공학 트렌드

SW 프로세스 개선 통찰 :
애자일로부터 얻은 것, 놓은 것 그리고 간직하는 것 Part 1
- 애자일의 핵심가치와 키 프랙티스

2012. 10. 29. (제30호)

목 차

소프트웨어공학센터
경영지원TF팀

Ⅰ. 소프트웨어 개발 프로세스의 차이점과 공통점

세상에는 다양한 소프트웨어 개발 프로세스가 존재하고 차이점도 분명하다. 그러나 소프트웨어 개발 프로세스가 효율적이며 품질 좋은 소프트웨어의 개발이라는 공통적 목표를 갖고 있기 때문에 차이점뿐만 아니라 공통점도 있게 마련이다.

UP(Unified Process) 객체지향 프로세스와 UML(Unified Modeling Language) 모델링 표준 언어를 공동으로 창시한 이바 야콥슨(Ivar Jacobson)박사는 개발 프로세스들이 공통성을 부정함으로써 개별 프로세스만의 특징들을 부각시키지만 실제로 개발 프로세스들은 많은 공통성에 기반하고 있다고 지적하였다(Jacobson, 2007).

어느 누구도 소프트웨어 개발의 모든 것을 다 알 수는 없고 한정된 부분에서의 전문가이기 때문에 소프트웨어 개발 프로세스에는 경험을 통하여 유용하다고 인정되는 공통된 프랙티스가 존재한다는 것이다. 애자일과 같은 가벼운 프로세스(light process)에 반하는 무거운 프로세스(heavy process)로 잘 알려져 있는 RUP (Rational Unified Process)을 살펴보면 둘 다 반복적이고 점진적인(Iterative and Incremental process) 개발을 핵심사항으로 담고 있다. 이처럼 소프트웨어 개발 프로세스들을 대립적인 관점에서 이해하기보다는 개발 프로세스들이 담고 있는 차이점과 공통점을 이해하고 자신의 상황에 적합한 방법을 가려내고 포용하려는 실용적인 관점이 유용하고 적절하다.

프로세스, 문서, 계획 중심의 워터폴 지향적이던 무거운 프로세스에서 사람, 작동하는 소프트웨어, 반복적 개발을 지향하는 가벼운 프로세스로 진화하는 배경에는 기술적인 우위보다는 심플한 실용성의 승리라는 생각이 든다. 마치 스펙 측면에서 보다 완벽한 SOAP(Simple Object Access Protocol, 단순 객체 접근 프로토콜)보다 간편하고 심플한 REST(Representational State Transfer)방식의 프로토콜을 개발자가 선호하는 것처럼 말이다.

필자는 외산 개발방법론에만 의존하던 1995년부터 국내 현실에 알맞는 개발방법론을 만들기 위했던 마르미(MaRMI) 개발 프로젝트를 시작으로 RUP, 객체지향, 컴포넌트 기반 개발, CMM(CMMi)와 같은 무거운 개발 프로세스를 거쳐 XP, Scrum과 같은 애자일한 가벼운 프로세스를 글로벌 개발환경에 적용하고 코칭을 수행하였다. 이러한 개발 프로세스의 적용 경험을 바탕으로 애자일을 수행하며 얻은 것과 무거운 프로세스 개선 방법 중에서 놓은 점들 그리고 모든 개발 프로세스에서 유용했던 내용을 같이 나누고자 한다.

