

연 + 재 + 순 + 서

- 1회 2003.12 | 개발자인가 소프트웨어 엔지니어인가?
- 2회 2004. 1 | 내가 계획한 일정, 관리자도 인정한다
- 3회 2004. 2 | 결합, 시간이 지나면 시간을 속내는 버그다
- 4회 2004. 3 | 나만의 프로젝트, 프로젝트 매니저로 가는 지름길이다
- 5회 2004. 4 | 열심히 일한 당신 떠나라! 당신의 미래를 위해서

한용수 | yshan@psptsp.com

현재 사단법인 소프트웨어공학협회 (SI)PSP 분과위원장으로 수회에 걸쳐 정통부의 지원으로 (SI)PSP 세미나 및 교육과정을 진행했고, 다양한 세미나를 통해서 국내 보급에 심혈을 기울이고 있다. 최근, 미트교육센터에서 네트워크 전문가 과정에 (SI)PSP 과목을 강의하고 있으며, 또한 (SI)PSP 강사를 양성하기 위해 과정 전개중이다

소프트웨어 분야에 종사하기 위해선 무엇보다 개발을 이해할 필요가 있다. 개발과정의 문제를 이해하고 극복하기 위해 현실적 대안을 마련하는 것은 프로젝트에서 개발실무를 하지 않으면 단순한 이론에 불과할 수 있다. PSP/TSP를 소개하는 목적 중 하나는 실무에서 문제를 경험으로 빨리 축적하고 성장할 수 있는 계기를 만들기 위해서다.

PSP로 나도 SW 개발 전문가 V

열심히 일한 당신 떠나라!
당신의 미래를 위해서

지난 4개월간 소프트웨어 엔지니어로서 필요한 자기관리 방법을 다루었다. 마지막이 될 이번 시간에는 지난 내용을 정리하고 PSP 내용을 설명해 독자 여러분이 미래를 개척하는데 도움이 될 수 있는 자료를 제공해 보고자 한다.

소프트웨어 개발환경이 변하고 있다

먼저 소프트웨어 엔지니어의 환경이 변화하고 있음을 파악할 필요가 있다. 소프트웨어 엔지니어에게 자신만의 '예술 같은' 코드를 '번개 같이' 만들던 시기는 지나갔다. 이제는 표준화된 코드를 지정한 일정에 따라 팀과 함께 만들어야 한다. 예술적 감각으로 까지 칭송하던 코드를 만들었다고 해도 이제는 더 이상 관심을 끌지 못하고 있다. 컴퓨터를 처음 대하던 시절 예상하지 못한 프로그램을 내놓으면 예술이라고 격찬했다. 80년대 8비트 컴퓨터가 보급되면서 사람들에게 믿을 수 없는 다양한 기능들이 선보였다. 그러나 이제는 섬세한 윈도우 프로그램을 누구나 사용하고 있기 때문에 아주 뛰어난 프로그램이 아니면 일반 사용자들의 눈에 띄지도 않는다. 또한 프로그램 크기가 매우 커져서 10년마다 40배 이상 커지고 있고 많은 사람이 함께 소프트웨어 하나를 만들어야 하기 때문에 팀뿐만 아니라 개인도 정확한 관리방법을 가지고 있지 않다면 실패를 반복하게 된다.

고객은 하루 빨리 제품을 사용하고 싶다

소프트웨어 크기가 늘어가고 있지만 고객은 기업경쟁에서 하루 빨리 신규 소프트웨어 시스템을 사용하고 싶어 한다. 경쟁업체가 새로운 시스템을 도입한다고 소문이 나면 너도나도 덩달아 시스템을 도입한다. 출발은 늦었지만 더 빨리 새로운 소프트웨어로 무장하고 경쟁에서 이기고 싶은 생각에 무리한 공급 일정을 제시하기도 하고 때로는 정말 터무니없는 공급일정을 요구하는 경우도 있다. 그렇다고 단순 기능을 원하는 것도 아니다. 최소한 경쟁자와 동일하거나 앞선 기능을 요구한다.

꼭 찬 일정을 설정하고 단 한 시간도 놓칠 수 없는 프로젝트에서 관리자를 비롯한 모든 개발자들이 정신없이 작업을 하지만 매일매일 발생하는 새로운 상황을 대처하다 보면 정작 해야 할 일들은 밀리고 막바지에 저녁과 주말까지 작업하지만 결국 예정 일정을 넘기고 만다. 이런 현실에서 어느 누구도 정확한 일정을 계획할 수 없다. 오직 엔지니어만 대체로 언제쯤 일을 마칠 수 있는지 감을 가지고 있을 뿐, 나머지 사람들은 끝나기를 기다리는 수밖에 없다. 그래서 소프트웨어 계획은 반드시 엔지니어를 통해서 수립하는 것이 올바른 해답이다. 프로그램 크기와 개발일자를 정확하게 예측할 수 있는 엔지니어의 능력은 오랜 경험에서 나온다. 앞으로 이런 경험을 짧은 기간에 얻을 수 있도록 해야 한다.

프로젝트의 절반은 테스트 시간이다

엔지니어가 일정 수립에 관여하기 위해선 효율적인 작업을 바탕으로 해야 한다. 제한된 시간에 작업을 마치기 위해선 효율을 높여야 한다. 개발 효율에 가장 문제를 일으키는 요소가 결함제거를 위해 많은 테스트를 한다는 점이다. 작은 프로그램을 만들어도 컴파일과 테스트 시간에 30% 이상 소요되는 것을 자주 본다. 완벽한 코드는 한번의 컴파일과 한번의 테스트로 개발이 완료된다. 그러나 엔지니어도 인간인 관계로 다양한 결함을 습관적으로 만들기도 하고, 혹은 잘못 이해하거나 적용하면서 결함이 만들어 진다.

또한 소프트웨어 크기가 커지면 그에 따라 결함은 더욱 심각한 상황으로 치닫게 된다. 소프트웨어는 겉보기와는 전혀 다른 코드로 만들어졌기 때문에 겉으로 드러나는 오류만 보고서 어떤 결함이 있는지 정확한 예측이 불가능하다. 만일 오류가 간접적인 형태로 나타날 경우 더욱 어렵다. 그래서 테스트를 통해 결함을 수정하는 작업은 많은 시간이 필요하다. 때론 증상을 알고 어느 부분이 문제인지 밝혀졌지만 전혀 수정할 수 없는 경우도 있다. 시간과의 싸움에서 숨어 있는 결함을 얼마나 잘 찾느냐가 승패를 가르게 된다. 이 싸움은 개인의 몫이기도 하지만 프로젝트 전체의 운명을 좌우하기도 한다. 궁극적으로 엔지니어 경력에 치명타를 남길 수 있다.

