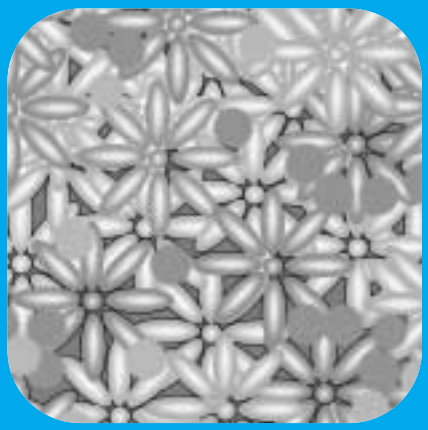

cover story Plus

패턴 적용에 대한 난상토론

"패턴 적용의 득과 실은 이것!"

패턴을 실제 프로젝트에 적용하는 과정에서 겪는 고충에는 어떤 것들이 있을까? 그리고 그 해결책은 무엇일까? 그에 관한 힌트를 얻고자 몇몇 개발자들이 모여 난상 토론을 진행했다. 마소 필자들이 다수 속한 개발자 스터디그룹 EvaCast의 회원들로부터 패턴에 대한 더 생생한 이야기를 들어보자.

EvaCast www.EvaCast.net | 패턴과 소프트웨어공학 기술들을 연구하는 스터디그룹이다. 이 땅에 패턴 전도사를 꿈꾸며 각종 세미나 및 기고 활동을 하고 있다. 최근에는 몇 년간 쌓은 스터디 결과물을 바탕으로 무료 동영상 서비스인 EvaCast (http://www.EvaCast.net)를 오픈했다. 여러 분야에 종사하는 팀원들의 다양성, 끊임없는 학구열, 그리고 데브피아의 지원으로 이미 6년여에 걸쳐 운영되어온 장수 스터디그룹이다. 최근에는 패턴계의 난공불락인 「POSA2」를 번역하고 있다.

영수 _ 먼저 패턴을 적용하면서 겪었던 여러 가지 에피소드를 나누고자 한다. 프로젝트에 패턴을 실제 적용했던 경험을 말해 달라.

팀원 간 패턴 이해의 차이

아무개 _ 코딩할 때 패턴을 적용해 봤다. 결국 주위에 있는 사람이 패턴을 모르면 결국 적용하기가 매우 어렵다. 결국 적용하는 데 어려움을 가져서, 사수(상급자)가 가진 자신만의 경험을 그대로 사용했다. 물론 내가 공부한 패턴을 적용하고 싶었지만, 사수가 경험적으로 사용하는 방식으로 프로그래밍하면 사수한테 많은 도움을 받을 수 있다. 반면에 내가 부재시 그 모듈에 문제가 발생했을 경우에 사수가 대신 대처할 수 없다는 것이 단점이다.

상정 _ 그럼 디자인 패턴과 사수가 정의한 패턴 중에서 어느 것이 나았는가?

아무개 _ 물론 디자인 패턴을 쓰는 게 나았다고 생각한다. 하지만 주위의 시선들, 즉 '시간상의 문제, 잘 돌아가잖아. 그러니 굳이 패턴까지 할 필요가 있을까?' 라는 생각 때문에 적용하지 못했다. 결국 패턴은 모든 팀원들이 이해해야만 사용할 수 있다는 교훈을 얻었다.

선욱 _ 이미 패턴화되어 있는 시스템을 사용해본 적이 있다. 물론 이것이 패턴이라는 것을 모른 채, 사용하는 사람도 많았다. 패턴화되어 있는 시스템의 전체적인 구조를 알고 쓰는 것과 모르고 무조건 이 함수에 써야 한다고 생각하는 것은 시선의 차이가

있다. 또한 여러 가지 프레임워크에서 자동 생성화된 코드를 충분히 이해하지 못해서, 잘못 이용한 사례도 많았다. 결국 국내 IT의 현실상 패턴을 잘 몰라도 프로그램을 짤 수 있으니 많이 사용되지 않는 것 같다.

상정 _ 하지만 고급 프로그래머로 가면 갈수록, 프레임워크를 구축하거나 이해하고 사용해야 되는 경우가 많으므로, 패턴은 좋은 레퍼런스가 되므로 이해해야 된다. 고급 프로그래머로 가기 위해서는 꼭 공부해야 되는 대상이 아닌가?

선욱 _ 맞는 말이다. 그러나 패턴과 같은 소프트웨어 설계 기술보다는 도메인 기반 지식을 중요시 여기는 곳이 많다. 금융 같은 경우는 금융법/상법과 같은 것들을 중요시 여기므로, 패턴이나 프로그래밍 기술 적용들은 상대적으로 덜 중요시되는 게 사실이다.