Ⅱ. 애자일로부터 얻은 것

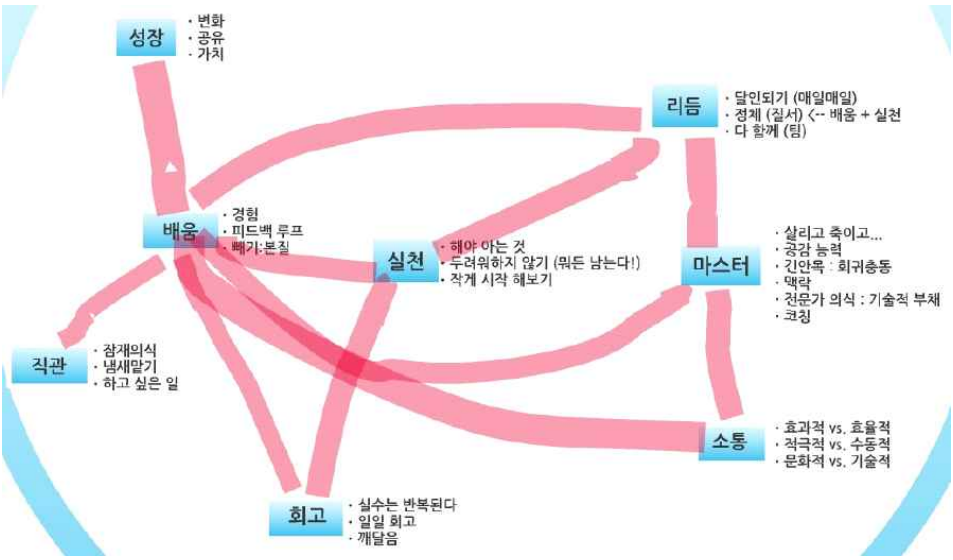
소프트웨어 개발 생산성과 품질을 향상시키기 위한 노력으로 1970년대부터 소프트웨어 공학이 발전하며 건축과 같이 자원과 비용, 일정을 미리 산정한 후 설계를 작성하고 이를 구현하는 공학적 체계를 갖추게 되었다. 이러한 소프트웨어 공학의 발전에도 불구하고 대다수의 소프트웨어 개발 프로젝트들은 납기 지연 및 예산 초과와 같은 현실적 어려움에서 크게 벗어나지 못하였다. 2000년대 들어서 소프트웨어 개발에서의 낭비를 줄이고 빠르게 개발하는 것이 중요하다고 생각하는 사람들이 모여서 애자일(Agile)이라는 새로운 개발 방법을 제시하였다. '민첩하다'라는 의미를 갖는 개발 프로세스의 공통성은 애자일 개발헌장(The Agile Manifesto)에 기반을 두고 있다.

- 개인과 상호작용이 공정과 도구보다 중요하다
- 작동하는 소프트웨어가 포괄적인 문서화보다 중요하다
- 고객과의 협력이 계약협상보다 중요하다
- 변화에 응대하는 것이 계획을 준수하는 것보다 중요하다.

국내에도 애자일이 소개되기 시작한 뒤로 10여년이 지난 오늘날에는 애자일을 적용한 프로젝트 사례나 커뮤니티들이 속속 생겨났고 많은 사람들에게 전파되었다. XP(Extreme Programming), Scrum과 같은 프로세스를 따르지 않더라도 CI(Continuous Integration), Refactoring, TDD(Test Driven Development)와 같은 개별 프랙티스들도 많이 활용되고 있다.

애자일의 성공사례를 제시하며 최고의 개발방법이라고 믿는 사람들도 있고 너무 이상주의적이라는 비판과 함께 무시해버리는 경우도 있다. 국내 소프트웨어 개발 프로젝트들의 현실을 감안한다면, 진실은 그 어느 중간쯤에 있을 것이라고 짐작된다. 애자일이라는 뜻이 변화를 포용하는 기민함에서 출발한 것은 기존의 공학적 방법으로는 모든 소프트웨어 개발 과정을 해결할 수 없었기 때문이다. 스크럼을 전사적 표준 프로세스로 채택하여 교육과 코칭을 진행하며 XP의 엔지니어링적인 실천방법을 도입해 오면서 느꼈던 점은 애자일은 프로세스나 기법 자체보다도 개발 과정에서 부딪힐 수밖에 없는 불확실한 사건과 변화에 대응하고 보다 나은 가치를 만들어 가기 위한 태도나 마음가짐의 중요하다는 점이었다. 아래 <그림 1>은 필자가 애자일을 통해서 새롭게 얻었던 내용을 정리한 것이다.

<그림 1> 애자일로부터 얻은 것



소통

일반적으로 프로젝트란 것이 팀을 이루어 공동으로 수행하게 되므로 의사소통은 어떤 개발 과정에서도 간과할 수 없는 중요한 항목이다. 기존의 무거운 개발 프로세스들도 소통을 강조하고 있지만 그저 명목상인 경우도 있고 대체로 효과성보다는 효율성을 강조하는 경향이 높다.