프로젝트는 규모에 관계없이 공통사항이 있다

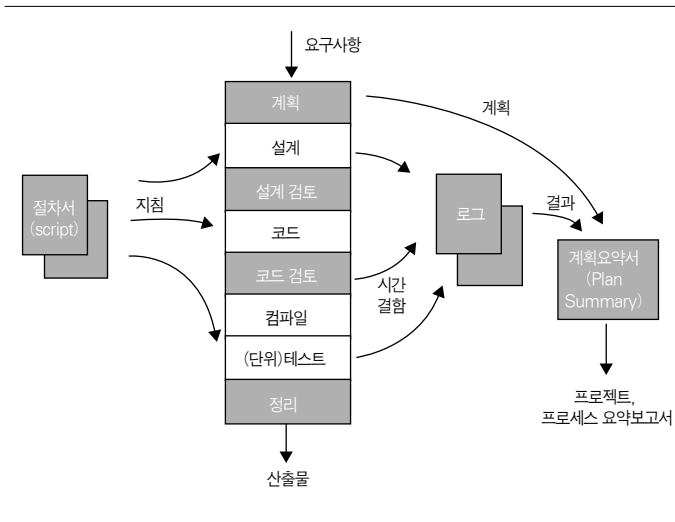
이런 상황들은 마치 외줄타기와 같이 항상 위험을 다뤄야 한다. 이런 위험을 최소화하기 위해 프로젝트 계획서를 작성하고 프로젝트 상황을 추적하면서 최종 성공에 이르기까지 항상 긴장한다. 위험관리 활동은 개인에게 있어서 동일하다. 각 개인의 작업결과가 모여서 프로젝트 결과가 되듯이 각 개인에게 있어서 역시 작지만 개인 프로젝트라고 할 수 있다. 팀 프로젝트에서 일정관리 하듯이 개인도 일정관리를 해야 하고, 팀 프로젝트가 품질관리를 하듯이 개인도 품질관리를 해야 한다. 그리고 개인이 효율적으로 일할 때 비로소 팀 프로젝트의 효율도 오르게 된다.

차이가 있다면 프로젝트는 여러 사람이 함께 하기 위해 원활한 의사소통과 서로 이해하고 협력하는 환경을 조성해야 한다. 이 부분도 자세히 보면 프로젝트가 고객과 지속적인 대화가 필요하듯 개인도 팀과 지속적인 대화가 필요하기 때문에 전혀 다른 것만은 아니다. 자신이 현재 하고 있는 작업을 훌륭하게 해낼 수 있다면 여러 사람이 함께하는 프로젝트도 완벽하게 할 수 있다. 그래서 자신이 현재 하고 있는 작업뿐만 아니라 작업을 관리하는 방법을 배우면 프로젝트 역시 쉽게 관리할 수 있다.

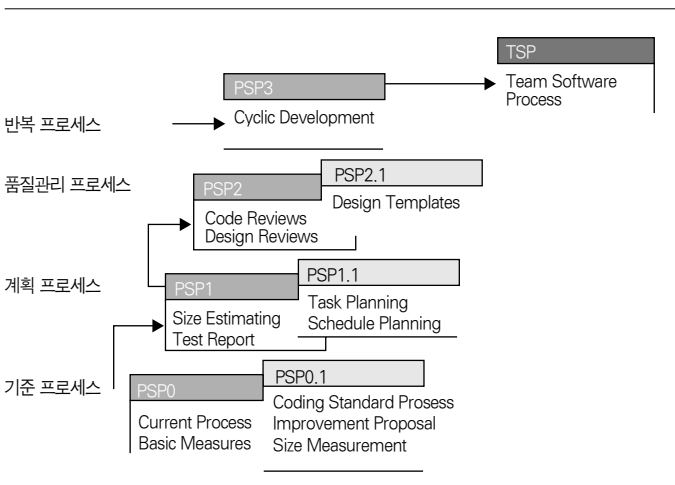
PSP와 TSP

PSP는 초급 엔지니어에게 출발부터 좋은 습관을 가질 수 있도록 하고, 고급 엔지니어에게 그 동안 가지고 있는 습관에 잘못이 없는지 확

<그림 1> PSP 프로세스 흐름



<그림 2> PSP 학습 단계(PSP Evolution)



인하고 교정할 수 있는 길을 제시한다. PSP 틀은 소프트웨어 개발과
 정에서 적절한 일정계획 수립과 품질(결함)관리를 통해서 좋은 프로
 그램을 만들 수 있도록 가이드한다. 학습과정에서 소프트웨어 개발자
 는 자신의 습관과 능력을 정확히 판단할 수 있고 프로그램 오류를 현
 처히 줄여 양질의 소프트웨어를 개발하는 방법을 배우게 된다. 그리
 고 소프트웨어 엔지니어가 자신의 작업내용과 작업절차를 정확하게
 인식하고 자신이 직접 자신에게 알맞은 개발방법을 찾아 최고의 소프
 트웨어를 만들 수 있다. PSP 엔지니어는 최적의 방법을 가지고 최대
 효율을 얻을 수 있도록 지원하는 TSP(Team Software Process)를
 이용하면 기존의 어떤 결과보다 훌륭한 결과를 얻을 수 있다.

PSP 엔지니어는 지금까지 '코딩-디버깅'으로 구분된 작업을 '계
 획-설계-설계검토-코딩-코드검토-컴파일-단위테스트-정리'로 상세

히 구분하여 작업하게 된다(그림 1). 이 프로세스는 개발과정에서 각
 단계를 명확히 구분하여 일정을 쉽게 지킬 수 있는 틀을 제공하고, 설
 계검토와 코드검토 과정에서 결함을 최대한 제거하여 컴파일과 단위
 테스트를 짧은 시간 내에 마칠 수 있도록 했다. 그리고 프로세스에서
 개발시간과 프로그램 크기 기록은 새로운 소프트웨어 개발에서 개발
 규모와 필요시간을 예측할 수 있도록 하고, 컴파일과 단위테스트 과
 정에서의 결함기록은 품질을 결정할 수 있도록 품질 모형을 제공한다. 결
 함정보는 통합테스트, 시스템테스트, 인수테스트에 얼마나 시간이 필
 요한지 어떤 부분에 검사(inspection)를 집중해야 할지 판단하는 귀
 중한 자료가 된다. 마지막으로 PSP 기록은 자신의 장단점을 파악하
 고 스스로 자신에게 적절한 방법을 찾아 갈 수 있도록 길을 안내한다.
 이런 목적을 달성하기 위해서 PSP는 단계적 학습이 필요하다.

PSP/TSP의 학습 단계

단계별 학습과정은 엔지니어 스스로 자신이 현재 상태를 확인할 수
 있도록 되어 있다. 특히 실습을 통해서 배우고 익히는 동안 필요성을
 정확히 이해하고 활용법도 예상할 수 있다. 기존 프로세스(PSP0)는
 현재 상태를 파악하기 위한 단계이고, 계획 프로세스(PSP1)는 수집
 자료를 토대로 작업 대상의 규모를 예측하고 일정관리를 할 수 있다.
 품질관리 프로세스(PSP2)는 결함을 최소화하는 방법과 설계를 배운
 다. 품질관리 프로세스 단계까지 1주내 개발할 수 있는 작은 프로그
 램에 이용되지만 반복 프로세스(PSP3)는 수주 혹은 수개월짜리 개
 발을 어떻게 할 것인지 배운다(<그림 2>). TSP는 PSP 엔지니어가
 함께 일하는 방법을 제공할 뿐만 아니라 팀의 목표설정, 팀원의 역할,
 개발과정의 지속적인 모니터링을 지원한다. 특히 프로젝트 계획에서
 팀 전체가 함께하는 계획수립과정은 강력한 결속력으로 완벽한 팀
 (Jelled Team)을 구성할 수 있다.

