션 개발자가 복잡한 문제를 해결하느라 머리를 싸매고 고민하는 시간을 줄여준다. 프레임워크를 사용한다는 것은 단순히 소스 코드로 구현된 기능들을 재사용하는 것뿐만 아니라, 프레임워크의 추상화 수준과 구조(design)까지 재사용하는 것을 의미한다. 프레임워크가 만들어둔 길을 따라가면서 필요한 부분을 확장하고 제공된 기능들을 가져다 쓰면 된다. 프레임워크를 쓰는 초보 개발자는 복잡한 기능을 손쉽게 적용하기도 한다. 하지만 동전에도 양면이 있는 것처럼 위에서 설명한 프레임워크의 이점들 이면에는 프레임워크 개발자들이 이겨낸 산고의 고통이 숨어 있다. 애플리케이션 개발자가 원하는 모든 것들을 마법처럼 뚝딱 제공해 줄 만큼 완벽한 프레임워크는 현실적으로 만들기 힘들다. 모두의 요구사항을 고려해서, 누구에게나 적용될 수 있는 추상화 수준을 찾아서 기능들을 일반화시키는 것은 쉽지 않다. 프레임워크를 구축하다가 자칫 지나치게 일반화되거나 매우 복잡해져서 이해하기 힘들어지는 일이 생기기 쉽다. 거기다 프레임워크를 구축하는 데 필요한 비용은 애플리케이션보다 훨씬 크다.

프레임워크와 애플리케이션을 동시 구축해야 되는 상황

일반적으로 프레임워크는 비슷한 애플리케이션을 개발하고 나서 구축하는 것이 좋다. 애플리케이션을 개발하면서 충분히 요구사항을 분석하고 설계하고 구현하면서 겪은 시행착오를 바탕으로 프레임워크에서 제공해야 될 기능과 제어 흐름(Control flow) 등을 충분히 도출해낼 수 있기 때문이다. 여러 애플리케이션을 대상으로 만들어져야 하는 프레임워크 입장에서는 유저인 애플리케이션 개발자들의 요구사항을 분석하는 것만큼 중요한 일이 없다. 따라서 앞서 경험했던 애플리케이션 개발 경험은 프레임워크 개발자에게 명확한 요구사항을 알려줄 수 있다(마소 2008년 12월호 '프레임워크 구축과 발전 - 세 가지 예제 Three Examples 패턴' 참고).

항상 위와 같은 상황에서만 프레임워크가 만들어지는 것은 아니다. 예를 하나 들어보자. 새롭게 시작하는 대규모 프로젝트에는 여러 팀들이 참여해 동시에 각자 애플리케이션을 만들어야 한다. 하지만 투입 인원과 규모가 큰 대규모 프로젝트라 하더라도 특정 도메인에 대한 프로젝트인 경우가 많다. 예를 들어 보험회사의 전산 시스템을 새롭게 구축하는 상황이라면, 각각의 팀들은 건강보험 시스템, 생명보험 시스템, 손해보험 시스템 등을 맡아서 하게 될 것이다. 물론 그 외에도 고객정보 관리시스템이나 지불시스템 등을 맡아서 하는 팀도 있을 것이다. 이처럼 비슷한 성향의 시스템을 구축해야 되는 상황이라면 프로젝트가 시작되기 전이라도 누구나 이런 시스템들은 잠재적으로 재사용될 만한 부분이 많을 것이라고 예측할 수 있다.