이해관계자가 늘어날수록 팀에서 의사소통 채널은 지수적으로 증가하게 마련이다. 대표적인 프로젝트 관리방법론이라 할 수 있는 PMBOK(Project Managment Body of Knowledge)에서는 프로젝트 관리자 업무의 90%가 의사소통이라고 정의하기도 한다. PMBOK에서는 의사소통을 효율화시킬 수 있는 의사소통 시스템을 체계적으로 갖추고 당사자에게 정확하게 전달되도록 요구한다.

24/7 온라인 접속이 가능한 환경과 의사소통 가이드가 제공되어도 프로젝트에서 의사소통의 문제점과 어려움은 좀처럼 해결되지 않는다. 프로젝트 성공 요인으로 가장 많이 등장하는 것 중에 하나가 바로 원활한 의사소통이다. 원활한 의사소통을 위해서는 말하는 사람의 의도가 빠르게 전달되는 것보다 제대로 전달되는 것이 중요하다. 즉, 효율성보다 효과성이 요구된다는 의미이다. 예들 들어, 물건을 주문했는데 총알배송 시스템으로 주문한지 몇 시간 만에 물건이 배송되더라도 엉뚱한 물건이 전달되면 아무런 의미가 없는 것이다. 소통 과정에서 말 자체보다 사람의 표정이나 분위기 속에 중요한 진실이 담기는 경우도 많다.

애자일에서는 직접 대면(Face to Face)하여 자주 그리고 규칙적으로 투명한 의사소통이 중요하다고 얘기한다. 계층적 구조의 보고 체계가 단단해질수록 의사소통이 수동화 되고 형식에만 치중하기 쉽다. 계층 및 조직 간의 장벽을 허물고 수평적이고 눈치를 보지 않고 대화할 수 있는 소통의 문화가 애자일에서의 소통방식이다.

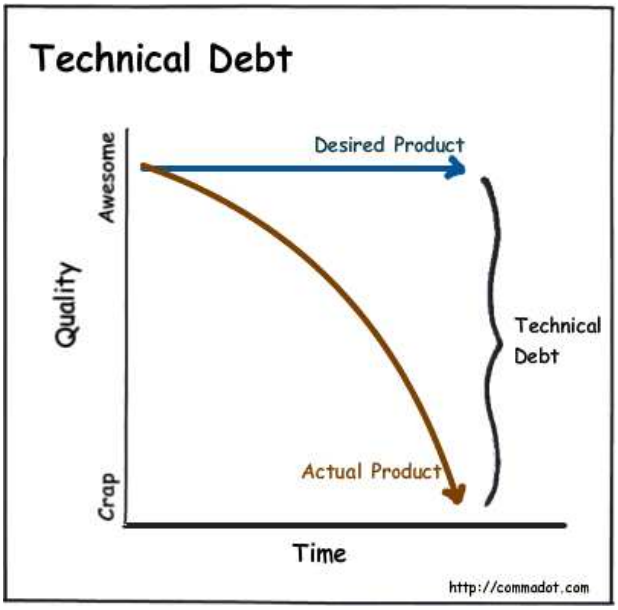
마스터

무거운 개발 프로세스에는 프로젝트 관리자, 테스트 관리자, 품질보증 관리자처럼 다양한 관리자의 역할이 정의된 경우가 많다. 프로젝트에서 전문화된 역할을 정의하고 권한과 책임을 구분하기 위함이다. 그런데 애자일에는 마스터(Master)혹은 코치(Coach)라는 생소하고 포괄적인 역할이 있다. 마스터는 팀의 장애물을 제거하고 애자일의 원칙을 지켜 주며 팀원의 업무수행을 돕기 위해 존재한다. 그래서 애자일 프로젝트의 중요한 성공요인 중 하나가 마스터의 능력이라 할 수 있다.

얼마 전 포춘지(Fortune)에 구글 회장인 에릭 슈미트(Eric Schmidt)가 자신이 들었던 최고의 조언이 바로 코치를 두라는 말이었다고 소개한 글이 실렸는데, 다음과 같다. "코치는 선수만큼 스포츠를 잘 하지 않아도 되는 사람입니다. 코치의 역할은 사물을 다른 시각에서 바라보고 자신의 관점에서 조언해주고 그 문제에 어떻게 접근할지 논의해주는 사람입니다." 팀원들의 잠재능력이 발휘할 수 있도록 만들어주는 것이 코칭인 마스터의 중요한 역할이다.