선진 _ 난 변화에 유연하게 만들기 위해 패턴을 사용했는데, 주위 분들이 패턴에 대해 이해를 못해서 오히려 힘든 적이 있었다.

용현 _ GoF 디자인 패턴을 배우고 있었는데, 코딩은 따로 하고 있었다. 패턴을 사용한다는 것은 무의미했다. 주변사람들이 패턴(상속)에 대해 이해를 못했고, 오히려 이것 때문에 코드가 스파게티가 된 적도 있다. 지금 같은 경우는 클래스 네이밍을 할 때, 패턴을 사용하기 시작했다. 그 이후 생활화가 되어 커뮤니케이션이 원활해지고, 코딩 기술이 향상된 것 같다.

영수 _ 패턴을 다른 사람들에게 알려줘 그들이 적용하기 시작했는가?

토론을 나눈 EvaCast 회원들

> “결국 패턴은 모든 팀원들이 이해해야만 사용할 수 있다는 교훈을 얻었다.”

용현 _ 코드 리뷰를 통해 주위 팀 동료에게 자연스레 전파되었다. Template Method 패턴을 적용했는데, 그게 좋다는 것을 알고 다른 이의 코드 리뷰에 Template Method가 퍼진 것을 보게 되었다. 그렇지만 실제 주위의 개발자들 가운데 그것이 패턴인 줄 모르고 사용하는 이들도 있다.

영수 _ 코드 리뷰를 통해 패턴을 전파하다니 상당히 좋은 도구인 것 같다. 하지만 팀 내에서 적용하기가 힘들지 않았는지 궁금하다.

용현 _ 물론 듣는 사람은 많은 것을 배울 수 있지만, 리뷰를 하는 당사자는 리뷰 준비를 위해 많은 시간을 뺏긴다. 그러나 외부로 보여줘야 하기 때문에 좀 더 신경 써서 코드를 보는 것은 사실이다.

국내 IT 환경이 가져오는 문제들

영수 _ 그럼 팀원 간의 이해도 외에 프로젝트에 패턴을 적용하기 어려운 이유는 또 무엇이 있을까?

용현 _ 프로젝트 일정이 너무 타이트하게 정해져 있어서, 그 짧은 시간 안에 패턴을 적용하기가 어렵다.

상정 _ 하지만 소프트웨어 생명 주기적인 관점에서, 배포한 후 쉽게 요구사항을 받아들이기 위해서는 패턴을 적용하는 게 오히려 맞다.

용현 _ 인수인계할 때, 패턴으로 적용한 코드 스타일이 다 달랐다. 여러 명의 개발자를 거친 코드인 경우에 어떤 사람은 Template Method 패턴을 적용했지만, 다른 사람은 전혀 다른 패턴을 이용해서 패턴 간에 상충되는 문제가 발생했다.

선욱 _ 요즘 이른바 ‘노(No) 설계’, 아예 설계를 하지 않는 것도 있다. 나중에 변하고 안 변하는 것을 보고 나서, 그 후에 리팩토링하는 것이 더 낫다는 의견도 많다.

선진 _ 설계를 하지 않는 것이 아니라 오전에 설계하고, 구현하고, 테스트하고 이렇게 하는 것이 곧 XP의 성격인데, 그것을 설계를 하지 않는다고 보는 이들도 있다. 열심히 패턴을 적용해 설계하고 리팩토링하는 것이 더 낫다.

용준 _ 하지만 국내 프로젝트 현실상 당장 설계를 할 시간이 없다. 무조건 코딩하고 올리고 피드백을 받기에 급급하다. 당장 결과를 내야 되니 그럴 수밖에 없다.

선욱 _ 국내에서는 GoF 디자인 패턴을 넘어선 사례가 별로 없는 것 같다.

영수 _ 프레임워크와 같은 기초 과학이 아니라, 마치 응용과학에 비유되는 SI에만 관심이 있어서 설계나 패턴에 대한 연구가 부족하다.

용현 _ 프레임워크에 익숙한 개발자는 프레임워크에 없는 기능이면 구현 못하는 개발자가 많다. 물론 나중에 역량이 쌓이면 자신만의 프레임워크를 만들 수도 있겠지만, 결국 그러기 위해서는 패턴을 공부해야 한다.

선욱 _ 국내에서는 기반 기술에 대한 연구가 힘든 이유가 있다. 예를 하나 들면, 자바(Java)에 3D 그래픽 라이브러리가 없을 때, 3D 라이브러리를 구축한 몇몇 벤처 회사들이 있었지만, 썬에서 다음 자바 라이브러리에 표준 3D 라이브러리를 추가하자 관련 업체가 망했다. 국내 IT 업체들이 기반 프레임워크를 구축하고 판매하면서 살아남기가 어려운 게 사실이다.

영수 _ 동감한다. 지금 국내에서 유명한 RIA 라이브러리도 다 외산 컴포넌트를 기반으로 국내 현실에 맞게 바꾼 것이 많다.