이런 상황에서 프로젝트에 참여한 팀들이 중복해서 같은 일을 하지 않으려면 어떻게 해야 할까? 가장 쉽게 생각해 볼 수 있는 방법은 프로젝트에 참여한 팀 중에서 한 팀이 재사용될 모듈들을 구현해 다른 팀들이 사용하도록 나눠주는 것이다. 단순히 날짜나 알려주는 모듈부터 데이터베이스에 접근해야 하는 복잡한 모듈까지 어떤 모듈이든 프로젝트에 참여한 팀들이 필요로 한다면 한 팀이 먼저 만들어서 다른 팀에게 제공하여 중복 작업을 줄일 수 있다. 위와 같은 상황처럼 여러 팀들이 공유하는 기능을 제공하는 경우엔 프레임워크도 적임자 중 하나라고 할 수 있다. 프레임워크가 이와 같은 프로젝트에서 유용하게 사용될 것이란 확신이 생긴다면 여러분은 프레임워크와 애플리케이션의 구축을 병렬적으로 진행해야 한다.

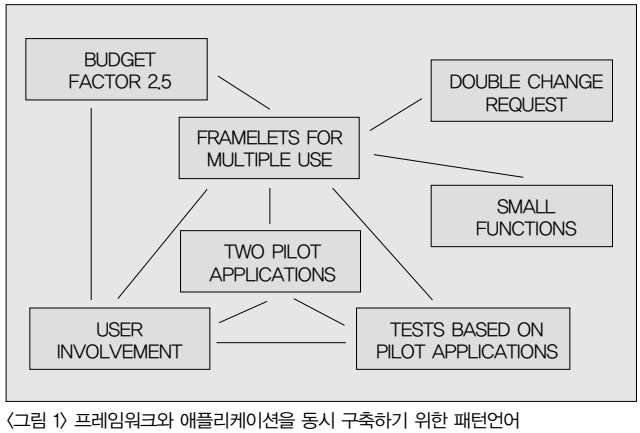
물론, 프레임워크가 적임자라고 해서 쉽게 구축할 수 있단 얘기는 아니다. 프레임워크 개발팀은 프레임워크를 구축하려면 프레임워크를 사용할 애플리케이션들에서 대해 충분히 이해하고 애플리케이션 개발팀의 요구사항을 잘 수렴해야 한다. 그래야만 프레임워크는 애플리케이션을 아직 구현하지 않은 애플리케이션 개발팀이 군침을 흘릴만한 기능들을 제공할 수 있다. 하지만 프레임워크를 개발하는 시점에선 아직 애플리케이션이 존재하지 않는다. 애플리케이션이 존재하지 않는데 애플리케이션을 이해하고 애플리케이션 개발팀의 요구사항을 충분히 알 수 있을까?

애플리케이션 개발팀의 입장을 살펴보자. 애플리케이션 개발팀은 쓸 만한 프레임워크가 나와 주길 기다리고 있다. 애플리케이션 개발팀 입장에선 불필요하게 중복된 일을 할 필요가 없으니 좋긴 하지만, 프레임워크 개발팀에선 최소한 세 개의 애플리케이션(Three examples)이 있어야 괜찮은 프레임워크를 만들 수 있다고 한다. 그렇다고 프레임워크 개발팀이 세 개의 애플리케이션을 만들 때까지 놀고먹을 순 없다. 애플리케이션 개발팀도 프로젝트를 진행해야 되고, 프레임워크 개발팀의 프레임워크도 도입해야 한다.

프레임워크를 구축하고 발전시키는 대부분의 패턴들은 프레임워크를 구축하기 이전에 애플리케이션에 대해 충분한 경험을 가지고 있어야 한다고 얘기한다. 애플리케이션과 프레임워크를 동시에 구축하는 것처럼 애플리케이션에 대한 경험이 없는 상황에서는 프레임워크를 설계하기가 매우 힘들다. 프레임워크는 애플리케이션 개발자가 원하는 기능을 제공하면서, 복잡하지 않게 설계되어야 한다. 이런 상황을 어떻게 극복해나갈 수 있을까?