마스터는 관리자가 아니다. 보고를 듣고 이래라 저래라 결정하는 사람이 아니기 때문에 팀원들과 공감하는 능력이 관리능력보다 중요하다. 말은 쉽지만 일정과 예산 등의 압박을 받는 경우, 익숙한 관리자 모드로 복귀하려는 회귀충동을 자연스럽게 느끼게 된다. 마스터는 긴 안목에서 사람에 대한 신뢰를 갖고 팀원들의 성장을 도와 팀 전체의 능력을 향상시키는 것이 '향상된 결과를 만들어낸다'라는 믿음을 갖고 유혹을 극복할 수 있어야 한다. 프로젝트가 실패하는 경우 한꺼번에 무너지는 것이 아니라, 기술적 부채(Technical debt)가 누적되어 회복이 불가능해질 때 까지 방치된 경우가 대부분이다. 기술적 부채는 당장 눈앞에 놓인 이익을 위해 급한 불만 꺼나가는 잘못된 의사결정이 지속되어 누적된 결과물이다. 마스터는 기술적 부채에 대해서 전문적 지식과 소명의식을 가지고 적극적으로 대응하도록 노력해야 한다.

<그림 2> 기술적 부채



(그림 인용: commadot.com)

리듬

체력향상을 위해 매주 7시간을 운동한다는 목표를 세웠다고 가정해보자. A는 하루 1시간씩 꾸준하게 운동을 하고 B는 하루에 몰아서 7시간을 운동한다고 할 때 누가 더 좋은 성과를 얻을 수 있을까? 당연하게 A일 것이다. 전문가 혹은 달인이 된다는 것은 규칙적인 훈련과 반복을 통하여 얻어지는 것이다.

낯설던 애자일 실천방법들도 프로젝트에서 반복되어 익숙해지면 긴장감과 재미가 사라지고 지루해지기 쉽다. 다른 실천방법을 시도하거나 향상된 목표를 세우고 실천방법에 변화를 주거나 새로운 실천방법을 배워보는 것이 필요하다. 일일, 개발주기, 릴리즈 주기별로 규칙적인 개발과정 속에는 변화와 개선도 포함되어 수행된다. 개발과 개선이 별도로 존재하는 것이 아니다. 팀에서 룰을 정할 때는 모두의 합의가 필요하고 합의된 룰은 준수되어져야 한다. 하나의 작업이 완료되는 "Done"의 의미는 팀 전체에 명확하고 일괄되게 통용되어야 한다. 아울러 내가 맡은 일이 끝났다고 손을 놓는 것이 아니라 기능을 교차하며 팀에 남은 작업을 함께 진행하여 병목으로 인한 지연을 방지하고 학습효과로 팀의 능력도 향상된다.

실천

무거운 개발 프로세스가 P-D-C-A(Planning-Do-Check-Action)사이클을 기반으로 계획 수립을 강조한다면, 가벼운 프로세스에서는 A-L-P(Action-Learning-Planning)접근을 통한 실천을 강조한다. 프로젝트에서 새로운 뭔가를 수행하거나 중간에 변화를 만들어내는 것은 많은 용기를 필요로 한다. XP의 5가지 가치(커뮤니케이션, 단순함, 피드백, 용기, 존중)에 용기가 들어간 이유이다. 막연하게 결과를 두려워하기보다 실패를 성공으로 가는 과정으로 여기는 긍정적 시각이 필요하다. 실패의 영향이 크거나 확신이 부족한 경우, 실험적으로 규모와 범위를 줄여 가볍게 시작해보는 것이 바람직하다.