기본 프로세스

먼저 첫 단계인 기본 프로세스는 현재 자신이 사용하고 있는 개발상
 황을 이해하는 단계다. 프로그램 개발에 사용하는 시간을 기록하고
 프로그램 개발과정에서 발생하는 결함을 기록한다(그림 3, 4). 그리
 고 작업결과가 얼마나 되는지 코드 라인수(LOC: Line of Code)를
 기록하여 자신의 개발능력, 결함처리상태를 확인할 수 있다. 프로그
 램 개발에서 얼마나 많은 작업량을 해낼 수 있는지 파악하는 것은 쉽
 지 않다. 오랜 경험을 통해서 예측할 수 있지만 측정해 보기 전엔 누
 구도 정확하지 않다. 또한 어떤 개발기간, 개발환경, 개발도구를 기
 준으로 하는가에 따라, 어떤 방법으로 크기를 계산하는가에 따라 다양
 한 답들이 나올 수 있다. 이 기본 프로세스의 목적은 현재 상태를 파
 악하는 것이기 때문에 가장 먼저 개발시간과 프로그램의 크기가 고려

<그림 3> 시간 기록 일지(Time Recording Log)

날짜	시작	종료 시간	인터럽트 시간	경과	활동	주석
27-Jul	19:00	20:19	0	29	Plan	2B
	20:20	21:06	0	46	DSGN	
	21:07	21:19	0	12	DR	
	21:20	22:24	0	65	CODE	
	22:25	22:41	0	17	CR	
	22:42	22:53	0	12	CMPL	
	22:54	23:15	0	22	TEST	
	23:16	0:09	0	55	PST	

<그림 4> 결함 기록 일지(Defect Recording Log)

Date	Number	Type	Inject	Remove	Fix Time	Fix Defect
27-Jul	1	20	CODE	CR	1	
Description missing						
27-Jul	2	50	CODE	CR	1	
Description miss match method argument type						
27-Jul	3	80	CODE	CMPL	2	
Description method define error - constraints						
27-Jul	4	80	CODE	TEST	3	
Description variable confusing						

되었고 소프트웨어 특성을 고려하여 결함을 분석대상으로 뽑고 있다.

자료수집은 개발시간을 단계별로 상세히 기록하고 개발과정에서 발생하는 사소한 결함이라도 수정시간과 함께 기록한다. 그리고 개발된 프로그램의 크기를 측정할 수 있는 기준을 마련하고 이 기준을 통해서 프로그램의 크기를 기록하게 된다. 이 기록과정에서 프로그램 작업에 사용한 시간, 개발과정에서 발생한 결함 수, 그리고 개발 코드량이 모이면, 평균 생산성(시간당 개발하는 프로그램 량)과 1000라인(KLOC)을 개발하면서 발생하는 결함 수, 계획-설계-코딩-검파일-테스트-정리과정에서 시간분배 형태를 알 수 있다.

모든 기록은 프로젝트 계획요약서(Project Plan Summery)로 정리한다. 이 요약서를 지속적으로 관리함으로써 자신의 능력변화를 파악한다. 수집자료는 앞으로 개발할 프로그램의 크기와 프로그램의 작성에 필요한 시간, 어디서 결함 발생율이 높은가, 그리고 어디서 결함을 가장 많이 제거 했는가 확인할 수 있다(<그림 5>). <그림 5>의 요약서는 크기, 시간, 결함자료만 표시하고 있지만 다음 단계를 거듭하면서 계획의 정확성, 품질 결과 등을 포함한다.

<그림 5> PSP0.1 프로젝트 계획요약서 예

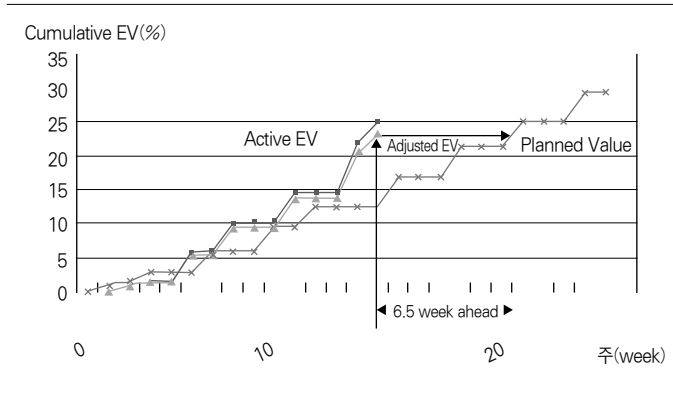
프로그램 크기 (LOC)	계획	실제	누적	
수집 전 (A)		0		
식별 (B)		0		
수집 중 (C)		0		
추가 (A)		130		
제거 (B)		37	37	
총 신규로 수집 (D)	150	130	342	
수집 후 LOC (E)		162	374	
신규 제거 (F)				
단계별 시간 (표)	계획	실제	누적	누적 %
계획	15	4	12	2.8%
식별	10	13	25	5.8%
수집	30	87	260	68.2%
입력됨	10	32	77	17.8%
(단위)테스트	10	1	11	2.2%
합계	20	7	27	6.3%
합계	95	144	437	100.0%
결함양성 (inserted)	계획	실제	누적	누적 %
계획	0	0	0	0.0%
식별	1	6	25.0%	
수집	0	19	75.0%	
입력됨	0	0	0.0%	
(단위)테스트	0	0	0.0%	
합계	7	24	100.0%	
결함제거 (removed)	계획	실제	누적	누적 %
계획	0	0	0.0%	
식별	0	0	0.0%	
수집	2	2	5.1%	
입력됨	4	16	72.3%	
(단위)테스트	1	4	18.2%	
합계제거	7	22	100.0%	
제거 후				

계획 프로세스

다음 단계는 수집자료를 계획에 활용하는 계획 프로세스다. 프로젝트 계획 요약서에서 생산성(LOC/hr)과 신규 개발 프로그램 크기를 확인하면 개발시간을 예상할 수 있다. 처음 개발 프로그램의 크기를 예측하는 경우 엉뚱한 결과를 얻을 수 있지만, 여러 차례 예측하다 보면 오랜 경험에 의존한 예측보다 더욱 신뢰할만 한 결과를 얻을 수 있다. PSP는 프로그램 크기와 개발시간을 예측하는 모델로 PROBE 방법을 사용할 것을 권하고 있다. 프로그램 요구사항을 보고 개념적인 설계를 한다. 그리고 각 구성요소를 특정 유형(Proxy)에 따라 얼마나 큰 프로그램이 필요한지 예측한다. 과거 통계자료를 이용하여 프로그램 크기와 개발시간을 계산하고, 계산 값이 현실성 있는지 검증한다. 만일 충분한 자료가 수집되지 않은 상태에선 엔지니어의 경험에 의한 산정방법도 이용할 수 있다.

