論文
4-12-13

프로그램 언어 과정에서의 Personal Software Process(PSP) 교육

윤영현(YoungHyun Yoon)[1]

요약

소프트웨어 개발자가 우수한 품질의 소프트웨어를 생산하기 위해서는 먼저 소프트웨어 품질에 대해서 확실하게 인지하고 있어야만 하며, 고품질의 소프트웨어는 개발자의 사소한 실수도 포함되어서는 안된다. 프로그램을 개발하는 모든 소프트웨어 엔지니어들은 모두가 다른 개발 습관을 가지고 있으며, 각자의 개발 방법을 개선하기 위한 노력 또한 모두가 다르다. 보다 효율적인 소프트웨어 개발이 되기 위해서는 모든 개발자들은 자신이 현재 진행하고 있는 작업에 대한 개발 계획을 수립해야 하는데, 이러한 개발 계획은 반드시 자신의 과거 개발 경험에 근거한 개발 계획이어야 한다는 것이다. 소프트웨어 개발자가 이러한 경험 요소를 반영하여 자신의 개발 능력을 지속적으로 향상시키기 위해서는 사전에 잘 정의되고 평가가 가능한 개선 방법을 사용해야만 한다. Personal Software Process(PSP)는 소프트웨어 엔지니어가 자신의 작업을 조절하고 관리하여 그들의 작업 방식을 개선할 수 있도록 지원하기 위한 것이다. PSP에서는 소프트웨어를 개발하기 위해 필요한 각종 양식, 이 양식을 작성하기 위한 작성법, 그리고 개발 절차가 포함되어 있다. PSP를 적절하게 사용한다면, PSP는 개발자에게 과거의 경험 자료를 축적하고 분석한 결과를 제공함으로써 개발을 진행함에 있어 요구되는 여러 가지 개발 계획을 효과적으로 수립하고 이를 준수할 수 있도록 해 줄 수 있다. 따라서 PSP는 프로그래밍 언어 교육 과정에 있는 학생들에게 프로그램 훈련을 시키기에 아주 유용한 도구이다. 본 논문에서는 PSP에 대한 전체적인 개념을 설명하고 각종 프로그래밍 언어 교육 과정에서 PSP를 적용하는 방안을 제시한다.

Abstract

To produce quality software products, engineers must feel personally responsible for the quality of products. Superior products are not produced by mistake; engineers must strive to do quality work. Every engineer is different and has own process improvement method. To be most effective, engineers must plan their work and they must base their plans on their own personal data. To consistently improve their performance, engineers must personally use well-defined and measured processes. The Personal Software Process(PSP) is a self-improvement process designed to help software engineers control, manage, and improve the way they work. It is a structured framework of forms, guidelines, and procedures for developing software. Properly used, the PSP provides the historical data you need to better make and meet commitments and it makes the routine elements of your job more predictable and more efficient. Then, PSP is the very useful tool to train the students in Programming Language Course. This paper is to provide the concept of PSP/TSP and the sample curriculums for Program Language Training Courses.

Personal Software Process(PSP) to train students of Programming Language Courses

논문접수 : 2003. 12. 12.
심사완료 : 2003. 12. 18.

1) 정회원 : 명지전문대학 정보통신과 조교수

1. 서 론

현대인의 일상생활은 컴퓨터 소프트웨어를 생각하지 않고는 존재할 수 없게 되었다. 전자우편을 확인하는 것으로 하루의 일과를 시작하여, 회사 내의 각종 정보시스템에 접속하여 업무를 처리하는 외에도 인터넷 뱅킹, 인터넷 쇼핑 등의 정보기술들이 생활의 중요한 부분을 차지하게 되었다. 뿐만 아니라 대부분의 현대식 공장들이 소프트웨어에 의해서 가동되고 있으며, 대규모의 금융거래가 소프트웨어에 의해서 자동으로 처리되는 등, 현대 인류가 사용하는 대부분의 제품과 서비스에서 소프트웨어의 중요성이 점점 더 증가되어 가고 있는 실정이다. 이러한 소프트웨어의 개발을 전문으로 하는 소프트웨어 엔지니어의 임무는 약속한 비용과 일정 내에서 우수한 품질의 소프트웨어 제품을 개발하는 것이다. 그러나, 약속한 비용과 일정 이내에 소프트웨어의 개발을 완료한 경우는 실제로 많지 않았다. 일례로 Standish 컨설팅 그룹의 2000년도 조사에 따르면, 기업 소프트웨어 개발 프로젝트의 23%가 실패하고, 단지 28%만이 원래 계획된 예산과 기한 내에 약속된 기능을 제공하였다고 한다[1].

1970년대의 소프트웨어 위기 이후, 소프트웨어 개발 프로젝트의 실패를 방지하기 위한 많은 노력들이 소프트웨어 공학 분야에 있었음에도 불구하고, 여전히 소프트웨어 개발 프로젝트의 실패와 관련된 높은 수치는 소프트웨어 개발 프로젝트실패의 주된 원인이 기술적인 문제뿐만 아니라 관리적인 문제에도 기인하기 때문이다. Carnegie Mellon University의 Software Engineering Institute (SEI)는 소프트웨어 개발 프로젝트의 관리적인 문제에 착안하여 소프트웨어의 품질을 높일 수 있는 일련의 소프트웨어 개발 프로세스로 Capability Maturity Model(CMM), Personal Software Process(PSP), Team Software Process(TSP)를 제안하고 있다.

미 공군이 소프트웨어 공급계약자를 평가하기 위해서 제안된 CMM은 오늘날 민간부문으로 확산되어 조직차원의 프로세스를 심사하여, 해당조직의 소프트웨어 프로세스 성숙도를 초기수준(initial), 반복수준(repeatable), 정의수준(defined), 관리수준(managed), 최적화수준(optimizing)의 5단계로 구분하여 필요한 프로세스 개선 모형을 제시해 주고 있다. CMM의 도입은 소프트웨어 개발 프로세스의 개선에 많은 기여가 있었지만 다음과 같은 문제점이 지적되었다. 첫째, CMM은 소프트웨어 공급계약자를 평가하기 위해서 제안되었으므로 각 조직들이 해야할 일(what they should do)만을 정의하고, 일하는 방식(how they should do)은 정의하지 않았다는 것이다. 둘째, CMM은 관리자를 위한 지침으로 실제 소프트웨어의 개발을 맡고 있는 소프트웨어 엔지니어들에게는 직접적인 지침을 제공하지 않았다는 것이다. 이들 두 문제점들을 해결하기 위해서 SEI는 개별 소프트웨어 엔지니어들이 우수한 소프트웨어를 개발하기 위해서 가져야할 소프트웨어 프로세스가 무엇인가를 연구하게 되었고, 그 결과가 바로 Personal Software Process이다. PSP의 도입으로 개별 소프트웨어 엔지니어는 소프트웨어 개발에 있어서 높은 생산성을 얻을 수가 있었다.

그러나, 소프트웨어의 규모가 커지고, 복잡해짐에 따라서 소프트웨어의 개발이 한 개인에 의해서가 아니라 다수의 소프트웨어 엔지니어들로 구성된 팀에 의한 개발이 보편화되고 있는 실정이다. 이러한 팀 개발환경에서는 PSP를 마친 개별 소프트웨어 엔지니어들이 PSP의 효과를 지속적으로 누릴 수가 없었고, PSP 훈련을 받기 전의 관행으로 퇴보하는 현상이 나타났으므로 SEI는 소프트웨어 개발 팀이 가져야할 소프트웨어 개발 프로세스를 추가로 정의하게 되었고, 이것이 바로 Team Software Process이다.

본 논문에서는 이러한 PSP와 TSP에 대한 개념을 소개하고 이를 프로그래밍 언어 교육 과정에 있는 학생들에게 프로그램 개발을 위한 바람직한 습관을 훈련시키기 위한 도구로 사용하기 위한 교육 과정을 소개하고자 한다. 이를 위해 본 논문의 2장에서는 소프트웨어의 품질 개선 프로세스의 개념과 필요성에 대해서 설명하며, 3장에서는 PSP의 개요로써 PSP를 위한 기본 보고서 양식과 작성 방법에 대해서 설명한다. 4장에서는 팀 단위의 소프트웨어 개발을 위한 프로세tm인 Team Software Process(TSP)에 대해 소개하며, 5장에서는 PSP와 TSP의 효과를 살펴본다. 6장에서는 프로그램 언어 과정에서 PSP를 교육한 사례

를 소개하며, 7장에서 결론으로 논문이 구성되어 있다.