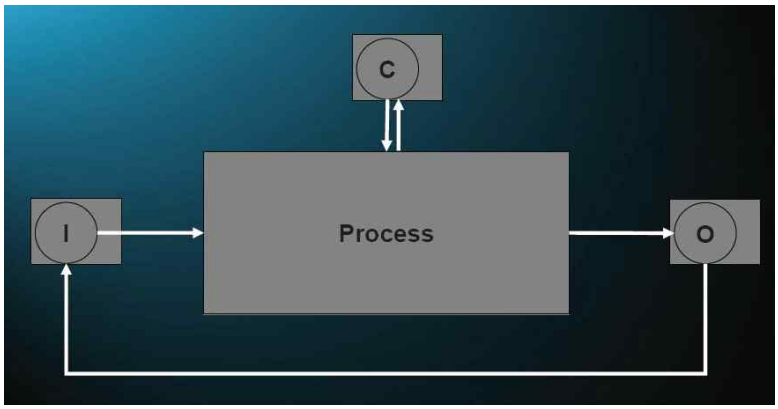
회고

학창시절에 틀렸던 문제를 반복해서 틀렸던 경험을 가지고 있을 것이다. 실수의 원인을 이해하여 의식적으로 수정하려는 노력이 없다면 같은 상황, 같은 조건에서는 예전과 똑같은 행동을 반복하려는 관성을 따른다. 잘못된 습관은 좀처럼 고쳐지지 않는 법이다. 틀린 문제를 모아 오답노트를 만들고 반복하여 학습했던 것처럼 애자일에서는 과거의 경험에서 잘못된 부분을 찾아 개선하려는 고찰을 강조한다. 회고(retrospective)라는 말은 '뒤'를 뜻하는 retro와 '본다'라는 뜻의 spectare가 합쳐진 것이다. '뒤를 돌아본다'라는 것은 과거를 통해 미래를 개선한다는 의미이다. 프로젝트 중 끝나는 시점뿐만 아니라 개발주기, 일일단위로 수행한다면 날마다 새롭게 개선될 수 있는 기회를 얻는 것이다.

배움

스크럼을 만든 켄 슈와버(Ken Schwaber)는 소프트웨어 개발 과정은 잡음(noise)과 불규칙성으로 예측이 불가능하기 때문에 경험적 관리 모델(empirical Management Model)이 적합하다고 설명하였다. 즉, 경험해보며 고쳐나가야 한다는 말이다. 지식과 실천이 떨어져 있는 경우 지식은 이론으로 남게 된다. 무엇을 안다는 것은 이해하여 경험하는 과정을 필요로 한다. 아래 <그림3>에서 "I"는 입력물, "C"는 통제 "O"는 결과물을 말한다. 여기서 통제의 주기가 중요하다. 자주 피드백을 받을수록 현실을 잘 반영할 수 있고 개선의 기회가 많아지는 것이다. 피드백이 구체적이고 제때에 이루어질수록 학습효과가 향상되어 적은 노력으로도 좋은 결과물을 만들어내는 선순환이 이루어진다.

<그림 3> 경험적 관리 모델



일반적으로 프로그래밍의 고수와 초보의 코드를 비교해보면 초보자일수록 코드가 복잡하고 고수에 비하여 같은 기능을 수행함에도 불구하고 코드가 길다. 남의 코드에서 가져온 Copy & Paste된 코드들은 문제가 발생해도 손을 댈 수 없도록 만들어 버리는 원인이 된다. 배움의 본질은 타인의 지식을 자신의 것으로 소화하고 꼭 필요한 것만 남기는 것이다. 개발 프로세스가 완벽해지려고 노력할수록 복잡해지고 실제 개발 과정에서 아무도 쓰지 않는 정보들로 채워지기 쉽다.

직관

프로젝트에서는 예측하지 못한 다양한 문제와 사건들에 마주치게 마련이다. 그때마다 데이터를 모아서 철저한 분석을 통해 의사결정을 내린다는 것은 불가능하거나 때늦은 결정되기 쉽다. 신경과학자인 리처드 레스택(Richard Restak)은 그의 저서인 "The Brain Has a Mind of Its Own"에서 뇌는 의식적 사고를 하지 않고도 복잡하고 예민한 의사결정을 낼 수 있다고 얘기한다. 문제에 부딪히면 의사결정이 꼭 필요한 시점까지 미루어 무의식이 최대한 작용할 수 있도록 관용적 태도를 갖는 것이 필요하다.

XP에서는 코드에서 리팩토링(Refactoring) 대상을 찾을 때 나쁜 냄새를 맡으라고 한다. 가령, 코드가 중복되거나 메소드가 길거나 클래스가 거대하다면 나쁜 냄새가 있는 것이다. 코드뿐만 아니라 프로젝트에서도 나쁜 냄새를 맡을 수 있다. 회의에서 서로 문제만 지적하고 결론이 없다거나, 팀원 간에 대화가 없다면 잘못되어 간다는 신호이다. 문서에만 의존하지 말고 직관적으로 냄새를 맡아 문제점을 조기에 해결하는 것이 효과적이고 효율적이다.