자신이 맡은 프로그램 개발시간을 산정한 후 일정수립을 한다. 적절한 개발일정은 안정적인 작업환경을 제공하고 상대적으로 좋은 소프트웨어를 만들 수 있다. 그러나 개발일정이 부족한 경우 일정에 쫓겨 개발과정에서 많은 실수를 범하고 결함이 많이 발생한다. 이 결함

<그림 6> 일정 추적



<그림 7> 코드 검토 목록(Code Review Checklist) 예

PROGRAM NAME AND #	Step	Description	Y/N	Y/N	Y/N
	Purpose	To guide you in conducting an effective code review			
	Prerequisites	As you complete each review step, check that you are the one to do the right. Complete the checklist for one program and before you start to review the next.			
	Complete	Verify that the code meets all the design.			
	Actions	Verify that modules are complete.			
	Validation	Check variable and parameter declarations - at program initiation - at start of every loop - at machine level entry			
	Code	Check function call format - classes - packages - variable names			
	Errors	Check name spelling and use - is it consistent? - is it well declared scope?			

은 많은 테스트를 반복하면서 찾아야 하기 때문에 일정이 더 지연될 수 있다. 반면에 개발일정이 충분한 경우 심리적으로 게으름을 피울 수 있고 그림으로써 막바지에 이르러 서둘러 작업 마무리하면서 역시 많은 결함이 발생할 수 있다. 일정관리는 이러한 두 상황을 극복하기 위한 방법을 제공한다(<그림 6>).

품질관리 프로세스

계획 프로세스가 계획된 기간 내 일을 마칠 수 있도록 방법을 제공하고 있지만 궁극적으로 생산성과 효율을 지속적으로 높이지는 못한다. 이러한 단점으로 품질관리 프로세스가 더 나은 제품을 더 빨리 만들 수 있는 방법을 제공하고 있다. 결함을 늦게 발견할수록 비용과 시간은 급격하게 증가한다. 소프트웨어에서 생산성과 효율성을 높이는 방법은 결함잔류 시간을 최대한 줄이는 것이다. 계획과 설계단계에서 요구사항 결함을 찾고, 설계검토에서 설계결함을 찾고, 코드검토에서 코드결함을 찾아 테스트결함을 최대한 줄이는 방법이 PSP에서 요구하는 품질관리 방법이다. 스스로 자신이 만든 설계와 코드를 위해서

<그림 8> 설계 검토 목록(Design Review Checklist) 예

PROGRAM NAME AND #	Step	Description	Y/N	Y/N	Y/N
	Purpose	To guide you in conducting an effective design review			
	Prerequisites	As you complete each review step, check that you are the one to do the right. Complete the checklist for one program and before you start to review the next.			
	Complete	Verify that the requirements, architectural, and high-level design are completely covered by the design. - all specified modules are produced - all needed scope are included - all required modules are stated			
	Code	Verify that program requirements in program - that stacks, loops, etc. are in the proper order - that expressions are made properly - Verify that all loops are properly initiated, incremented, and terminated			
	Errors/Warnings	Check all special cases - empty, null, maximum, minimum, negative, zero - set values, overflow, underflow - ensure 'impossible' conditions are absolutely impossible - handle all control input conditions			

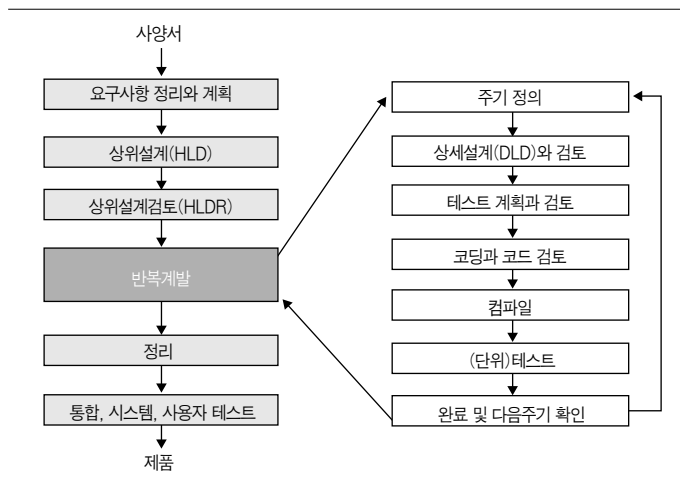
PSP는 자신에게 맞는 검토목록(checklist) 사용을 권장하고 이러한 방법은 70% 이상의 결함을 찾아내고 있다(<그림 7, 8>).

코드결함의 대부분이 컴파일과 코드검토로 제거되지만, 설계결함은 설계검토로 제거하더라도 수많은 테스트가 필요하다. 품질관리 프로세스는 설계결함을 원천적으로 줄일 수 있도록 설계체계와 설계내용 검증방법을 제시한다. 특히 설계검증 방법은 수학적인 모델을 사용하여 자동으로도 검증할 수 있는 길을 제시한다.

반복 프로세스

반복 프로세스는 이전 과정과는 많이 다르다. 이 단계는 PSP의 완성 단계로 한 사람이 시스템을 만들어 가는 방법을 제공한다. 앞서 배운 일정관리를 이용하여 전체 개발과정을 관리하는 방법으로 TSP와 함께 사용할 경우 더 큰 시스템을 만들 수 있는 방법이다. 반복 프로세스는 시스템의 일부를 혼자 개발하는 방법으로도 매우 효과가 있다.

<그림 9> PSP3 프로세스 흐름



이 프로세스는 엔지니어가 충분한 능력을 갖추고 있어야 가능하다. 엔지니어는 요구사항이 어떤 내용을 다루는 것인지 충분한 업무지식이 필요하고 그것을 문서화할 수 있어야 한다. 그리고 기반기술 이해와 시스템 설계능력과 함께 소프트웨어 형상관리, 위험관리에 능숙해야 한다. 즉 혼자서 프로그램 일부를 주관적으로 만들어야 하는 상황이나 전략적으로 최고의 엔지니어들이 최단 시간내 시스템을 구축해야 할 경우 매우 효과가 있고, 초보 엔지니어의 경우 많은 위험이 따른다(그림 9)).

TSP

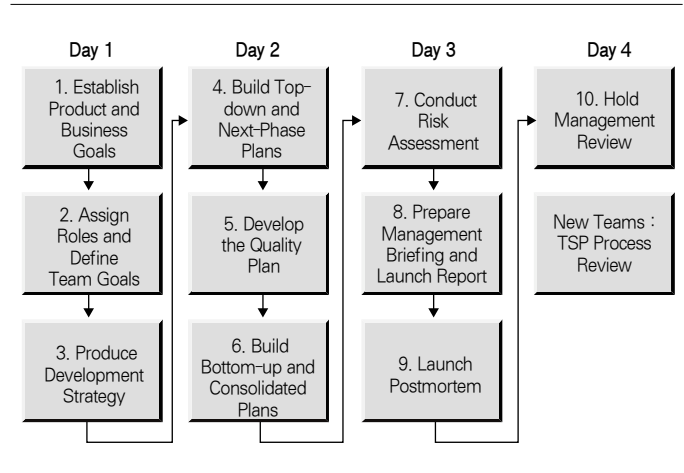
PSP로 최정예 엔지니어가 탄생한다. 아무리 뛰어난 능력을 가진 엔지니어도 보통 환경에 놓이게 되면 장점을 활용할 수 없다. 최정예 엔지니어들이 최고의 결과를 만들 수 있도록 환경을 조성하고 프로젝트 진행 방법을 지원하는 것이 TSP다. 이미 계획/품질 관리력이 뛰어난 엔지니어에게 기업전략에 맞춰 일정을 수립하고 진행할 수 있는 권한을 줌으로써 성공가능성을 대폭 높일 수 있다. TSP를 적용하기 위해선 반드시 PSP 엔지니어를 양성할 필요가 있다. 그리고 엔지니어의 능력에 따라 적절한 PSP 레벨을 엔지니어별로 지정할 필요가 있다. 필요 이상의 높은 단계로 엔지니어 작업을 요구하면 효율이 떨어질 뿐 아니라 충분한 품질이 나오지 못하는 경우도 있다.