패턴 적용에 대한 가이드 라인 (도메인 기반의 지식과 리팩토링)

영수 _ 패턴을 잘 적용하기 위한 가이드라인에는 무엇이 있을까?

아무개 _ 잘못 적용한 경우를 하나 예로 들어보겠다. 유명 컨설턴트가 엔터프라이즈(Enterprise) 시스템에서 사용하는 패턴(Pipe & Filter, Composite Message)을 임베디드 시스템에 사용했다. 시스템에서 하나의 통신을 하는 데 10초 이상의 시간이 걸리자 개발자들이 반응속도에 경악하게 되었고, 결국 초기 설계한 패턴 형태의 아키텍처를 포기하고 소켓(Socket) 기반으로 간적이 있다. 또 원활한 메시지 디스패칭을 위해 디스패칭할 핸들러 정보를 파일에 기술했다(Component Configurator). 하지만 제한적인 리소스와 싸우는 임베디드 시스템에서는 핸들러 정보를 메모리에 모두 올리는 방식은 적합하지 않았다. 패턴 사용 시에 성능적인 이슈들은 신경 쓰지 않고, 유연성이나 확장성 위주의 패턴을 잘못 적용하는 경우를 보았다.

상정 _ 제한된 임베디드 시스템에서도 이러한 문제를 해결하기 위해 다른 방법 등을 동원한다. 레지스트리 시스템을 구축하지 못했지만, Shared Memory로 비슷한 문제를 해결한 적이 있다. 이러한 것들도 역시 하나의 패턴이라고 생각한다.

아무개 _ 또한 변화를 예측하지 못한 경우에도 발생한다. 절대 기능이 추가될 리 없다고 생각했던 프로젝트가 있었다. 어떤 모듈 부분에 '기능 추가 없음, UI 추가 없음'을 가정했던 프로젝트였는데 기획팀에서 이것은 기능 추가가 아니라 버그 수정이라고 말한 적이 있다. '다른 회사에는 다 있는데 왜 우리는 없는가? 어서 추가해라.' 이런 식이었다.

영수 _ 그건 요구사항 변경으로 봐야 하는 거 아닌가?

아무개 _ 기획 쪽에서는 프로그래머와 보는 관점이 다르다. 변하지 않는 부분을 발판으로 작성해야 하는데, 예측하지 못한 부분이 변한다. 그러한 점이 어렵다.

선욱 _ 패턴도 변하지 않는 것을 정하고, 변하는 것을 가려내는 데 결국 도메인에 해박한 지식이 필요하다.

영수 _ 도메인 지식이 없으면 유명한 아키텍트도 힘들었던 상황들을 주위에서 보았다. 초기에 설계한 디자인이 중간에 요구사항이 바뀌어 뒤엎는 경우를 보았기 때문이다. 특히 그 부분이 기반 구조에 있는 특성(Feature)이라면 악 영향이 더 커진다.

> "프레임워크에 익숙한 개발자는 프레임워크에 없는 기능이면 구현 못하는 개발자가 많다. 물론 나중에 역량이 쌓이면 자신만의 프레임워크를 만들 수도 있겠지만, 결국 그러기 위해서는 패턴을 공부해야 한다."

용현 _ 변화를 정확히 예측하지 못하는 경우에는 패턴 적용이 독이 될 수 있다.

영수 _ 결국 패턴도 변화를 예측하지 못하는 상황에서는 답이 없다. 그렇다고 성능, 이해 당사자의 요구를 무시해가며 모든 걸 변화에 유연하게 대처하도록 Over Engineering을 적용할 수는 없지 않을까? 그럼 도메인 지식과 결합된 도메인 기반의 패턴이 있는가?

선욱 _ 그런 건 없지만, Software Factory는 봤다. 예를 들어 A라는 차를 만들었을 경우, 이 요구사항을 추적해 놓으면, 다음에 유사한 차 B를 만들었을 때, 요구사항 부분들이 이미 예전에 구축

되어 있으므로 현재의 요구사항과 비교할 수 있어 좀 더 쉽게 요구사항을 파악할 수 있다. 이러한 요구사항을 분석하는 Analysis 패턴도 있다.

상정 _ Test Case도 역시 비슷한 사례가 있었다. 예전에 만든 휴대폰의 Test Case가 축적되어 있어서 유사한 휴대폰을 만들 때 예전 Test Case 패턴이 그대로 이용되었다.

선욱 _ 패턴 적용에 있어서는 패턴과 도메인이 함께 고려되어야 한다. 패턴 전문가와 도메인 전문가가 협력해야 한다. 결국 기업 문화가 중요하다.