프레임워크와 애플리케이션을 동시 구축하기 위한 패턴언어

2000년도에 열린 PLoP(Pattern Language of Programs) 학회에 프레임워크와 애플리케이션을 동시에 구축하는 패턴언어가 소개되었다. 총 7개의 패턴으로 구성된 패턴언어로 프레임워크와 애플리케이션을 동시에 구축하면서 발생하기 쉬운 몇몇 문제점에 대한 해결책들을 제안하고 있다. 이 패턴언어의 패턴들은 기존의 프레임워크 구축에 대한 패턴들과 유사하다. 다시 말해 기존의 패턴들을 애플리케이션과 프레임워크를 병렬적으로 구축해야 되는 상황에 맞게 적절히 변형해 적용하였다.



〈그림 1〉 프레임워크와 애플리케이션을 동시 구축하기 위한 패턴언어

프레임워크와 애플리케이션을 동시에 구축하기 위한 패턴 언어는 다음과 같이 구성되어 있다.

- Framelets for Multiple Use
- Budget Factor 2.5
- Two Pilot Applications
- Small Functions
- User Involvement
- Tests Based on Pilot Applications
- Double Change Request

패턴은 프로그래밍을 하면서 발생하는 문제만 다루지 않는다. 대인관계나 UI와 같은 프로그래밍 외적인 문제에 대한 해결책들도 패턴이 될 수 있다. 위에 나열한 7개의 패턴도 마찬가지다. 프레임워크와 애플리케이션을 동시에 구축하는 프로젝트를 진행하면서 발생하는 다양한 문제를 다루는 패턴들이다. 각 패턴에 대해 하나씩 짚어보자.

Framelets for Multiple Use

● 애플리케이션과 프레임워크를 동시에 구축하는 것을 어떻게 합리화할 수 있을까?

먼저 프레임렛(Framelet)이라는 생소한 단어부터 살펴보자. 프레임렛은 작은 프레임워크를 뜻한다. 프레임워크라고 해서 항상 애플리케이션을 전체적으로 커버하고 있진 않다. 여러 애플리케이션에게 부분적으로 공통된 부분을 프레임렛이라는 작은 프레임워크로 만들 수 있다. 예를 들어 프로젝트의 여러 애플리케이션이 데이터베이스에 접근해야 된다면, 데이터베이스에 대한 부분을 계층화시켜 하나의 독립적이고 재사용 가능한 계층(layer)으로 만드는 것이다. 그럼 Framelets for Multiple Use는 어떤 상황에 적용되는지 알아보자.

우린 프로젝트를 시작하면서 개발해야 될 시스템에 잠재적으로 재사용되는 부분이 있다는 것을 어느 정도 짐작할 수 있다. 즉 프로젝트에서 개발되어야 할 여러 개의 애플리케이션들이 비슷한 기능을 필요로 하고 있다. 각각의 애플리케이션이 비슷한 기능을 각자 개발한다고 가정해 보자. 아마도 각각의 애플리케이션들은 그 기능을 개발팀마다 다르게 적당히 추상화시키고 성능이나 객체간의 관계, 확장성 등을 고려해서 애플리케이션에 특화되도록 개발할 것이다. 비슷한 기능이 애플리케이션마다 조금씩 다르게 적용되는 건 당연한 일이지만, 불필요하게 중복해 비용이 투자되어야 된다는 단점이 있다. 그렇다고 무작정 재사용성을 위해 애플리케이션들이 원하는 비슷한 기능에서 공통된 추상화(Common Abstraction)를 찾아 애플리케이션들이 재사용할 수 있게 인터페이스나 클래스를 제공해야 된다는 건 아니다.

재사용 가능한 소프트웨어를 구축하는 것에 대한 기준을 제시한 규칙이 있다. Rule of Three라는 이 규칙은 최소 3번 이상 재사용이 보장되어야만 재사용 가능한 소프트웨어를 구축하라고 한다. 이와 비슷하게 소프트웨어 프로덕트 라인 엔지니어링(Software Product Line Engineering)에서도 경험적으로 보았을 때, 최소 3번은 재사용되어야 손익분기점(Break even-point)을 넘는다고 얘기하고 있다.