TSP는 별도의 학습과정을 가지지 않고 현장학습을 전제로 한다. PSP의 경우 정확하게 어떤 부분을 익혀야 할지 전제될 수 있는 반면에 TSP는 프로젝트마다 워낙 다양한 가능성이 있기 때문에 이를 구체적으로 지정하여 학습할 수 있도록 준비되어 있지 않다. 그래서 일반적으로 PSP와 TSP를 이해하는 노련한 전문가를 통해서 프로젝트 전체를 진행하는 과정에서 필요한 사항을 정의하고 진단하고 진행할 수 있도록 코치한다. TSP의 첫 과정으로 착수프로세스가 있다. 4일간의 과정에서 프로젝트의 목표와 전략을 세우고 요구사항 파악, 일정수립, 위험분석을 한다. 마지막으로 계획서를 만들어 경영전략과 일치되도록 조정과정을 거쳐 프로젝트 준비를 마친다(그림 10)). 이러한 계획과정은 3개월 단위로 반복하고(Relaunch) 지난 3개월 과정에서 문제점을 개선하여 다음 3개월을 준비하게 됨으로 매번 새로운 마음으로 시작할 수 있도록 한다. 이 과정은 프로젝트에 관계하는 모든 인원이 반드시 참여하여 정보를 교환하고 조정하면서 최적의 팀(Jelled Team)을 만드는 데 탁월한 효과가 있다(그림 11)).

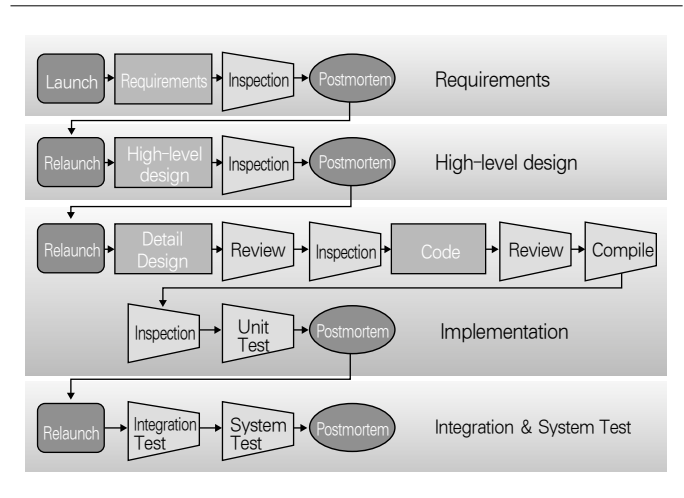
프로젝트가 진화한다

PSP/TSP를 프로젝트에 적용하는 데 있어서 소프트웨어 개발조직에 어떻게 영향을 미칠 수 있는지 살펴볼 필요가 있다. 엔지니어가 만든 상세한 데이터는 그 동안 소프트웨어 프로젝트를 관리하는 방법에 일

<그림 10> TSP 착수 프로세스(Launch process)



<그림 11> TSP 구조



대 혁신을 가져오게 된다. 상세한 자료는 개발자에게 현재 작업을 지속적으로 확인하고 발전할 수 있는 계기를, 프로젝트 관리자에게 프로젝트 진행상황을 경영자에게 각 프로젝트가 정상적으로 진행되고 있는지를 보여주게 된다. 이 때 결함자료가 첨부되면서 결함으로 일정에 얼마나 차질이 생길지 모니터링할 수 있는 가능성을 제공하여 위험이 증대되면 바로 처리방법을 모색할 수 있도록 지원하게 된다.

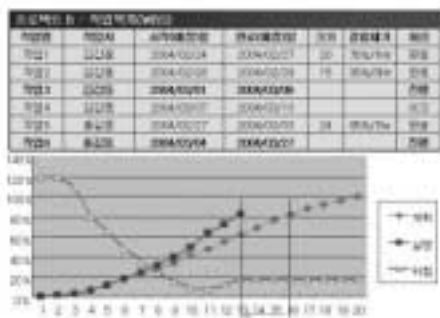
엔지니어는 스스로 발전한다

엔지니어는 자신의 작업결과를 정확히 보고할 수 있다. 단순히 완료 여부만 알릴 수 있던 상황에서 자신이 개발한 프로그램이 얼마나 크고 개발시간은 얼마나 사용했고, 그 과정에서 얼마나 결함이 제거되었는지 보고할 수 있게 되었다. 이 보고 결과는 과거 이력을 가지고 앞으로 테스트 시간이 얼마나 걸릴 것인지 예측이 가능하다(2004. 2

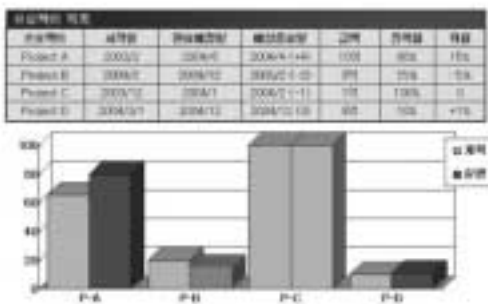
<그림 12> 엔지니어의 작업 현황



<그림 13> 프로젝트의 작업 현황



<그림 14> 경영자의 프로젝트 현황



월 기사 참조). 만일 자신의 작업이 예정보다 빠르게 끝났다고 하더라도 앞으로 남은 테스트에서 얼마나 많은 시간이 소비될 것인지 안다면 엔지니어는 다음 프로그램에서 더욱 적극적으로 결함을 제거할 수 있도록 자신을 스스로 독려하고 다음 일정에 차질이 생길 것으로 예상되면 현재 일정을 조정할 수 있게 된다(<그림 12>).

관리자는 현재 상황을 정확히 직시한다

관리자는 엔지니어로부터 받은 풍부한 자료를 이용하여 완벽한 프로

젝트 관리 체계를 운영할 수 있다. 그리고 이미 많은 부분의 정보를 엔지니어와 함께 공유하게 됨으로써 정확한 의사결정이 이루어지고 관리자는 좀 더 생산적인 업무를 추진할 수 있게 된다. 프로젝트 계획 과정은 팀 전체가 스스로 자신이 하게 될 일과 일정을 계획하고 이를 취합하는 과정으로 팀원은 서로 책임감을 가지고 더욱 친밀한 유대감으로 프로젝트를 진행한다(Planned Value). 실행과정은 각 작업자가 스스로 계획한 작업을 충실히 마치고 나서 작업결과를 정해진 양식으로 전달함으로써 관리자는 별도로 자료를 취합할 필요 없고 각종 회의시간도 대폭 절약할 수 있다(Earned Value). 그리고 프로젝트 착수부터 안정된 운영이 가능하므로 진행중 발생하는 위험요소를 명확히 식별할 수 있고 이러한 위험을 집중적으로 관리할 수 있는 운영체제로 전환된다. 그리고 결함으로 인한 테스트 기간의 변화를 미리 알 수 있음으로써 납기 문제를 사전에 예방할 수 있는 기회를 일찍 포착할 수 있다(<그림 13>).