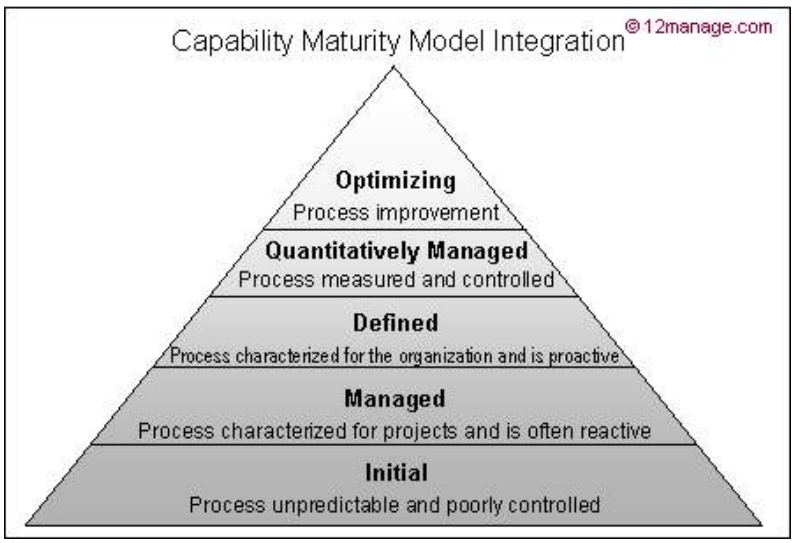
간혹 내가 정말 원하는 일이 무엇인지 고민스러울 때가 있다. 고민이 깊어질수록

해답은 더욱 찾기 어려워진다. 이때도 직관의 힘을 빌려보면 도움이 된다. 비판적 생각에 쫓기지 말고 방해받지 않는 공간을 찾아서 가만히 마음에 귀를 기울여보자. 의외로 쉽게 답을 찾을 수 있을지 모른다.

성장

일반적으로 성장이라는 말은 아이가 어른으로 커가는 것처럼 현 수준에서 윗단계로 올라간다는 뉘앙스를 담고 있다. CMMI(Capability Maturity Model Integration) 모델에서 Level 1인 초기(Initial) 단계에서 능력성숙도가 개선됨에 따라 Level이 올라가는데 최고 단계인 Level 5가 최적화(Optimizing)단계이다. 더 올라갈 곳이 없으니 지속적인 개선을 하라는 것이다. 애자일에서의 성장은 이 단계처럼 변화를 추구하는 지속성에 관한 것이다. 애자일에서 성장의 한계는 능력의 부족에서 오는 것이 아니라 변화를 받아들이는 열정의 부족에서 나온다.

<그림 4> CMMI 모델



(자료 인용: 12manage.com)

변화를 추구하는데 혼자 하는 것보다는 함께 나눌 동료가 있는 것이 유리하다. 문제를 함께 고민해주고 격려해주는 동료를 만들어보자. 그런데 변화를 만드는데 있어 중요한 것이 지향하는 목표이다. 애자일에서 지향하는 목표란 무엇일까? 필자는 고객에게 매일 가치를 전달하는 것이라고 생각한다. 계획된 일을 완료했다고 고객에게 가치가 전달되는 것은 아니다. 고객에게 보다 높은 가치를 어떻게 전달할 수 있을까를 하루에도 몇 번씩 고민을 나눈다면 사람이든 제품이든 반드시 성장할 수 있을 것으로 본다.

<참고 자료>

1. "Enough Process – Let's Do Practices", Journal of Object Technology, Ivar Jacobson, 2007
2. The Standish Group CHAOS report (프로젝트 성공요인 분석 보고서), http://blog.standishgroup.com/pmresearch
3. The Brain Has a Mind of Its Own, Richard Restak, Crown Publishers, 1993
4. 미국 프로젝트 관리자 협회, http://www.pmi.org/PMBOK-Guide-and-Standards.aspx
5. 애자일 이야기, http://agile.egloos.com
6. 소프트웨어 프로세스 이야기, http://swprocess.egloos.com.