재사용성을 보장하려면 원하는 기능을 구현하는 데 있어서 더 많은 금전적 비용과 시간이 소요된다. 여러 애플리케이션 팀들은 비슷한 기능을 필요로 하더라도 각 팀에서 만들어야 될 애플리케이션에서 구현하기 쉽고 사용하기가 편하길 원한다. 해당 기능에 대한 각 팀이 관심을 가지는 부분은 조금씩 다를 수밖에 없다. 이러한 관심이나 요구사항은 미묘하게 충돌하게 마련이다. 재사용성을 높이려면 이런 부분을 적절하게 만족시키면서 원하는 기능이 충실히 구현되어야 한다.

재사용성이 중요한 프레임워크도 마찬가지다. 프레임워크를 구축하는 일은 애플리케이션보다 더 많은 시간이 필요하다. 앞에서 얘기한 사례들처럼 프레임워크도 추상화 수준에 따라서 대략 2~3배 정도 필요하다. 하지만 애플리케이션과 프레임워크가 동시에 구축되어야 하는 상황이라면 얘기가 달라진다. 애플리케이션은 프레임워크가 완벽히 완성될 때까지 기다릴 수 없다. 병렬적으로 진행되어야 하기 때문이다. 애플리케이션 개발팀 입장에선 아무리 늦어도 애플리케이션이 설계를 시작할 땐 이미 프레임워크의 명세서가 준비되어 있어야 하고, 애플리케이션이 구현될 때는 프레임워크가 준비되어 있어야 한다. 정리하자면, 프레임워크가 시작하기 전에 프레임워크를 구축하는 데 필요한 비용과 난관들을 감수할 만큼 충분히 이득이 될 만한지를 판단해야 된다는 얘기다. Framelets for Multiple Use 패턴이 제시하는 해결책은 다음과 같다.

● 프레임워크가 심플한 상태를 잘 유지할 수 있고 최소 세 번 이상 사용될 수 있다면 프레임워크를 구축해라.

프레임워크는 많은 일을 하는 것보다 정말 필요한 핵심적인 부분들에 집중해서 심플한 상태를 유지해야 한다. 프레임워크가 할 일이 많아질수록 프레임워크 사용자들의 요구사항들이 충돌할 가능성이 높아진다는 것을 잊지 말자.

최소 세 번 이상 사용되어야 한다는 조건은 중요하다. 프레임워크를 쓸 일이 없으면 성공적으로 구축된다 하더라도, 애플리케이션들을 따로 만드는 것보다 비용적으로 손해를 보게 된다. 거

기다 프로젝트가 시작되고 난 뒤에 프레임워크를 사용하게 될 애플리케이션의 숫자가 줄어들 수도 있다는 잠재적인 위험도 존재한다. 그러므로 프레임워크를 네 번이나 다섯 번은 사용되어야 마음 놓고 프로젝트를 시작할 수 있을 것이다.

프레임렛은 애플리케이션 입장에서 본다면, 전체 시스템의 일부분에 대한 작은 프레임워크이다. 프레임렛은 애플리케이션의 다른 계층이나 객체와의 의존성에 대해 고려해야 된다. 그러므로 'don't call us, we call you'(헐리우드 원칙)을 항상 염두에 두도록 하자.

Budget Factor 2.5

● 프로젝트 초기에 프레임워크를 구축하는 데 필요한 예산이 얼마인가?

프레임워크는 애플리케이션보다 비싼 존재다. 프레임워크가 더 많은 예산을 필요로 하는 데는 그만한 이유가 있다. 쉽게 생각해볼 수 있는 이유는 여러 애플리케이션에 맞는 추상화 수준을 찾고 일반화된 기능들을 제공하는 게 쉽지 않다는 것이다. 거기다 프레임워크와 애플리케이션을 동시에 구축하게 된다면, 프레임워크 개발팀은 개발만 해서는 안 된다. 프로젝트가 진행되는 동안 프레임워크 개발팀은 프레임워크 개발, 유지보수 그리고 개발한 프레임워크에 대한 교육까지 책임져야 한다. 충분한 기술과 경험이 없는 개발자가 아니고서야 동시에 이 일들을 처리하는 건 쉽지 않다.