2. 소프트웨어의 품질개선 프로세스

2.1 소프트웨어의 품질

소프트웨어 산업의 시장이 국제화됨에 따라서, 소프트웨어 개발속도가 점점 더 중요해 지고 있다. 경쟁사 보다 우수한 품질의 소프트웨어를 경쟁사 보다 먼저 시장에 출시할 수 있을 때, 해당 기업은 경쟁적 우위를 유지할 수 있다. 그러나, 소프트웨어 엔지니어의 작업이 경쟁적이지 못하면, 우수한 품질의 소프트웨어를 개발할 수 없게되고, 이것은 곧 기업의 지속적 성장을 위해서 필요한 신기술 개발에 투자해야할 기업의 중요한 자원인 시간과 예산을 소프트웨어의 결함을 수정하는데 추가로 사용함으로써 기업의 도산으로까지 이어질 수 있는 심각한 영향을 낳기도 한다.

일례로 dBASE III+로 유명한 Ashton Tate는 1987년에 매출액 2억 1500만 달러에 달하는 굴지의 소프트웨어 회사였다 (참고로, Lotus는 2억 8300만 달러, Microsoft는 2억 6000만 달러). PC용 데이터베이스 시장은 Clipper와 Foxbase 등의 신제품의 출시로 경쟁이 심화되었으며. Ashton Tate는 이러한 후발 기업들의 도전 [illegible] 위해서 dBASE III+의 기능을 대폭 강화한 Dbase IV를 계획하게 되었다. 1988년 2월에 Dbase IV를 5월까지 발표하겠다고 약속했지만, 5월이 되어서 제품의 출시를 2달 연기한다고 발표했으며, 8월이 되자 다시 제품의 발표를 2달 연기하였다. 9월 말에 다시 제품의 출시를 10월 말까지 연기하였으며, 일단 출시하고 결함을 나중에 고치자 (Ship it, we'll fix it later)는 경영진의 결정에 따라 1988년 11월에 결함을 완전히 수정하지 못한 Dbase IV를 공급하기 시작했다. 그러나, Dbase IV의 사용자들이 증가함에 따라서, Dbase IV에 대한 결함의 보고는 기하급수적으로 증대하였으며, Ashton Tate의 엔지니어들이 테스트와 결함의 수정을 계속하였지만, 1991년 2월까지도 베타 테스트가 끝나지 않아서 분기당 560만 달러의 손실을 기록하고 결국 Borland사에게 매각되고 말았다[2].

일반적으로 소프트웨어의 결함을 수정하는데 필요한 비용은 소프트웨어 개발단계의 후반부로 갈수록 증대된다. 따라서 결함이 없는 소프트웨어의 개발을 위해서는 계발계획, 요구분석, 기본설계, 상세설계, 구축, 테스트, 설치 및 인도의 소프트웨어 개발의 각 단계의 초기단계에서부터 소프트웨어의 품질에 대한 세심한 주의가 필요하다. 뿐만 아니라 소프트웨어의 품질을 측정하고 관리하지 않는다면 소프트웨어의 품질확보는 사실상 불가능하다. Personal Software Process와 Team Software Process는 소프트웨어의 품질을 관리할 수 있는 체계적이고도 효과적인 훈련을 소프트웨어 엔지니어에게 제공함으로써 소프트웨어의 개발을 직접 맡고 있는 당사자들이 결함이 적은 소프트웨어를 개발할 수 있도록 지원하고 있다.

결함이 적은 소프트웨어를 개발하기 위해서 Personal Software Process는 코드검토(code review)의 중요성을 강조하고 있다. 소프트웨어의 결함을 제거하기 위한 3가지 전략을 이용하여 코드검토가 얼마나 효과적인지를 살펴보자(그림 1 참고). 첫 번째 전략은 테스트만을 실시하는 것이고, 두 번째 전략은 검사(inspection)와 테스트를 실시하는 것이고, 세 번째 전략은 검사 전에 코드검토를 실시하여 코드검토, 검사, 테스트를 순차적으로 시행하는 것이다. 일반적으로 소프트웨어의 결함은 초기단계에 찾는 것이 쉬우므로 각 [illegible] 결함제거율(yield)을 코드검토에서 70%, 검사에서 60%, 단위테스트에서 50%, 통합테스트에서 40%, 시스템 테스트에서 30%로 가정하자. 그리고, 소프트웨어 개발의 초기단계에서 발견된 결함은 개발단계의 후반부에서 발견된 결함보다 수정하기가 상대적으로 쉬우므로 각 단계에서 발견된 결함을 수정하는데 소요되는 시간을 코드검토에서는 0.9분, 검사에서는 15분, 단위테스트에서는 16분, 통합테스트에서는 120분, 시스템 테스트에서는 414분으로 가정하자. 테스트에 시간이 많이 소요되는 이유는 테스트를 통해서 오류가 확인될 경우, 오류가 난 부분을 조사하여 그 원인을 찾아서 제거하고, 다시 컴파일과 테스트의 과정을 거쳐야하기 때문이다[2].

<그림 1>에서는 어떤 프로그램의 결함이 10,000개라면, 문법오류를 포함하는 간단한 5,000개의 결함은 컴파일과정을 통해서 제거되었다. 코드검토를 통해서 해당 소프트웨어가 가지고 있는 전체 결함의 70%를 제거할 수 있으며, 코드검토

를 통해서 발견된 하나의 결함을 수정하는데

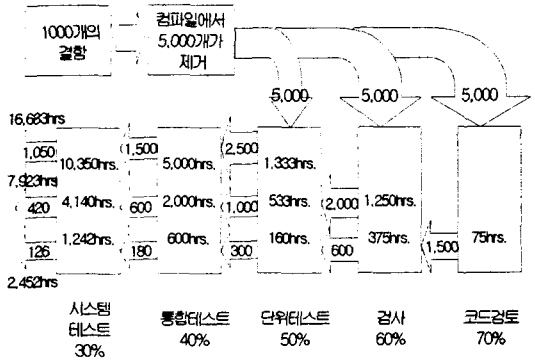


그림 1 결함제거전략

그림 1 결함제거전략

소요되는 시간은 0.9분이므로, 테스트만을 통해서 결함을 발견하고 제거할 경우는 총 16,683시간이 소요되는 반면에 코드검토/검사/테스트를 모두 할 경우는 2,452시간이 소요됨을 알 수 있다. 뿐만 아니라 사용자에게 최종 전달되는 소프트웨어의 품질에 있어서도 테스트만을 거친 경우는 1,050개의 결함이 내재되어 있는 반면에 코드검토/검사/테스트를 모두 거친 경우는 126개의 잠재적 결함을 제외한 모든 결함이 제거되었다. 따라서 소프트웨어의 결함을 찾기 위해서 Personal Software Process를 통해 코드검토를 위한 특별한 훈련을 함으로 해서 소프트웨어 엔지니어는 자신이 개발한 소프트웨어의 결함을 효과적으로 발견하고 제거하여 소프트웨어의 품질을 높이고, 소프트웨어 개발의 생산성을 높일 수 있다.

2.2 품질개선 프로세스

소프트웨어 엔지니어들이 소프트웨어의 품질을 개선시키기 위해서는 품질개선 프로세스를 따라야 한다. Personal Software Process와 Team Software Process를 제안한 Humphrey는 사격훈련의 일화를 들어서 개선프로세스를 설명하고 있다[3]. 그는 미 해군에 근무할 때, 기관총 사격을 배워야 했는데, 모의표적을 이용하여 연습을 하였다. 사격점수는 엉망이었고, 여러 번의 사격연습에도 불구하고 사격성적은 좀처럼 향상되지 않았다. 사격연습을 지켜보던 교관은 왼손으로 사격을 해보라고 제안했으며, 오른손잡이이던 그는 처음에는 왼손 사격이 부자연스러웠지만, 몇 번의 사격연습 후에는 왼손사격으로 여러 번 만족할 만한 사격성적을 얻을 수 있었다.