> "패턴 적용에 있어서는 패턴과 도메인이 함께 고려되어야 한다. 패턴 전문가와 도메인 전문가가 협력해야 한다. 결국 기업 문화가 중요하다."

선진 _ 어디까지 패턴을 따르고 우리 프로젝트에 맞게 다듬어야 할까? 그리고 또 어떠한 것들을 도메인적인 문제로 봐야 하는지 기준을 정하는 게 매우 어려운 문제다. 또한 리팩토링도 적극적으로 고려해야 한다. 리팩토링에서 인라인 메소드나 익스트랙트 메소드가 상반된다. 이러한 적용을 어떻게 판단해야 하는가? 좋은 관점을 가진 사람이 중간에서 조절해야 하지 않을까? 미래를 예견할 수 있는 능력이 필요하다.

패턴 학습에 대한 이야기

선욱 _ 모든 학습이 연관되어 있는 것은 연달아 보는 것이 좋다. GoF 패턴을 보고 처음에는 잘 이해하지 못하더라도, 다시 반복적으로 학습하고 다른 사람의 의견을 받아들일 수 있는 자세가 중요하다.

지원 _ 현재 플랫폼을 구축하면서 많이 배우는데, 특히 운영체제 하부와 연관된 작업을 하면서 운영체제는 패턴 덩어리라고 느꼈다. 운영체제 코드간의 의존성들을 파악하고, 특정한 패턴들을 찾아보면서 많이 배울 수 있었다.

선욱 _ 패턴의 안경을 쓰고 시스템을 바라보면 정말 좋은 학습이 될 것이다.

용현 _ 일정이 없는 프로젝트에서는 새로운 기술과 패턴을 적용하기가 어렵다. 하지만 오픈소스와 같은 경우는 일정에 크게 영향을 미치지 않는다면 패턴들을 많이 적용했다. 여러 가지 패턴의 활용 사례들을 공부하는 데 도움이 된다.

상정 _ 패턴을 적용하다가 너무 세부적인 설계까지 내려가면, 패턴보다 작은 스케일의 관용구(Idiom)를 보게 된다. 이 부분에 대한 학습도 필요하다.

영수 _ 또한 패턴을 다양한 전략의 관점으로 보아야 한다. 하나의 문제를 해결하기 위해 하나의 패턴만 있는 것으로 오해하는 경우가 있다. 하지만 실제 하나의 문제를 해결하기 위해 여러 가지 패턴이 존재하므로 상황에 맞게 적절히 선택해야 한다. 예를 들어 분산객체에서 마샬링할 때를 떠올려보자. Composite Message Pattern 같은 경우 Loosely하게 설계하면, 중간에 주고받는 데이터를 하나의 메타데이터(metadata), 예를 들면 XML을 이용하기도 한다. 운영체제에 독립적이라는 장점을 얻을 수 있지만, 성능이 낮아지는 게 단점이다. 이와 반대 사례로 Simple Object라는 방법이 있다. 클라이언트에서 서버 쪽 핸들러의 메모리 주소를 얻어옴으로써 아주 빠르게 실행하는 방법이다. 하지만 운영체제나 플랫폼에 종속적이라는 단점이 있다. 그러므로 상황에 맞게 적합한 패턴을 선택하는 게 핵심이다.

아무개 _ 너무나 패턴적인 방법으로만 접근하는 것 역시 문제가 있다. 대부분의 성능적인 이슈는 알고리즘으로 풀어나가는 게 맞다.

영수 _ 우리 스터디가 6년이 다 되어 간다. 패턴을 개인적으로 공부했을 때와 비교해 스터디에 참여했을 때의 장점이 무엇인지 말해 달라.

선욱 _ 혼자 아는 것과 아는 걸 표현하는 것은 다르다. 10분을 발표하기 위해서는 1시간 이상의 준비가 필요하다. 패턴을 발표하는 스터디를 꾸준히 하다 보니, 자신이 발표하는 부분은 확실히 이해하게 되고, 그것을 기반으로 다른 패턴들도 이해할 수 있게 되었다. 서로 간에 지식을 쉽게 습득할 수 있는 것이 좋다.

지원 _ 스터디를 통해 이론적으로 보고 회사에서 실제 적용하면서 쓰기 때문에 선순환의 과정을 창출할 수 있어서 도움이 되었다.

현구 _ 회원들과 함께 발표하고 피드백을 받아서 다양한 의견을 수렴할 수 있었다. 다른 시각의 관점과 적용 사례 등을 공유함으로써 패턴을 더 폭 넓게 바라볼 수 있었다.

맺음

토론을 마치고 토론 장소를 나서려고 하는 순간에 모든 배움에 도움이 될 만한 명언을 하나 발견했다. 그 글을 적으며 토론 내용 정리를 마무리한다.

'작은 일도 목표를 세워라. 그러면 반드시 성공할 것이다.'

- 슐러