시간을 생각해 보자. 동시에 구축을 한다는 건 애플리케이션 개발팀과 프레임워크 개발팀에게 주어진 시간이 동일하다는 얘기다. 프레임워크 개발팀은 할 일이 많다. 개발도 해야 되고 유지보수도 해야 되고 애플리케이션 개발자에게 교육도 해야 된다. 시간은 한정적이고 해야 할 일은 많다. 자칫 프레임워크가 늦어지는 일이 생긴다면 프레임워크뿐만 아니라 애플리케이션까지 여파가 미칠 것이다. 일손이 부족하고 데드라인은 다가오는 급박한 상황이 발생한다 하더라도 개발자를 더 추가할 수 없다(맨-먼스 미신은 너무 유명하다). 그럼 예산이 얼마쯤이면 이런 문제들을 충분히 극복할 수 있을까?

● 애플리케이션 한 개를 만드는 데 필요한 예산의 2.5배로 계산해라. 그리고 프레임워크를 개발하기에 충분한 실력을 갖춘 팀을 찾아라. 그리고 그 팀이 프로젝트를 시작하는 시점부터 충분한 인원으로 구성되도록 한다.

2.5배라는 수치에 대한 근거는 앞에서 얘기한 것처럼 적당한 추상화 수준을 찾는 게 쉽지 않은 점과 프레임워크에 대한 교육도 책임져야 한다는 것이다. 그리고 팀원 수는 많으면 좋다. 하나의 애플리케이션만 만들 때보다 대략 2배 정도는 필요하다. 개발 과정에 대한 패턴언어도 있다. 짐 코플리엔(Jim Coplien)이 제안한 Size The Organisation 패턴이나 Size The Schedule 패턴을 참고하자.

Two Pilot Applications

● 프레임워크에 대한 요구사항을 어떻게 찾아낼 수 있는가?

이미 구현된 애플리케이션들에서 공통점을 가진 기능들이 있다면, 애플리케이션간의 차이점들을 걷어내고 공통된 추상화를 찾아내는 일을 구현된 애플리케이션이 없는 상황보다 비교적 쉽게 할 수 있다.

그럼 구현된 애플리케이션이 없는 상황에선 어떻게 할까? 프레임워크 개발팀에겐 필요한 만큼 몇몇 애플리케이션을 구현해 볼 시간도 없다. 그리고 프레임워크를 사용하게 될 애플리케이션이 원하는 기능들을 분석해야 되고, 적절한 추상화 수준도 찾아야 한다. 애플리케이션 개발팀들은 각자 자기 요구사항들이 프레임워크 개발팀이 수용해 주길 원한다. 그 요구사항 중에선 서로 충돌하는 내용이 있기도 한다.

프레임워크 개발팀은 모든 걸 다 수용할 수 없다. 프레임워크는 심플해야 된다. 모든 걸 다 수용하게 된다면 프레임워크는 복잡해지기만 할 뿐이다. 이는 결코 프로젝트에 좋은 영향을 주지 못한다. Two Pilot Applications의 해결책을 살펴보자.

● 프레임워크를 사용하게 될 두 파일럿 애플리케이션을 찾아라.

- 파일럿 애플리케이션은 매우 보편적인 요구사항을 가진 애플리케이션이어야 한다.
- 파일럿 애플리케이션은 핵심적인 프레임워크 유저들과 밀접한 관계를 유지하는 데 중요한 역할을 한다.
- 파일럿 애플리케이션은 좀 더 일찍 개발을 시작해야 한다.