사격일화는 여러 가지를 시사하고 있다. 첫째, 문제를 진단하기 위해서는 측정이 필요하다. 얼마나 많은 표적을 명중시켰고, 얼마나 많은 표적을 놓쳤는지를 알고 있었기에 교관은 뭔가 다른 것을 해야 한다는 것을 알 수 있었다. 둘째, 측정자료들을 이용한 객관적인 분석이 필요하다. 사격시에 수행했던 장전, 자세잡기, 목표물 추적, 조준, 격발 등의 일련의 과정을 지켜보면서, 교관은 그가 사격에서 사용하였던 프로세스를 분석하였다. 셋째, 개선을 위해서는 변화가 필요하다. 교관의 목표는 어떤 단계들이 문제의 원인인지를 밝히는 것으로, 교관은 조준에 문제가 있다고 분석했으며, 사격 자세를 왼손사격으로 변경할 것을 제안했다. 끝으로 가장 중요한 것은 변화 그 자체이다. 사람들은 누구나 현재의 습관이 자연스러우며, 새로운 것을 시도하는 것을 꺼려하기 때문에 프로세스의 개선은 말처럼 쉽지 않다. 오른손잡이에게 왼손으로 사격을 하라는 교관의 제안된 변화를 받아들인 후에 사격점수는 몰라 볼 정도로 향상되었다.

소프트웨어 엔지니어들이 원래 계획된 예산과 기한 내에 약속한 기능을 제공하는 소프트웨어를 개발할 수 없다면, 지금까지 해 오던 똑 같은 방식으로 해서는, 옛날과 같은 결과들을 얻을 수 밖에 없다. 그러므로, 소프트웨어 개발에 있어서 개선이 필요한 프로세스가 무엇인지를 파악하고, 현재 수행하고 있는 프로세스 방법을 변경하여야 비로소 원하는 개선을 얻을 수 있다. Personal Software Process는 (1) 소프트웨어 개발 프로세스를 개선하기 위해서 품질의 목표를 정의하고, (2) 개발된 제품의 품질을 측정하여, (3) 사용된 프로세스를 이해하고, (4) 사용한 프로세스 중에서 변경이 필요한 프로세스를 조절하고, (5) 조절된 프로세스를 사용한 후, (6) 그 결과를 측정하며, (7) 결과를 목표와 비교하여, (8) 추가로 필요한 프로세스를 조절하는 일련의 품질개선 프로세스의 전 과정에 대한 체계적 훈련을 강조하고 있다.

2.3 훈련의 필요성

Personal Software Process와 Team Software

Process는 소프트웨어 엔지니어들이 소프트웨어 엔지니어로써 필요한 기술을 개발하고 향상시키기 위한 체계적 훈련을 제공한다. 소프트웨어 공학에서 프로세스에 대한 훈련이 필요한 이유로는 첫째, 다른 분야의 전문가들은 정식 교육을 통해서 필요한 직업기술 및 방법을 익히고 있기 때문이다. 화학자들은 실험장비를 다루는 법과 정확한 분석방법을 실험에 앞서 미리 훈련하며, 의사들도 환자의 수술을 직접 맡기 전에 수술절차를 훈련하는 오랜 기간의 수련을 거치게 된다. 즉, 다른 분야의 전문가들은 해당분야의 기본적인 능력을 훈련을 통해서 습득했다는 것을 증명할 수 있을 때, 비로소 해당분야의 가장 간단한 업무나마 수행할 수 있도록 허락된다. 이에 반해, 소프트웨어 분야의 경우는 제대로 훈련받지 않은 소프트웨어 엔지니어들이 필요한 기술들을 실제 업무를 수행하면서 훈련하고 있는 실정이다.

둘째, 소프트웨어 엔지니어들은 지금까지의 개발경험을 통해서 체득한 자기 나름대로의 소프트웨어 개발 프로세스를 가지고 있기 때문이다. 소프트웨어의 개발은 지적인 활동으로 소프트웨어 엔지니어들에게 특정한 방식으로 일할 것을 지시하는 것은 실제로 무의미하다. 대부분의 소프트웨어 엔지니어들은 프로그램을 처음 배울 때 사용했던 방식을 지금까지 사용하고 있다. 뿐만아니라 대학교의 소프트웨어 공학관련 교육은 기술적인 주제에 치중하며, 소프트웨어의 개발에 필요한 기간과 예산을 추정하여 개발 계획을 수립하거나, 소프트웨어의 품질을 측정하고 관리하는 방법에 대한 교육은 미비한 실정이다[4].

따라서, 소프트웨어 엔지니어들이 원래 계획한 예산과 기한내에 약속한 기능을 제공하는 소프트웨어를 개발할 수 있도록 지원하기 위해서는 소프트웨어 엔지니어들의 지금까지의 잘못된 관행을 찾아서 이를 해결하기 위한 품질개선 프로세스를 Personal Software Process와 Team Software Process를 통해서 훈련할 수 있도록 하여야 한다.

3. Personal Process

3.1 Personal Software Process의 개요

PSP의 기본목표는 CMM의 다섯 번째 단계인 최적화수준의 모든 원칙을 개별 소프트웨어 엔지니어들이 훈련하여 자신의 개발 프로세스에 적용할 수 있도록 지원하는 것이다. 즉, PSP 훈련을 마친 소프트웨어 엔지니어들은 각 프로젝트에 대한 자신의 계획을 수립하고, 자신의 프로세스를 추적하며, 필요한 프로세스를 새롭게 정의할 뿐만 아니라, 자신의 업무를 측정하고, 자기가 개발한 소프트웨어의 품질을 측정하고 관리할 수 있어야 한다.

PSP 훈련은 소프트웨어 엔지니어가 우수한 작업을 할 수 있도록 도와주기 위해서 고안되었으며, 10개의 프로그램 과제로 구성되어 있다.

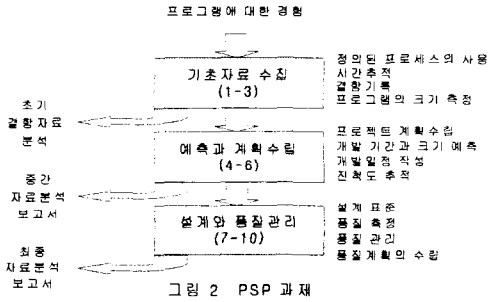
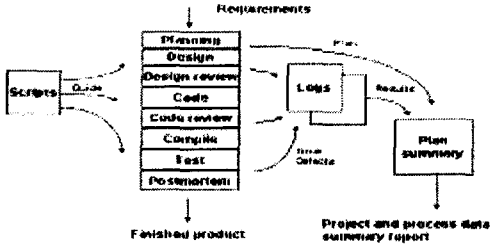


그림 2 PSP 과제

<그림 2> PSP 과제

PSP 훈련의 전반부에서는 3개의 프로그램 과제를 통해서, 개선활동에 필요한 기초 자료를 수집하고 결함자료를 분석하기 위해, 정의된 프로세스를 사용하는 방법과 시간을 추적하고, 결함을 기록하는 방법, 프로그램의 크기를 측정하는 방법을 훈련한다. 중반부에서는 3개의 프로그램 과제를 통해서 예측과 계획수립을 위한 중간 자료분석 보고서를 작성하기 위해서 프로젝트의 계획을 수립하고, 개발기간과 프로그램의 크기를 예측하며, 예측된 개발기간과 프로그램 크기를 이용하여 개발일정을 작성하는 방법과, 개발계획과 실제 개발 진행상황을 비교하여 달성도(earned-value)를 추적하는 방법을 훈련하게 된다. 끝으로 정리부에서는 4개의 프로그램 과제를 통해서 설계와 품질관리와 관련된 최종 자료분석 보고서를 작성하는데 필요한 설계 표준의 작성법, 소프트웨어 품질의 측정과 관리방법, 소프트웨어 품질관리를 위한 계획수립 방법을 훈련한다.