경영자는 프로젝트별 상태를 확인한다

경영자에게 있어서 PSP/TSP는 가장 좋은 경영도구로써 자리 잡을 수 있다. 엔지니어가 작성한 자료를 바로 경영자가 이용하기 때문에 중간보고 과정을 거치면서 왜곡된 정보로 인하여 불합리한 결정을 내리지 않게 된다. 위험요소가 없고 결함만이 현재 일정에 영향을 미칠 수 있다고 볼 때 기업을 대표하는 결함제거율과 평균 결함제거 시간을 이용하여 해당 프로젝트가 계획된 통합테스트 이후에 어느 정도 기간이 필요한지 측정할 수 있기 때문에 정확하게 프로젝트의 지연 가능성을 한눈에 확인할 수 있게 된다. 프로젝트 비용과 경중에 따라 적절한 대응전략으로 실제 프로젝트를 지원하고 앞으로 어떠한 전략으로 차기 프로젝트를 진행할 것인지 명확한 그림을 그릴 수 있다(<그림 14>).

PSP, TSP를 적용한 효과

PSP는 학습과정에서부터 그 효과가 나타난다. 프로그램 크기와 개발 시간이 더 정확하고 테스트결함은 현저히 줄어든다. 예상한 시간에 프로그램 개발완료 가능성도 높아진다. 테스트과정에서 결함을 줄이기보다는 설계나 코드검토로 결함을 줄이는 것이 효과적이라는 사실도 체득하게 된다. 더욱이 이러한 체득은 설계와 코딩에서 결함이 없도록 노력하여 코드 100라인(LOC)에 결함이 10개 정도 나타났던 상황이 절반(5개 정도)으로 줄어든다는 사실은 매우 시사점이 크다.

20개 TSP 프로젝트 결과를 보면 다음과 같다(참조 : Noopur Davis et al, 'The Team Software Process (TSP) in Practice: A Summary of Recent Results' CMU/SEI-2003-TR-014, 2003). 첫 번째로 볼 수 있는 것은 생산성 증가와 결함 감소다. 그러면서도 공급

일정을 맞춰 공급할 수 있게 됨으로 많은 조직에서 엔지니어가 매우 만족하고 이직률이 급격히 떨어졌다. 특히 우리가 관심을 가지고 있는 생산성은 무려 78%가 증가하고 결함으로 인한 품질비용(COQ; Cost of Quality)은 전체적으로 30%가 줄어든다. 비록 PSP에서 검토가 추가되고 TSP에서 조사(Inspection)가 추가되지만 테스트 비용이 58% 줄어들게 됨으로써 테스트 시간도 그와 상응하여 절대적으로 줄어든다(〈표 1〉).

일정은 모든 PSP 프로젝트에서 계획대비 -20%에서 +27% 오차가 나타났다. 평균적으로 6% 정도 일정에 벗어난 것은 TSP를 적용하지 않은 프로젝트가 계획일정 120% 이내에서 완료될 가능성이 32%이고 프로젝트 50%가 취소되거나 100% 이상 지연된다는 사실을 감안하면 경이적인 결과다(〈그림 15〉).

결함을 비교하면 더욱 효과를 얻고 있다는 것을 알 수 있다. 전 세계적으로 몇 안되는 CMM Level 5 조직보다 17배 이상 결함밀도가 낮다. 별도의 프로젝트 관리와 소프트웨어 관리방법이 없는 조직(CMM Level 1)보다는 125배나 낮은 수치다(〈그림 16〉).

이러한 결과를 보면 TSP를 적용하면 짧은 시간에 제품을 납품하면서도 결함은 적고 엔지니어들은 작업에 만족한다. 즉 더 이상의 야간작업이나 주말작업이 없어도 소프트웨어 개발은 정해진 기간에 공급이 가능하다는 것이다.

PSP/TSP 미래

PSP/TSP는 그 역사가 이제 10년이 되고 있다. 그 동안 많은 조직에서 PSP/TSP가 정말 효과가 있는지 의문을 가지고 적용여부를 검토하고 있었다. 이제 관련된 다양한 결과가 만들어지기 시작하면서 어떻게 활용할 것인지 눈을 뜨기 시작했다. 첫 번째 나타난 반응은 프로젝트 효율을 높이기 위한 것으로 이미 이야기되었고, 두 번째는 시스템 보안성을 높이는 데 이용하는 것과 소프트웨어 구매 요소로서 적용하고자 하는 노력이다.

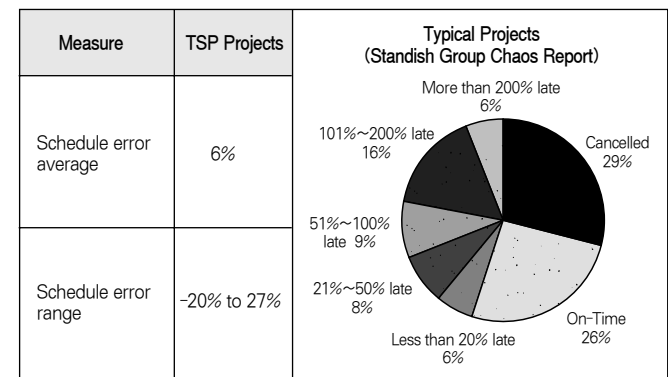
Security를 지원하는 PSP

많은 보안 관련 문제가 소프트웨어 결함으로 인해 발생했기 때문에 결함을 최대한 줄이는 방법으로 PSP가 적용할 수 있음을 보여주고 있다. 소프트웨어 결함의 5%가 보안문제를 야기시킬 수 있다는 점이다. 일반적으로 고객에게 공급되는 소프트웨어의 품질은 1000라인 당 2개 정도 결함이 발견된다고 한다(2 defects/KLOC). 만일 10만 라인 프로그램이 공급되면 200개의 결함이 있고, 이 결함 중에 5%인 10개의 결함이 보안문제를 가지고 있다. 만일 PSP/TSP를 적용하면 0.06 defects/KLOC로 10만라인의 프로그램은 6개의 결함이 있고 이중 5%인 0.3개가 보안문제를 야기시킬 수 있다. 이러한 문제를 구

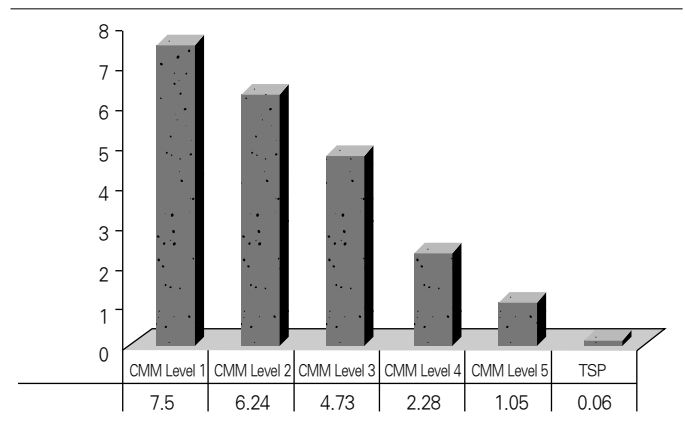
〈표 1〉 생산성과 품질비용

Measure	Average
Productivity improvement	78%
Failure COQ reduction	58%
Total COQ reduction	30%

〈그림 15〉 프로젝트 일정 지연



〈그림 16〉 공급된 소프트웨어의 결함밀도



체적으로 밝혀 보기 위해 SEI(카네기멜런대학교 소프트웨어공학연구소)는 협력자를 찾고 있다.