파일럿 애플리케이션은 프로젝트를 진행하면서 전반적으로 긍정적인 역할을 해줄 수 있다. 파일럿 애플리케이션을 통해 프레임워크를 어떻게 적용해야 하는지에 대한 경험을 쌓고, 그 경험을 다른 팀에게 알려줄 수 있다. 또한 프레임워크를 적용하면서

프레임워크 개발비용에 대한 간단한 수식

- 프레임워크 개발 = 2.5 × 애플리케이션 개발
- 3 × 애플리케이션 개발 = 프레임워크 개발 + 3 × 프레임워크를 이용해서 애플리케이션 개발
- 프레임워크를 이용해서 애플리케이션 개발 = 1/6 × 애플리케이션 개발

발생하는 문제점이나 요구사항과 같은 피드백을 프레임워크 개발팀에게 알려줄 수 있다. 프레임워크 개발팀은 파일럿 애플리케이션을 개발하는 애플리케이션 팀과 같은 장소에서 함께 개발하면서 적극적으로 피드백을 얻는 것이 좋다.

파일럿 애플리케이션이 주는 이점을 생각하면 '파일럿 애플리케이션이 많으면 많을수록 좋은 게 아닐까?' 라고 생각할 수 있다. 하지만 파일럿 애플리케이션이 2개만 있어도 원하는 역할을 수행하는 데 부족함이 없다. 오히려 많은 파일럿 애플리케이션은 관리하기만 힘들어질 뿐이다.

Small Functions

● 프레임워크의 기능(functionality)들을 인터페이스(interface functions)로 어떻게 나눌 수 있을까?

애플리케이션의 일부분을 프레임워크로 만든 것이 프레임렛이다. 프레임렛이 헐리우드 원칙에 따라서 구현된다곤 하지만, 애플리케이션을 구성하는 다른 부분들과 같이 통합되는 경우를 고려해야 할 필요가 있다. 프레임렛(혹은 프레임워크)은 애플리케이션이 완성되기 위해 필요한 기능들에 대한 인터페이스를 제공해야 한다. 여기에 문제가 있다.

- 적은 수의 기능들로 구성된 프레임워크가 좀 더 이해하기 쉽다.
- 기능이 단순할수록 프레임워크는 좀 더 이해하기 쉽다.

적은 수의 단순한 기능으로 이루어진 프레임워크가 이해하는 데는 제일 좋겠지만, 현실적으로 프레임워크가 제공해야 할 기능들은 생각처럼 적지도 않고 단순하지도 않다. 결국 프레임워크 개발자는 선택의 기로에서 고민해야 한다. 개수는 적지만 크고 강력한 기능들로 이루어진 프레임워크와 기능의 개수는 많지만 작고 단순한 기능으로 이루어진 프레임워크라는 선택의 기로에서 말이다. 트레이드-오프(trade-off) 관계인 이 고민 속에서 애플리케이션은 프레임워크의 기능을 자신이 원하는 대로 조금씩 다르게 사용한다는 점을 염두에 두자. 그럼 이 문제를 어떻게 해결하고 있는지 살펴보자.

● 적은 수의 강력한 기능보다 많은 수의 단순한 기능을 선호하라.

우선적으로 프레임워크의 기능을 사용하는 사용자들이 해당 기능을 이해하는 데 어려움이 없어야 한다. 그러기 위해선 프레임워크의 기능이 단순한 것이 이해하는 데 도움된다는 점을 기억해라. 또한 단순하고 작은 기능일수록 특정 조건에 상관없이 프레임워크 사용자들이 원하는 기능에 부합할 확률이 높다. 물론 생각보다 단순한 기능이라 실망하는 사용자도 있을 것이다. 그런 경우에는 마치 뷔페식처럼 프레임워크에서 제공하는 기능 중 필요한 몇몇의 기능들을 골라서 적절히 결합(Combination)하여 사용자가 원하는 기능으로 만들어 낼 수도 있다. 물론, 경우에 따라서 적은 수로 이루어진 강력한 기능들이 필요할 때도 있다. 이 경우엔 기능의 개수가 적기 때문에 때때로 프레임워크에 대한 학습시간을 줄여줄 수도 있다. 하지만 원하는 기능을 선택할 수 있는 폭도 좁아지기 때문에 프레임워크가 사용될 가능성도 낮아질 것이다.