<그림 3> PSP0 프로세스

PSP에서는 훈련의 효과를 극대화하기 위해서 다양한 스크립트(script), 양식(form)과 양식의 작성법을 설명한 지시문(instruction), 각종 표준(standard)과 템플릿(template)등을 제공한다. 스크립트는 각 프로세스의 목적, 시작조건, 프로세스의 세부단계별 활동, 종료조건 등을 정리한 것으로 PSP0 프로세스의 경우, PSP0 Process Script, PSP0 Planning Script, PSP0 Development Script, PSP0 Postmortem Script가 사용되며, 주요활동에 사용한 시간을 기록하기 위한 시간기록일지(Time Recording Log)의 양식과 작성법, 개발과정에서 발견하고 수정한 결함을 기록하기 위한 결함기록일지(Defect Recording Log)의 양식과 작성법, 결함유형에 대한 표준(Defect Type Standard) 등이 사용된다.

대학원 과정의 학생을 위한 PSP 전체 과정[5]은 10개의 프로그래밍 과제와 함께 5개의 분석과제를 수행함으로써 PSP를 실습하며, 학부생을 위한 PSP 과정[3]은 프로그래밍 과목의 일부로서 PSP 훈련을 위한 기초적인 개념과 양식의 사용법을 학습한다. 일반 소프트웨어 엔지니어들을 위해서 SEI에서 제공하는 PSP 훈련은 계획수립을 위한 PSP I과 품질관리를 위한 PSP II로 구성되어 있으며, PSP I과 PSP II는 각각 5일간 60시간의 교육으로 구성되어 있다[9]. PSP는 지금까지 여러 대학교와 산업체에 소개되었으며, PSP과목을 수강한 소프트웨어 엔지니어로부터 얻은 많은 자료를 분석해 보면, PSP가 엔지니어의 계획 능력을 향상시키고, 그들이 생산한 소프트웨어 제품의 품질을 향상 시키는데 효과적이었음을 보여주고 있다. PSP의 기본개념인 시간관리와 결함관리에 대해서는 3.2에서 다시 정리하였다.

3.2 시간관리와 결함관리

약속한 비용과 일정이내에 소프트웨어를 개발하기 위해서는 정확한 계획의 수립이 선행되어야 한다. PSP에서는 계획을 보다 정확하게 수립하기 위해서, 정확한 시간관리를 강조하고 있다. 시간사용방식에는 예외 사항이 있을 수 있지만, 일반적으로 지난 주의 시간사용방식을 안다면, 다음 일 주일간의 시간사용방식을 보다 정확하게 예측할 수 있다. 한편, 사람들은 일반적으로 빠르게 진행된 일들을 좋아하기 때문에 빠른 일들에 사용된 시간들은 실제보다 축소하여 기억하는 경향이 있으며, 반면에 느리게 진행되는 일, 지루한 일 혹은 힘든 일들은 실제보다 시간이 더 오래 걸렸다고 생각하는 경향이 있다. 그러므로, 어떤 일에 어느 정도의 시간을 실제 사용했는지를 알기 위해서는 시간사용에 대한 정확한 기록을 유지하여, 평소에 우리가 시간을 어떻게 보내고 있는지를 추적해야 한다. 시간사용에 대한 기록을 분석하여 특정 활동에 필요한 시간을 예측하고, 이를 바탕으로 계획을 수립하여 문서화하고 나면, 실제 시간 사용의 결과와 원래 계획을 비교하여, 계획 중 잘못된 부분이 무엇인지, 어떻게 계획과정을 개선할 수 있는지를 검토하여야 한다.

시간기록을 위해서 PSP는 시간기록일지(Time Recording Log)를 사용한다. 시간기록일지의 작성을 위해서는 주요활동을 몇가지 범주로 분류한 후에, 주요활동에 실제 사용한 시간을 분단위로 기록한다. 그림 4에서는 주요활동을 계획(plan), 설계(design), 프로그램 작성(code), 컴파일, 테스트, 사후검토(postmortem)로 구분하였으며, 계획은 7시 58분에 시작하여 8시 45분에 마쳤으며, 그 동안에 전화를 받는데 3분을 소요하였으므로 실제 계획에 사용한 시간(delta time)은 44분이 소요되었음을 알 수 있다. 이렇게 작성된 매일의 시간기록일지는 주단위로 요약되어 주간활동기록(Weekly Activity Record)에 기록되어 프로젝트 계획 수립을 위한 기초자료로 활용된다. 즉, 각 주요활동에 사용한 과거 자료가 있을 경우, 소프트웨어 엔지니어들은 이들 자료를 활용하여 새로운 소프트웨어 개발 계획을 수립할 경우, 보다 정확하고도 실행가능성이 높은 계획을 수립할 수 있게 된다.

소프트웨어 개발 프로젝트에 있어서 약속한 비용과 일정의 준수와 함께 강조되어야 할 것은 우수한 품질의 소프트웨어를 개발하는 것이다. 개발된 소프트웨어의 결함을 최소화하기 위해서 PSP는 소프트웨어의 결함을 관리하는 훈련을 제시하고 있다. 소프트웨어 엔지니어들은 각자의 능력과 특성에 있어서 서로 다른 개발 경험을 가지고 있으므로, 자신이 흔히 하는 실수가 무엇인지를 파악하여 이들 문제점들을 해결하기 위한 개선활동에 중점적인 노력을 기울일 필요가 있다.

PSP에서는 결함기록일지(Defect Recording Log)를 이용하여 결함을 발견한 날짜, 결함의 일련번호, 결함의 유형, 결함이 발생한 개발 프로세스, 결함을 수정한 개발 프로세스, 결함의 수정에 소요된 시간,

Time Recording Log

Date	Start	Stop	Interruption Time	Delta Time	Phase	Comments
5/13	7:58	8:45	3	44	Plan	phone call
	8:47	10:29	2	100	Design	create and review design
	7:49	8:59		70	Code	coded main and all functions
	9:15	9:46		31	Compile	compiled and linked
	9:47	10:10		23	Test	ran tests A, B, and C
	4:33	4:51		18	Postmortem	

그림 4 시간기록일지의 예

<그림 4> 시간기록일지의 예

Defect Recording Log

Date	Number	Type	Inject	Remove	Fix Time	Fix Defect
5/13	1	20	CODE	COMPL	22	
Description:	syntax error in scanf statement					
Date	Number	Type	Inject	Remove	Fix Time	Fix Defect
5/13	2	30	CODE	COMPL	16	
Description:	error in linked list struct type declarations within access functions					
Date	Number	Type	Inject	Remove	Fix Time	Fix Defect
5/13	3-6	20	CODE	COMPL	1	
Description:	missing ;					
Date	Number	Type	Inject	Remove	Fix Time	Fix Defect
5/13	7	30	CODE	COMPL	1	
Description:	incorrect spelling of identifier in declaration					

그림 5 결함기록일지의 예

<그림 5> 결함기록일지의 예

Type Number	Type Name	Description
10	Documentation	comments, messages
20	Syntax	spelling, punctuation, typos, instruction formats
30	Build, Package	change management, library, version control
40	Assignment	declaration, duplicate names, scope, limits
50	Interface	procedure calls and references, I/O, user formats
60	Checking	error messages, inadequate checks
70	Data	structure, content
80	Function	logic, pointers, loops, recursion, computation, function defects
90	System	configuration, timing, memory
100	Environment	design, compile, test, or other support system problems

그림 6 결함유형에 대한 표준

<그림 6> 결함유형에 대한 표준

결함의 수정과정에서 추가로 발생한 결함의 여부 등을 기록하여 관리한다. 그림 5에서 발견된 첫 번째 결함은 프로그램의 작성단계에서 발생하였으며, 컴파일 과정에서 발견되어 수정하는데 총 22분이 소요되었음을 알 수 있다. 그림 5에서 발생한 결함은 모두 같은 결함유형이며, 그림 6에서 결함유형 20은 문장오류를 나타내므로, 결함을 유형별로 분류하면 개별 소프트웨어 엔지니어들은 자신의 문제점을 스스로 파악할 수 있게 된다.

PSP 훈련을 통해서 소프트웨어 엔지니어들은 정의된 프로세스를 사용하고, 소프트웨어 개발계획의 수립에 필요한 자료들을 수집하여, 개인의 과거 자료에 근거하여 개별 소프트웨어 엔지니어들이 지킬 수 있는 개발계획을 수립하는 방법을 연습하게 된다. 뿐만 아니라 설계표준의 중요성과 코드검토를 통한 결함의 발견과 수정의 중요성을 PSP 훈련의 10개 프로그램 과제를 통해서 확인할 수 있으므로 소프트웨어 개발 초기단계에서부터 소프트웨어 품질을 측정하고 관리하여 우수한 품질의 소프트웨어를 개발할 수 있게 된다.