제품 품질을 표시하는 PSP

소프트웨어가 대형화되면서 상용제품을 사용하고 나머지 부분을 개발하고 있다. OS, 데이터베이스, 웹 서버, 애플리케이션 서버, PC, 네트워크 모니터링 툴 등을 구매하는 것은 물론이고, 개발 툴까지 구매한다. 어느 조직이건 순수하게 모든 소프트웨어를 개발하지는 않는다. 완제품으로 만들어진 상용 소프트웨어 COTS(Commercial Off-The-Shelf)를 구매하건, 아니면 반제품으로 만들어진 상용 소프트웨

어 MOTS(Modifiable Off-The-Shelf)를 이용하건 간에 소프트웨어를 구매하지만 결함이 얼마나 있는지 알 수 없다. 새로 개발한 소프트웨어 역시 얼마나 결함이 있는지 알 수 없다. 소비자(고객)에게 아직까지 제품이 어느 정도 신뢰할 수 있는지 누구도 충분히 자료를 제공하지 못하고 있다. 소프트웨어는 눈으로 직접 품질을 확인할 수 없기 때문에 어떻게 품질을 보증하고 구매하는 기준을 설정해야 하는가가 불분명하다. 이러한 고민을 해결하는 방법으로 SEI는 PSP/TSP를 통해서 확보하는 최종 테스트 과정의 결함밀도와 소프트웨어 규모로 신뢰성을 표시하려고 한다. 물론 현재 이러한 문제를 SQA(서비스품질보증)로 어느 정도 지원하고 있지만 아직 구체적으로 표시하지 못하는 한계점을 가지고 있다.

소프트웨어 엔지니어의 입지 변화

지금까지 PSP/TSP에 대한 전체 내용을 살펴보았다. 소프트웨어 개발은 비교적 단순하게만 여겼지만 실제 그렇지 않음을 확인했다. 이러한 문제가 밝혀지고 적용되는 사이에 국제적으로 많은 변화가 있었다. 그런 변화를 살펴보고 엔지니어에게 자신의 방향을 설정할 수 있는 몇 가지 자료를 소개하고자 한다.

최근 발행된 비즈니스 위크(Business Week)가 특별기사로 미국 내 소프트웨어의 명암을 소개했다. 10년 가까이 호황을 누렸던 소프트웨어 산업분야에 일대 돌풍이 예견된 것은 어제 오늘의 일만은 아니지만, 특별기사로 다루면서 미국과 인도 최고학부 인재 두 사람의 서로 상반된 미래를 보여주고 있다. 미국에서 그 많던 취업설명회조차 딱 끊어졌고 자신이 무엇을 해야 할지 감을 잡지 못하고 있는 상황과 그와 반대로 인도에서 미국이 투자한 연구기관에 이미 취업결정이 되었지만 자신이 원하는 일이 아닐 때 과감히 새로운 길을 개척하겠다는 의지를 보여준 두 사람의 IT 산업의 명암을 너무나도 뚜렷하게 그리고 있다.

미국은 그 동안 인건비를 줄이기 위해서 인도에 많은 소프트웨어 개발을 의뢰했다. 저가의 노동력에 유창한 영어는 심지어 고객센터를 인도에서 운영할 만큼 입지조건을 갖추고 있기 때문에 미국 소프트웨어 산업은 인도로 이전하고 있다. 그러나 미국은 희망을 가지고 있다. 단순 저가 작업은 해외로 이전해도 부가가치가 높은 부분은 계속 유지하고 있기 때문이다.

1 설계 분야

미국 내 몇 천 명밖에 되지 않은 직종으로 복잡한 업무 시스템을 설계하고 미래를 예측한다. адам 보스워드는 BEA 시스템 수석설계자로 그 예다. 연봉 15만?25만 달러. 해외 해외조달 가능성 적음

2 연구분석 분야

미국에 있어서 결정적인 혁신을 가져온 인력으로 약 2만5천 명 정도다. 대부분 안정된 대학에 있다. 연봉 5만(학교)~19만5천 달러(연구소). 비교적 안정되어 있지만 해외에서 조달될 수 있다.

3 컨설팅 분야

비즈니스 지식을 가지고 있는 컨설턴트는 고객에 맞는 기술을 조언하고 새로운 소프트웨어를 설치하고 새로운 애플리케이션을 만든다. 연봉 7만2천~20만 달러. 고객은 컨설턴트와 함께 직접 대화하기를 원하기 때문에 미국 내에서 안정적이다.

4 프로젝트 관리 분야

국제적인 소프트웨어 기업에 있어서 결정적 역할을 한다. 적시에 필요한 소프트웨어를 만들기 위해서 다양한 문화와 시간대를 적절히 조정하는 역할을 한다. 연봉 9만6천~13만 달러. 좋은 능력을 가진 관리자는 전혀 문제가 없다. 지난 2년 동안 14.3%나 연봉이 늘었다.

5 업무분석 분야

애매한 입장이기는 하지만 미국의 업무분석가는 10만 명으로 업무 요구사항을 찾아내고 프로그래머 사양서를 작성한다. 고객과 개발자가 떨어져서 일하기 때문에 핵심적인 역할이 되었다. 연봉 5만2천~9만 달러. 충분한 대외능력과 업무지식이 있다면 프로그래머에게 있어서 비교적 안전한 도피처가 되고 있다.

6 프로그래머 분야

정보산업의 보병이다. 애플리케이션 코드를 만들고 수정하고 테스트한다. 미국 전체 소프트웨어 산업에 있어서 1/30이나 되는 1백만 명이 소프트웨어 엔지니어와 프로그래머다. 연봉 5만2천~8만1천 달러(2002년 이후 15% 줄어 들). 가장 많은 일자리가 해외로 이전되고 앞으로 6년 내 나머지 16%가 이전될 것으로 예측하고 있음

국내 소프트웨어 상황도 만만치 않다

그러나 우리의 현실은 너무나도 많은 문제를 안고 있다. 힘들고 어렵고 위험한 3D 업종에 소프트웨어 개발이 포함되고 있다. 이공계 기피 현상을 비롯하여 고된 업무, 그에 비해 턱없이 부족한 보상체계, 숨가쁜 기술변화에 교육지원 부족, 우수한 엔지니어의 관리직 전환 혹은 전직 등은 소프트웨어 산업분야에 명암을 어둡게 하고 있다. 고만고만한 엔지니어로 구성된 프로젝트 진행은 더더욱 어려워지고 고객은 별반 시덥지 않은 제품공급을 우려해 이미 적정 보상체계를 무시하고 있는 상황으로 국내 소프트웨어 산업이 고사하고 있다는 절규를 다시금 새겨볼 필요가 있다.