User Involvement

● 애플리케이션을 구축하면서 프레임워크를 사용할 수 있는 다른 팀들의 멤버들에게 어떻게 해야 프레임워크에 대한 확신을 심어줄 수 있는가?

애플리케이션을 구축해야 되는 다른 팀들은 원하는 기능을 제공하는 프레임워크가 있다면 그 프레임워크를 재사용하고 싶어한다. 프레임워크를 이용하면 원하는 목표를 달성하기 훨씬 쉬워질 것이라 생각하기 때문이다. 프레임워크를 통해 원하는 목적을 달성하기 위해서는 프레임워크에 대해 충분히 이해하고 있어야 한다. 하지만 프레임워크 사용자들은 경험 부족이나 프레임워크에 대한 부족한 이해 탓에 개발 과정 중에 문제가 발생하더라도 본인의 부족함보다 프레임워크가 문제라고 생각하곤 한다.

프레임워크는 종종 유연성(Flexibility)과 효율성(Efficiency) 간의 관계도 고려해야 한다. 효율성이 중요한 상황이라면 애플리케이션 개발팀은 프레임워크가 적용된 애플리케이션을 성능이 잘 나오도록 만들어야 한다. 프레임워크 개발팀은 이러한 부분에 대해 애플리케이션 개발자가 원하는 대로 프레임워크를 튜닝하거나 확장할 수 있도록 해야 한다.

● 프레임워크를 사용하는 팀들을 참여시켜라.

프레임워크에 대한 이해가 부족한 사용자들을 이해시키는 일은 프레임워크 개발팀이 해야 할 역할 중 하나이다. 프레임워크가 애플리케이션에 실제 적용한 경우를 사용자들에게 보여주고 애플리케이션에 적용하기 위해 어떤 준비를 했는지 알려줄 필요가 있다. 프레임워크 개발팀이 할 수 있는 일에 대한 예를 몇 가지 들면 다음과 같다.

- 워크샵을 통해 프레임워크를 애플리케이션에 적용하는 법을 알려준다.
- 애플리케이션 생성을 쉽게 할 수 있는 툴을 제공하고, 툴에 대한 설명을 해준다.
- 프레임워크가 적용된 애플리케이션에 대한 테스트를 제공한다.
- 프레임워크가 적용된 애플리케이션을 최적화하는 방법을 보여준다.

- 프레임워크에 대한 튜토리얼을 만들고 문서를 제공한다.

이런 방법엔 단점도 있다. 바로 비용이다. 개발만 하는 것보다 훨씬 많은 시간과 금전적 비용이 필요하다. 물론 Budget Factor 2.5를 잘 적용하였다면 큰 문제가 되진 않을 것이다.

Tests Based on Pilot Applications

● 어떻게 하면 프레임워크를 충분하고 신뢰성 있게 테스트할 수 있는가?

프레임워크에서 버그가 발생하게 된다면, 프레임워크를 사용하는 모든 애플리케이션에 영향을 줄 수 있다. 하지만 프레임워크 테스트는 쉬운 일이 아니다. 프레임워크는 여러 애플리케이션이 사용해야 하기 때문에, 프레임워크로 할 수 있는 일이 애플리케이션에 비해 그 범위가 훨씬 넓다. 어디서부터 어디까지 테스트해야 될지 막막하다. 더군다나 프레임렛의 경우엔 전체 시스템 구조가 아니다. 애플리케이션의 일부분일 뿐이다. 그럼 어떻게 해야 충분한 테스트를 할 수 있을까.