4. Team Software Process

4.1 TSP의 개요

소프트웨어의 규모가 커짐에 따라서, 상업용 소프트웨어의 개발이 한 개인의 소프트웨어 엔지니어에 의해서 주도되는 경우는 흔하지 않으며, 대부분의 소프트웨어들이 일단의 소프트웨어 엔지니어들로 구성된 팀에 의해서 개발되고 있는 실정이다. 일반적으로 팀은 각 개인의 능력을 합한 것보다 더 큰 능력을 발휘할 수 있는 시너지 효과를 누릴 수도 있지만, 다양한 기술과 인적자원을 바탕으로 팀 활동이 전개되므로 개인적 소프트웨어 개발 프로세스보다 훨씬 고려하여야 할 점이 많게 된다. Team Software Process는 소프트웨어 개발팀에게 동기를 부여하여 효과적인 소프트웨어 개발팀을 구성하고 유지하는 훈련방법을 제공한다.

TSP에서 목표로 하는 개발팀은 (1) 개별 구성원들은 업무지식과 업무수행능력을 가지고 있으며, (2) 모든 팀 구성원들이 협동하여 달성하고자 하는 팀 목표를 가지고 있으며, (3) 팀 구성원들이

모두 팀 목표가 달성 가능하다고 믿고 팀 목표의 달성을 위한 자신의 역할을 정의하고 있으며, (4) 팀 구성원의 개별 업무수행과 진도점검을 위한 공통된 프로세스와 계획을 가지고 있으며, (5) 팀 리더는 팀 활동을 지원하고 보호하고 동시에 팀 구성원들에게 팀활동의 진행상태를 알려주는 팀이다.

이러한 이상적인 팀을 구성하기 위한 선결조건으로 TSP는 개별 소프트웨어 엔지니어들이 PSP를 통해서 계획수립, 측정, 그리고 추적에 대한 기술과 함께 프로세스, 품질, 그리고 헌신의 중요성을 익히도록 제안하고 있다. 일단 팀 구성원들의 준비가 끝나면, 착수식(launch)을 하게된다. 착수식은 일반적으로 4일간 진행되며 (그림 7 참고), 착수식 동안 팀은 팀 목표를 설정하고, 팀 구성원간의 역할을 할당하며, 업무수행을 위한 계획을 설정하게 된다. 착수식이 끝나면, 팀은 개발작업에 들어가고, 팀은 주별 회

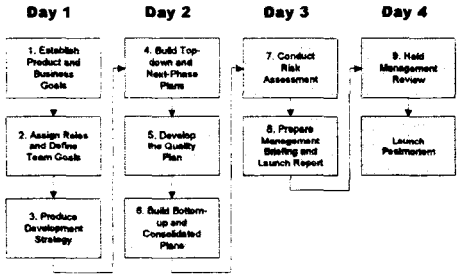


그림 7 TSP 착수식 일정

<그림 7> TSP 착수식 일정

의를 통해서 주단위로 개별 소프트웨어 엔지니어의 개발진행 상황과 팀의 개발진행 상황을 점검하고 필요한 조치를 취하게 된다. 팀 리더는 정기적으로 팀의 진행상황을 상급자에게 보고하고, 외부로부터의 불필요한 간섭을 배제하여 팀 구성원들이 계획한 목표를 헌신적으로 달성할 수 있도록 동기를 부여하고 보호한다. 일반적으로 팀 작업이 5~6개월이 지나면 소프트웨어에 대한 요구사항의 변경, 팀 구성원의 변경, 팀 구성원의 학습효과에 의한 생산성 향상 등을 반영할 수 있도록 착수식을 다시 하여야 한다.

TSP에서 팀 착수식은 성공적인 팀을 구성하기 위한 가장 중요한 활동으로 특별히 강조되고 있다. 소프트웨어의 개발이 계획한 예산과 기한 내에 약속한 기능을 제공하기 위해서는 예산과 기한과 기능에 대한 계획이 관리자에 의해서가 아니라, 개발을 담당하는 소프트웨어 엔지니어들로 구성된 개발 팀에 의해서 수립되어야 한다는 것이 TSP의 기본 철학이다. 관리자가 다른 조직과의 계약을 위해서 만든 예산, 기한, 기능에 대한 계획이 개발 팀과의 약속으로 구체화될 수 없다면, 관리자의 계획은 개발 팀의 헌신을 전제로 개발 팀이 수립한 예산, 기한, 기능에 대한 계획으로 변경되어야 한다. 그러므로 최고경영자는 팀 착수식에 반드시 참석하여 계획하고 있는 프로젝트의 중요성을 개발 팀에서 인식시키고, 개발 팀은 최선을 다하여 실행 가능한 개발 계획을 수립하고, 최고경영진을 포함한 이해 당사자에게 수립된 계획의 타당성을 설명하고 승인 받을 수 있어야 한다.

한편, TSP 착수식의 구체적인 결과물들은 그림 8과 같다. 팀 목표는 품질에 대한 목표를 기술한 것이고, 개념설계는 소프트웨어 엔지니어들이 개발하고자 하는 제품에 대한 개념을 구체화한 것이다. 팀 전략은 팀의 개발전략으로 버전, 프로토타입, 통합계획 등을 포함하여야 하며, 일정계획은 팀의 상세한 주별계획으로 일반적으로 Gantt 차트를 많이 사용한다. 달성도(earned-value) 계획은 작업의 완성도를 점검하는 것으로 팀 활동의 진행상태를 점검할 수 있다. 팀 구성원의 역할은 품질관리, 설계, 계획, 테스트 등 각자 개별 구성원이 맡은 역할을 구체적으로 명시하여야 하고, 작업계획은 각 주요작업에 대한 달성도와 예상완성 날짜 등에 대한 내용을 포함하여야 한다. 끝으로 각 개발단계별로 예상되는 결함의 수준을 품질계획을 통해서 미리 예측하고, 예상할 수 있는 주요 위험과 위험의 영향을 위험평가에서 미리 평가하여, 중요한 문제가 발생할 경우에 대한 대체방안을 기술한 대체계획까지 TSP 착수식을 통해서 수립하여야 한다.

4.2 TSP 훈련과 TSPi

TSP 팀이 성공하기 위해서 SEI는 팀 리더와 팀 구성원을 신중히 선발하여 미리 훈련시켜야 한다고 강조하고 있다. 그림 9는 SEI에서 제공하는 TSP 훈련이다. TSP의 도입을 위해서 SEI는 팀 구성원, 팀 관리자, 중역진과 최고경영자들을

위한 6개 과정를 개설하고 있다. TSP의 기본개념, 계획, 품질, 헌신, 동기부여 등과 관련된 1.5일간의 중역세미나과정을 통해서 중역진과 최고경영진들이 TSP의 중요성을 인식하고, TSP의 추진을 결정하고 나면, 관리자들과 팀 리더는 중역세미나과정과 별도로 3일간의 팀원의 훈련과 팀 관리와 관련된 관리자 과정을 마쳐야 한다.

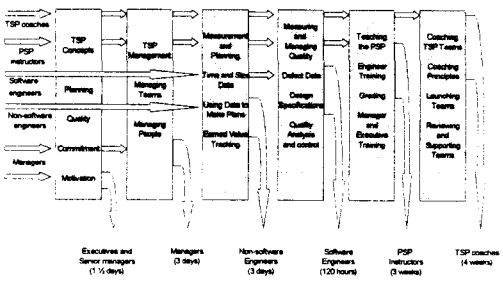


그림 9 TSP 훈련

<그림 8> TSP 훈련

팀원은 일반적으로 중역세미나과정과 관리자 과정을 거치지 않는다. 소프트웨어 엔지니어들은 프로세스를 정의하고, 사용하는 방법, 프로세스와 제품을 측정하는 방법, 자신의 업무를 계획하고 추적하는 방법, 소프트웨어의 품질을 측정하고 관리하는 방법 등을 훈련하여야 하는데, 이는 일반적으로 3.1에 설명한 60시간씩 2주에 걸쳐서 실시되는 Personal Software Process 훈련을 통해서 실시된다. 일반 팀 구성원들은 실제로 소프트웨어를 개발하지는 않지만, TSP 착수식과 TSP와 관련된 다양한 팀 활동에 참여하여야 하므로 측정과 계획에 대한 교육을 받아야 한다. 3일간에 걸친 교육을 통해서 일반 팀 구성원들은 프로그래밍 활동이 포함된 소프트웨어 품질의 측정과 관리를 위한 교육을 받게되므로 프로그램에 대한 이해가 있어야 한다. PSP와 TSP를 조직내에 확산하기 위해서는 PSP 강사와 TSP 코치를 위한 별도의 과정을 추가로 마쳐야 한다.