이러한 절규는 조만간 아비규환으로 바뀌지 않을까도 생각한다. 이미 미국에선 인도를 이용하면서도 많은 부분을 보상할 수 있는 저력이 있지만 국내 사정은 전혀 다르다. 우리나라 고객의 불분명한 표현을 이해하고 요구사항을 작성하여 시스템을 만들 것인가 생각할 필요가 있다. 중국 조선족을 이용하는 방안도 이야기되고 있지만 아직 조선족을 이용할 만큼 우리가 정확한 요구사항을 만들고 관리할

수 있는 능력이 되었는가를 자문해 볼 수 있다. 또한 국제적 소프트웨어 프로젝트를 얼마나 잘 수행할 수 있는지 고려해봐야 한다.

전문화 시대가 되어야 한다

소프트웨어 개발은 다양한 분야가 있다. 예전엔 단순히 개발자만 있으면 모든 것이 해결되었지만 이제는 개발자만으로 해결이 되지 않는다. 다양한 전문가들이 한데 모여야 번듯한 소프트웨어를 개발할 수 있다. 전문가가 되지 않으면 앞으로 자신감 있게 소프트웨어를 다루기 힘들다. 전문가는 적어도 자신이 전문으로 하는 분야에 대해서 최소한 문제점을 분석하고 해결할 수 있어야 할 뿐 아니라, 관련 분야와 통합할 수 있어야 하고 가치를 평가할 수 있는 안목을 가져야 한다. 자신의 전문지식이 정말 고객에게 도움이 될 수 있어야 한다. 그리고 전문가는 일을 함에 있어서 결과로만 승부하지 않고 결과를 만들어 가는 과정에서 고객과 잘 융화되어 효율적으로 이끌어 가야 한다. 정확한 상황과악에서 필요한 사항을 고객에게 전달할 수 있도록 계획을 짜고 추진하는 능력이 있어야 한다. 마지막으로 중요한 것은 전문가라고 하더라도 모든 분야를 다 알 수는 없다. 자신이 모르는 분야에 도움 받을 수 있는 네트워크를 개발해야 한다.

당신이 전문 분야는?

전문가가 되기 위해선 수많은 전문 분야 중에 자신에게 맞는 분야를 찾아야 한다. 어떤 사람은 코드 개발에 적성이 맞다고 하지만, 그 코드도 어떤 종류냐에 따라 사뭇 다르다. 업무용 프로그램인지, 아니면 게임 프로그램인지, 그도 아니면 제어용 프로그램인지 자신의 영역을 명확히 하는 것이 좋다. 물론 초기에 이러한 명확성은 그 가치를 인정받지 못할 수 있지만, 어느 정도 시간이 흐르면 자신의 영역에서 좋은 결과를 얻을 수 있다. 전문 영역을 구분할 수 있는 기준은 매우 많다. 그중에서 IEEE 컴퓨터 협회와 ACM 교육위원회가 진행하는 대학교육을 위한 컴퓨터 관련 과정을 소개하면 컴퓨터 분야를 컴퓨터 사이언스(Computer Science), 컴퓨터 엔지니어링(Computer Engineering), 소프트웨어 엔지니어링(Software Engineering), 정보 시스템(Information Systems)으로 나누어 필요한 지식이 무엇인지 정리하고 있다. 아직 완벽하다고 할 수 없겠지만 전문가로서 필요한 지식이 어떤 것들이 있는지 참조할만 하다. 이 중에서 소프트웨어 엔지니어링과 정보 시스템 분야를 보고 자신에게 필요한 것이 무엇인지 확인할 필요가 있다. 소프트웨어 엔지니어링은 프로그램 개발과 운영에 집중하여 엔지니어(개발자)가 관심을 가져야 할 내용을 주로 다루고 있고, 정보 시스템은 정보 발굴과 이용에 집중하여 업무분석에 적용할 수 있다(참조 : <http://www.computer.org/education/cc2001/>). 소프트웨어 엔지니어링은 'Final Draft of the Software

Engineering Education Knowledge(SEEK), 2003' 에 엔지니어에게 필요한 지식을 엔지니어 전문 영역별로 제시하고 있다.

전문지식을 얻으려면?

엔지니어의 경우 가장 시급한 전문지식은 소프트웨어 엔지니어링에 대한 지식이다. 대학이나 대학원에서 소프트웨어 엔지니어링에 대해서 강의를 듣지만 워낙 방대하고 현실적인 측면과 상당한 거리감으로 많은 사람들이 현장적용을 못한다. 그러나 시간이 지남에 따라 다양한 전문지식을 필요로 하고 이러한 지식은 앞서 설명한 SEEK나 SW EBOOK(Software Engineering Body of Knowledge, 2003)를 참조하면 상세한 자료를 구할 수 있다. SWEBOOK은 소프트웨어공학에 필요한 지식을 체계적으로 정리해서 기본적인 이해를 돕고 있고 좀 더 상세한 자료를 참조할 수 있도록 분야별로 참고할 자료목록을 제공하고 있다. 최근 SWEBOOK 역시 Project Management Professional과 비슷하게 전문가자격인증 제도를 도입하려고 한다. 이제 이런 분야를 소개하고 설명하는 국내 서적들이 차츰 늘고 있어 조만간 다양한 정보를 쉽게 접할 수 있을 것으로 기대한다.

미래를 개척하는 밑바탕이 되길 바라며

소프트웨어 분야에 종사하기 위해선 무엇보다 개발을 이해할 필요가 있다. 개발과정의 문제를 이해하고 극복하기 위해 현실적 대안을 마련하는 것은 프로젝트에서 개발실무를 하지 않으면 단순한 이론에 불과할 수 있다. PSP/TSP를 소개하는 목적 중 하나는 실무에서 문제를 빨리 경험으로 축적하고 성장할 수 있는 계기를 만들기 위해서다. 앞으로 여러분의 미래를 개척하는 데 중요한 밑바탕이 될 수 있을 것이라 기대한다. 그 동안 부족한 글을 열심히 읽어 준 독자들에게 감사드리며, 다음에 더 좋은 내용을 소개할 수 있기를 바란다. ☺

장리 | 박준상 | zenith@korea.cnet.com

[PSP/TSP 학습가이드]

처음 PSP를 배우고자 하는 사람이라면 「퍼스널 소프트웨어 프로세스(PSP) 입문, 피어슨에듀케이션, 2003」을 통해서 익힐 수 있으며, 더 많은 것을 원한다면 「A Discipline for Software Engineering, Addison-Wesley, 1995」을 참조하면 된다. TSP를 익히는 경우 기본적인 내용을 위해서 「Introduction to the Team Software Process, Addison-Wesley, 1997」을 그리고 현장에서 적용하기 위한 참조로는 「Winning with Software, Addison-Wesley, 2001」가 도움이 된다. 그리고 좀 더 많은 자료를 참조하고자 한다면 www.psptsp.com에서 정보를 얻을 수 있으니 참조하기 바란다.