● 파일럿 애플리케이션들을 바탕으로 회귀 테스트를 해라.

- 파일럿 애플리케이션에서 핵심적인 컴포넌트를 찾아낸다.
- 파일럿 애플리케이션에서 가장 보편적인 시나리오를 찾아낸다.
- 파일럿 애플리케이션에서 적용했던 테스트를 프레임워크가 변경된 경우에도 테스트로 활용한다.

이 방법만으론 몇몇 특정 시나리오는 테스트하지 못할 수도 있다. 특정 상황이나 특정 애플리케이션에서만 생겨날 수 있는 시나리오가 존재할 수도 있기 때문이다. 이런 경우엔 임의로 해당 시나리오를 테스트에 포함시킨다. 그리고 필요하다면 성능이나 프레임워크의 안정성과 같은 부분에 대한 테스트 케이스도 넣어준다.

Double Change Request

● 프레임워크에 새로운 기능(functionality)들을 추가해 달라는 요청들이 많으면 어떻게 해야 하는가?

프레임워크가 필요한 기능을 제공해주면 애플리케이션 개발자들은 한결 수월하게 개발할 수 있다. 애플리케이션 개발팀 입장에선 필요한 기능이 다른 팀에서도 필요한 기능인지는 관심사항이 아니다. 필요하고 중요하다고 생각되면 프레임워크 개발팀에 요청하는 것이다. 그리고 애플리케이션 개발팀의 요청들은 서로 충돌하는 요구사항들인 경우가 많다. 애플리케이션마다 프레임워크의 기능들을 조금씩 다르게 사용하는 건 당연한 일이다.

프레임워크 개발팀은 애플리케이션 개발팀이 필요로 하는 기능을 제공해야 하지만, 프레임워크를 심플하게 유지해야 된다. 그럼 언제 그 요청을 수락하고 프레임워크에 새로운 기능을 추가해야 하는지 기준이 필요하다.

● 적어도 두 팀 이상이 추가될 기능에 대해 필요로 하는 경우에 추가적인 요청을 수락하라.

프레임워크 개발팀에게 새로운 기능을 추가해 달라는 요청이 들어왔다면, 다른 팀들에게도 이 기능이 필요한지 물어보자. 잠재적으로 재사용될 가능성이 높지만, 다른 팀에선 아직 요청하지 않았을 수도 있다. 충분한 수요가 있는지 알아보자. 애플리케이션에 특화된 기능을 추가해주길 원한다면 당연히 거절해야 한다. 하지만 애플리케이션 개발팀이 원하는 기능을 쉽게 추가할 수 있도록 인터페이스를 제공해주자. 프레임워크 개발팀이 구축한 프레임워크는 최소 세 번은 사용된다. 그 중에서 두 팀이 원한다는 얘기는 최소 세 번은 재사용되어야 하는 기준에는 못 미치기 때문에 별도의 프레임워크를 구축할 정도의 가치는 없다는 얘기다. 오히려 최소 세 번은 사용될 프레임워크에서 두 팀이 원하는 기능을 제공하는 것이 낫다.

이해의 시작

지금까지 프레임워크와 애플리케이션을 동시에 구축하는 상황에서 적용할 수 있는 패턴들을 살펴봤다. 대규모 프로젝트에 참여하여 애플리케이션들과 프레임워크가 같이 진행되어야 하는 상황에서 직접 적용해 볼 수 있다면 금상첨화겠지만, 그렇지 못하더라도 프레임워크가 가지는 특성을 이해하고 여러분이 프레임워크를 공부하는 데 있어서 도움이 되었으면 하는 바람이다.

참고자료

1. Adreas Ruping, Building Frameworks and Applications Simultaneously, PLoP 2000.
2. Adreas Ruping, Patterns for Successful Framework Development, PLoP.
3. Krzysztof Cwalina, Brad Abrams, Framework Design Guidelines 2nd Edition, 2008.