중역진의 적극적인 지원이 없이는 소프트웨어 엔지니어들이 현재 수행 중인 개발 프로젝트에서 PSP 훈련을 위해서 필요한 120시간을 할애하는 것이 거의 불가능하다. 그러나 PSP 훈련이 없이는 소프트웨어 엔지니어들이 자신의 업무에 대한 계획을 수립하고, 진행상황을 추적하며, 소프트웨어의 품질을 측정하고 향상시키는 방법을 알 수 없으므로, TSP를 성공적으로 수행할 수 없게 된다. 그러므로 TSP를 도입하기 위해서는 중역진과 소프트웨어 엔지니어들을 포함한 모든 관련자에 대한 적절한 교육이 선행되어야 한다.

TSP와 관련된 마지막 주제로 TSP와 TSPi를 구분해 보자. Team Software Process는 PSP의 원칙을 팀 소프트웨어 개발환경에 적용한 것으로 3~20명 정도의 구성원을 가진 팀이 수개월에 걸쳐서 진행되는 프로젝트에 적합한 프로세스이다. 반면에 Introductory Team Software Process(TSPi)는 TSP의 기본 개념과 방법을 대학교육에 적합하도록 축소시킨 것으로 3~6명으로 구성된 팀이 한 학기에 마칠 수 있는 규모의 프로젝트에 적합한 프로세스이다. TSPi는 정형화된 중급정도의 난이도를 가진 문제를 학생들에게 제공하고, 그 해결과정의 훈련을 통해서 TSP 과정을 훈련시킨다. TSPi의 교재[6]에서 제공하는 과제는 LOC change counter와 program complexity analyzer이며, 이들 과제를 학생들은 여러 번의 사이클(cycle)을 거쳐서 개발하게 되며, 각 사이클별로 완성된 소프트웨어를 개발하여야 한다. 즉, 첫 번째 사이클에서는 TSP의 7단계인 전략(strategy), 계획(planning), 요구사항(requirement), 설계(design), 구현(implementation), 시험(test), 사후처리(postmortem)에 정의된 프로세스를 학습하는 것을 목표로 해서, 이들 프로세스 과정을 이해하고 이를 적용하여 가장 중요하고도 기본적인 시스템을 완성하고, 두 번째 사이클에서는 첫 번째 사이클에서 학습한 TSP의 7단계 프로세스를 다시 적용하여 첫 번째 사이클에서 개발된 시스템의 기능을 향상시킨다. 그리고 학기말까지의 시간적 여유가 있으면 추가적인 사이클 수행을 통해서 시스템의 완성도를 높여간다.

TSPi에서 각 팀원은 팀 리더, 개발관리자, 계획관리자, 품질/프로세스 관리자, 지원관리자로 구분되는 관리자의 역할과 개발 엔지니어로서의 역할을 동시에 수행하여야 한다. 각 관리자들이 해야할 임무에 대해서는 각종 스크립트를 이용하여 상세히 설명하고 있다. 일례로 팀 리더는 팀 구성원들이 각자 맡은 임무를 수행하도록 격려하며, 팀 구성원간의 갈등이 있을 경우 이를 해결해야

한다. 팀 회의를 주관하고 팀 과제의 진행상황을 담당교수에게 보고하는 것도 팀 리더의 임무이다. 뿐만 아니라 팀 리더는 구성원들에게 팀에 부여된 업무를 배분하여야 하는데 이때는 같은 시간 내에 모든 팀 구성원들이 맡은 임무를 마칠 수 있도록 팀 구성원들의 능력을 고려하여야 한다.

TSPi에 대한 사례로는 Southern Polytechnic State University의 CS4724 Software Engineering Projec[10]t, Embry-Riddle Aeronautical University의 SE300 Software Engineering[11] , University of South Carolina의 CSCE 741 Software Process [12]등을 참고할 수 있다. 이들 TSPi 사례는 TSPi를 통해서 팀의 구성과 관리능력들을 학습할 수 있었으며, 소규모 팀 프로젝트의 경우 TSPi가 매우 효과적이지만, PSP 훈련이 선행되어야 한다고 공통적으로 지적하고 있다.

5. PSP/TSP의 효과

PSP와 TSP의 효과는 (1) 예측력의 향상, (2) 소프트웨어 품질의 향상, (3) 개발기간의 단축, (4) 이직율의 감소로 요약할 수 있다[8]. 첫째, PSP와 TSP의 도입으로 프로그램 개발에 대한 기초자료가 축적되면, 이들 자료들을 이용하여 개발기간과 프로그램의 크기를 예측할 수 있으며, 예측된 개발기간과 프로그램의 크기를 이용하여 개발계획을 수립하게 되므로 계획이 보다 정확해진다. 그림 10은 client-server, embedded, mainframe, internet/intranet용 신기술을 활용하는 소프트웨어를 주로 개발하고 있는 Advanced Information Services(AIS)의 계획의 정확성을 도식화 한 것이다. AIS는 프로세스 개선을 위한 별도의 노력이 없었던 1986~1992년 사이의 사업의 초기단계에는 일정과 예산의 예측이 불가능하였으며, 소프트웨어의 품질도 측정할 수가 없었다. 이 기간동안 AIS는 일정은 평균 112%, 예산은 평균 85%가 계획에서 이탈하였다. 1992년에 AIS는 시장환경의 변화에 대응하기 위하여 CMM을 도입하여 프로세스 개선을 위한 심사를 실시하고, 활동계획을 작성하는 등 CMM 체계하에서의 훈련을 강화하였다. 그 결과 일정은 평균 41%, 예산은 평균 17%만이 계획에서 이탈하였다. 1996년에 AIS는 보다 정확한 예산과 일정의 예측없이는 시장에서의 경쟁력 확보가 곤란하다고 판단하여 PSP를 도입하였고 그 결과 일정은 오차범위 20%, 예산은 오차범위 25% 내에서의 계획의 수립이 가능하였다.

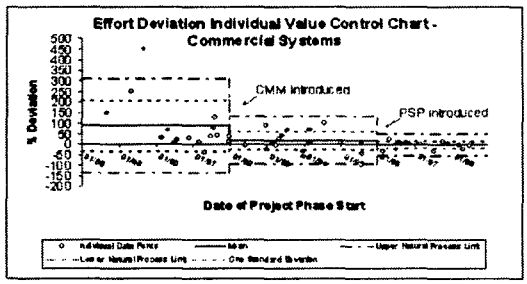


그림 10 예측력의 향상

<그림 9> 예측력의 향상

둘째, PSP와 TSP의 도입으로 소프트웨어 개발 프로세스가 훈련을 통해서 정의된 프로세스에 따라서 관리되면, 소프트웨어의 결함이 감소하여 소프트웨어의 품질이 향상된다. Boeing은 1991년부터 CMM의 도입을 통해서 프로세스의 개선에 주력하여 많은 조직들이 CMM의 최상위 수준인 최적수준에 도달하였다. Boeing의 Pilot 1은 PSP 훈련을 마친 32명을 포함한 35명의 소프트웨어 엔지니어에 의해서 새로운 기능의 추가하는 수정작업 프로젝트로 release 6, 7, 8은 TSP를 사용하지 않았으며, release 9는 TSP를 사용하였다. 그림 11은 TSP의 도입을 통해서 프로그램의 크기는 2.36배 증가하였지만, 소프트웨어의 결함은 75%나 감소하였음을 보여주고 있다. 이러한 소프트웨어 결함의 감소는 PSP 훈련을 통해서 자신의 결점을 잘 알고 있는 소프트웨어 엔지니어들이 결함과 관련된 자신의 문제점을 스스로 해결하고자 더욱 노력하게 되고, TSP 훈련을 통해 품질/프로세스 관리자의 역할에 대해서 각 팀 구성원들이 보다 잘 이해하여 품질관리를 위해서 특별한 노력을 기울이기 때문이다.

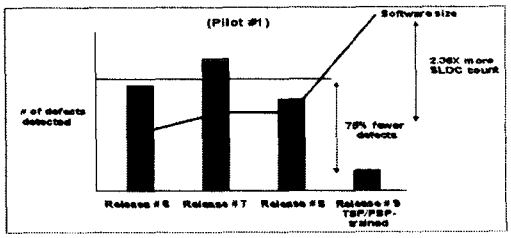


그림 11 소프트웨어 품질의 향상

<그림 10> 소프트웨어 품질의 향상

셋째, PSP와 TSP의 도입에 따른 소프트웨어의 결함의 감소는, 테스트와 결함의 수정에 필요한 시간을 획기적으로 줄여줌으로 궁극적으로는 개발기간의 단축으로 이어진다. 끝으로, 소프트웨어 엔지니어들을 포함하여 모든 사람은 좋은 사람들과 일하기를 좋아한다. 자신이 속한 팀에서 설정한 일정과 예산 계획을 반복적으로 지킬 수 있다면, 소프트웨어 엔지니어들은 남다른 만족과 보람을 느낄 것이고 이는 이직율의 감소로 이어져 우수한 개발 팀의 성과를 지속적으로 누릴 수 있게 해 준다.

6. 프로그래밍 언어 교육 과정에서의 PSP 교육 사례

지금까지 소프트웨어 개발 방식의 개선을 위한 PSP와 TSP에 대해 간략히 살펴 보았다. 이러한 PSP와 TSP가 소프트웨어 개발에서의 생산성 향상에 기여하도록 하기 위해서는 프로그래머 개인의 개발 습관이 구체화되기 이전인 프로그램 교육 단계에서부터 체계적으로 이루어지는 것이 가장 효과적인 교육 방안이다. 조기에 PSP에 대한 교육과 훈련이 실시된다면 개발자들은 시간 기록, 오류 기록과 같이 번거롭다고 생각되는 새로운 프로세스 개선 훈련에 대한 거부감이 없이 자연스럽게 PSP를 통한 개발 방법 개선에 참여할 수 있을 것이다. 따라서, 본 논문에서는 프로그래밍 언어 교육 과정에서 소프트웨어 개발 프로세스의 중요성을 교육생에게 인지시키고 PSP를 훈련시키기 위한 교육 사례를 제시하고자 한다.

근본적으로 PSP에서 가장 강조하는 부분은 각종 프로그램 개발 계획을 수립하는데 있어 개발자 개인의 과거 개발 경험에 근거하여 개발 계획을 수립하는 것이다. 이를 위해서 학생들에게 제시하는 각종 프로그래밍 과제물을 학생들이 제출하기 위해서는 학생들의 개발 결과물과 더불어 PSP0에서 요구하는 각종 보고서를 같이 제출하도록 하는게 무엇보다 중요한 점이다.

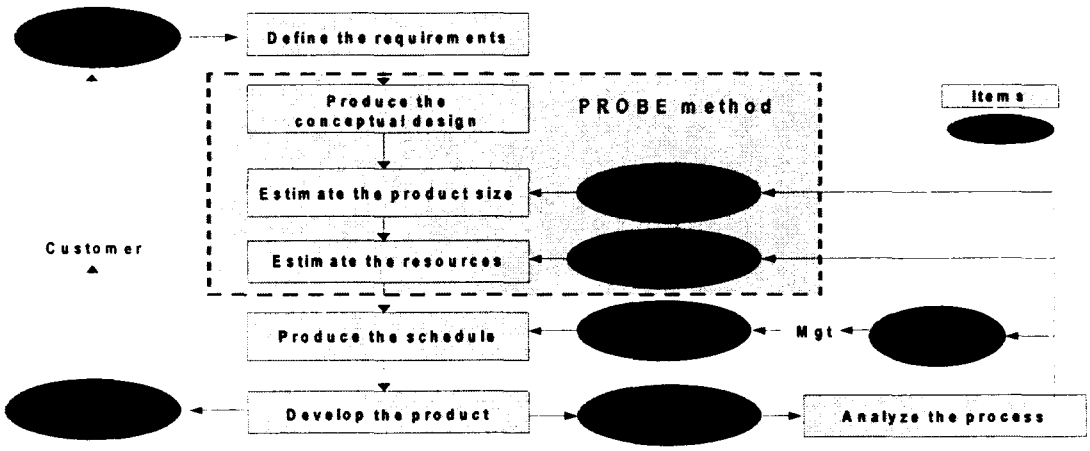
본 본문에서 적용한 교육사례는 이미 C언어의 기초 과정을 습득한 학생들을 대상으로 실시하는 네트워크 프로그래밍 과정을 대상 교육으로 삼았다. 여기에서와 같이 이미 C언어에 대한 기본 과정을 수강한 학생들을 교육 대상으로 선정한 이유는 제한된 시간동안 C언어 초보자에게 C언어에 대한 기본을 강의하기에도 시간이 부족할 뿐만 아니라 프로그램에 대한 기초 지식이 전혀없는 상태에서 기록하는 것은 시행착오로 인해 발생되는 잘못된 결과 데이터들이 PSP 보고서에 반영되는 것을 방지하기 위한 것이다. 즉, 프로그램 언어에 대한 최소한의 기초 지식을 수강한 학생들을 대상으로 PSP 훈련을 실시하였다. 교육 시간은 12주로 구성되어 총 360시간으로 되어 있으며, 이 중에서 PSP 강의는 3일간 24시간으로 구성되었다. <표 1>에서는 강의 내용 및 강의 시기이다. <표 1>에서 보인 바와 같이 PSP를 훈련하는데 있어 강의 시점 역시도 매우 중요한 것으로, 전체 교육 과정이 진행되는 과정에서 적절하지 않은 시점에서 PSP 강의가 진행된다면 학생들의 프로세스 개선 훈련에서 단계별로 실시해야 하는 각종 보고서 및 예측 과정이 일치하지 않을 수 있다. 또한, 강의 진행 과정에서 개발자 자신의 개발 경험 자료를 축적하고 분석할 수 있도록 하기 위해 전체 강의 일정에서 단계별 진행을 위한 적정한 규모의 과제물을 학생들에게 제공해 주어야만 한다.

<표 1> PSP 강의 일자별 내용

강의일 수	강의 내용	강의 시기
1일차	▪ 소프트웨어 개발 프로세스 ▪ PSP0/PSP0.1의 각종 보고서 양식 설명	▪ 전체 강의일정의 도입부분에 실시
2일차	▪ 제출된 PSP0 보고서 문제점 분석 ▪ PSP1/PSP1.1 ▪ PROBE 메소드	▪ PSP0 보고서 3회 이상 수집 후 실시
3일차	▪ PSP1 보고서 문제점 분석 ▪ PSP2/PSP3 설명 ▪ 개인별 프로세스 개선 사례 발표	▪ PSP1 보고서 3회 이상 수집후 실시 ▪ 개선사례 발표는 별도로 실시

<표 1>의 교육 일정에서 제시된 PROxy-Based Estimating(PROBE) 메소드는 PSP에서 각종 프로그램 개발 계획과 일정을 수립하기 위해 요구되는 개발될 프로그램의 크기를 산출하기 위해 제시하고 있는 방법이다. PROBE 메소드의 활용 방안은 <그림 11>에 나타나 있다.

<그림 11>에서 나타낸 바와 같이 PROBE 메소드는 프로그램 개발을 위한 각 단계에서 산출되는 데이터를 활용하여 다음 개발 계획을 수립하는데 재 활용하자는 것으로 PROBE 메소드에서 새로이 개발될 프로그램의 크기를 산출하는 방식은 (식1)에 나타나 있다.



<그림 11> 프로그램 크기 예측을 위한 PROBE Method 적용 방안

$$y_k = \beta_0 + \beta_1 x_k \qquad (식1)$$

$$\beta_1 = \sum_{i=1}^{n} x_i y_i - n x_{avg} y_{avg} \Big/ \sum_{i=1}^{n} x_i^2 - n(x_{avg})^2$$

$$\beta_0 = y_{avg} - \beta_1 x_{avg}$$

(식1)에서 x_k는 개발자 스스로가 설정한 Proxy로부터 산출된 프로그램 크기의 예측 결과이며, y_k는 자신의 과거 경험 결과를 반영한 $_0$ 와 $_1$

을 x_k에 반영한 것으로 이 값이 바로 PSP에서 제시하는 새로운 프로그램의 크기가 되는 것이다. PSP에서 제시하는 PROBE 메소드를 수행하기 위해 요구되는 최소한의 과거 경험 수는 3회 이상의 PSP0 보고서가 존재해야만 한다. (식1)에서 $_0$이 의미하는 것은 개발자가 프로그램을 개발하는데 항상 습관적으로 포함되는 코드를 의미하는 것으로 개발자의 습관을 PSP에서 반영하고 있는 것이다. 따라서, PSP에서 $_0$값은 항상 0 보다 커야 한다. 또한, (식1)에서 $_1$이 의미하는 것은 사용자가 새로운 프로그램을 개발하기 위해 프로그램 크기를 예측하는데 있어, 개인의 직관에 의하여 예측한 프로그램 크기가 개발된 이후의 실제 프로그램 크기와 얼마나 차이가 있었는지를 보여주는 것이다. 예를 들어 $_1$값이 0.25라고 가정한다면 개발된 이후의 실제 프로그램 크기는 개발자가 개발 초기에 예측한 프로그램 크기의 25% 밖에 되지 않는다는 것을 의미한다. 결국, 이 개발자는 프로그램 크기를 예측하는데 있어 항상 너무 크게 프로그램 크기를 예측하는 성향이 있음을 나타낸다. 따라서, PSP에서는 (식1)에서 제시된 $_0$의 값이 0보다 적거나 $_1$의 값이 1에 근사한 값이 아니라 1과 너무 떨어져 있는 값이라면, 개발자의 과거 경험 정보가 잘못된 것이라 판단할 수 있다. 이러한 경우의 PROBE 메소드는 부정확한 값을 예측할 수 밖에 없어 사용하는게 의미가 없어진다. 결국 PROBE 메소드를 통한 프로그램 크기 예측을 위해서는 개발자의 과거 경험 정보를 정확하게 기록하는 것이 매우 중요한 것임을 알 수 있다. 따라서, 수강생들이 제출한 3회 이상의 PSP0 보고서를 근간으로 수강생들 스스로 각자의 $_0$와 $_1$을 산출하도록 하여 이 값의 사용 가능 여부를 판단해 주어야 한다. 이러한 결과를 바탕으로 PSP1을 교육함으로써 과거 경험을 근간한 새로운 프로그램 크기 및 개발 일정 예측 훈련을 실시할 수 있게 된다.

논문에서 실험한 PROBE 메소드의 $_0$는 -10.3에서 13까지, $_1$은 0.3 ~ 0.7까지의 값으로 폭넓게 분포하고 있어, 과거 경험 결과를 기록하는 것이 단시간에 이루어지는 습관이 아니라 아주 오랜 시간동안 지속되는 훈련을 통해 얻어질 수 있는 습관임을 알 수 있었다.

7. 결 론

소프트웨어 개발하는데 있어 계획된 예산과 기한 내에 약속한 기능을 제공하는 소프트웨어를 개발하는 것은 모든 소프트웨어 엔지니어들의 소망일 것이다. 그러나, 소프트웨어 개발과 관련된 성공사례 만큼이나 많은 실패사례들이 소개되고 있으며, 만일 지금까지 방식으로는 이러한 실패를 방지할 수 없다면, 소프트웨어 엔지니어들은 뭔가 다른 방식을 위한 개별적인 노력을 강구해야 한다.

개선을 위해서는 현상을 분석하는 것이 무엇보다 중요하며, Personal Software Process를 통해서 개인 소프트웨어 엔지니어들은 자신들의 소프트웨어 개발 프로세스를 반성하고 개선할 수 있는 기회를 가질 수가 있다. 최근에 발간된 대학의 컴퓨터-소프트웨어 교육 강화방안[4]에 대한 보고서를 보면, "대규모 프로젝트 수행시 꼭 필요한 팀 체제 소프트웨어 개발 교육의 부족"과 "팀원으로서 더불어 일할 수 있는 능력"의 중요성을 지적하고 있다. 이러한 소프트웨어 개발을 위한 팀 활동에 필요한 자질들은 Team Software Process를 통해서 훈련할 수 있을 것이다.

따라서, 향후 우리나라의 소프트웨어 산업 발전과 개발된 소프트웨어의 품질을 향상시키기 위해서는 장기적으로 PSP와 TSP의 교육을 각종 프로그램 언어 교육 과정에서 다루거나, 대학의 정식 교과과정으로 채택하기 위한 표준 교안의 작성이 추진되어야 한다. 표준 교안은 효과적인 PSP와 TSP 교육을 각 교육 내용에 따라 단계적으로 진행할 수 있도록 구성되어야 하며, 또한 개발자가 자신의 경험 자료를 기록하는 과정에서 왜곡되지 않은 정확한 결과를 기록할 수 있도록 하고, 교육 담당자의 역할을 사전에 잘 정의해 주어 충실한 경험 자료를 축적할 수 있도록 하는 것이 무엇보다도 중요하다. 본 논문에서는 C언어를 이용한 네트워크 프로그래밍 과정에서 PSP를 교육한 일례를 통해 PSP의 교육 내용 및 교육 시점을 소개하였고, 여기서 신뢰할 수 있는 예측 정보를 획득하는 것이 얼마나 어려운 일이라는 것을 통해 지속적인 PSP 훈련이 필요하다는 것을 알 수 있었다.

PSP와 TSP의 기본 철학은 신뢰이다. PSP와

TSP는 개별 소프트웨어 엔지니어들의 자발적인 헌신없이는 공허한 것이며, PSP와 TSP를 소프트웨어 엔지니어들의 생산성을 점검하는 수단으로 사용하려 할 경우, 소프트웨어 엔지니어들은 자신의 개발 프로세스에 대한 자료의 수집과 관리에 왜곡을 가할 수 있으며, 이는 더 큰 문제를 초래할 수 있기 때문이다. 대신에 경영자는 모든 소프트웨어 엔지니어들이 자신이 개발한 소프트웨어에 자부심을 느낄 수 있도록 자기 개선을 위한 노력에 스스로 동참할 수 있도록 권장되어야 하며, 이러한 자기개선 노력의 일환으로 PSP와 TSP의 도입이 추진될 때, PSP와 TSP의 효과가 극대화되어서 궁극적으로 경영자들이 원하는 계획된 예산과 기한 내에 약속한 기능을 제공하는 소프트웨어의 개발로 이어지게 된다.

참고 문헌

[1] "CIO Magazine," 2001. 5. 1. p.50

[2] Humphrey, W. S., "Every Business is a Software Business, Draft for Publishing," 2001

[3] Humphrey, W. S., "Introduction to the Personal Software Process, Addison Wesley," 1997

[4] 김진형, "대학의 컴퓨터-소프트웨어 교육강화 방안," 한국소프트웨어진흥원, 2001.11.17.

[5] Humphrey, W. S., "A Discipline for Software Engineering," Addison Wesley, 1995

[6] Humphrey, W.S., "Introduction to the Team Software Process," Addison Wesley, 1999

[7] "PSP/TSPi Summer Faculty Workshop 자료," The University of Queensland, 2001, 12. 3-6

[8] McAndrews, D.R., "The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practice," Technical Report CMU/SEI-2000-TR-015, 2000

[9] http://www.sei.cmu.edu/products/courses/psp/psp.trng.html

[10] http://lovelace.spsu.edu/jdiaz/classes/default.htm

[11] http://faculty.erau.edu/hilburn/se300/se300.html

[12] http://www.cse.sc.edu/~cannon/csce741

[13] CrossTalk: The Journal of Defense Software Engineering, 13(6), PSP/TSP 특집, 2000.

[14] IEEE Software, 17(6), PSP/TSP 특집, 2000