**The Mythical Man-Month**

20220019 안재영

**1장: "The Tar Pit"**

* 대규모 소프트웨어 개발은 마치 "타르 구덩이" 같다. 타르 속에 갇힌 동물들은 어느 쪽발이라도 움직이는 데 별 지장이 없지만, 동시적인 요인들에 의해 움직임은 점차 둔해진다. 대규모 프로그래밍도 마찬가지로, 문제의 까다로움에 모두가 놀라면서도 그 본질의 분별은 어려워진다.
* 프로그래밍의 즐거움 : 무언가를 만드는 데서 오는 순전한 기쁨, 다른 이들에게 쓸모 있는 사물을 만드는 데서 오는 기쁨, 서로 맞물리는 복잡한 법칙들의 조화를 바라보는 기쁨, 지속적인 배움에서 오는 기쁨, 유연하고 다루기 쉬운 표현 수단을 사용한 작업이 주는 기쁨은 프로그래밍의 즐거움을 만드는 요소들이다.
* 프로그래밍의 고달픔 : 반면, 완벽함에 대한 요구, 업무에 대한 자주적 결정력의 부족, 불가피한 타인의 프로그램에의 의존, 사소한 버그 수정이라는 “일”, 오히려 개발 막바지에 더 발견되는 버그들, 완성품이 한물간 것으로 치부되게 만드는 빠른 발전 속도 등은 프로그래밍을 힘들게 만들기도 한다.
* 많은 이들을 허우적거리게 만드는 타르 구덩이임과 동시에 나름의 즐거움과 고달픔을 담고 있는 창조적 활동, 이것이 바로 프로그래밍이다.

**2장: "The Mythical Man-Month" (사람과 시간이 비례하지 않음)**

* 부족한 시간은 소프트웨어 프로젝트를 망가뜨리는 최다 요인이다.
  + 우리의 추정 능력은 형편없다. 모든 일이 잘 될거라는 가정을 하고는 한다.
  + 추정에 있어 투입 공수와 작업 진척도를 혼동해 인원, 기간이 상호 교환 가능하다고 여기고는 한다.
  + 자기가 내린 추정이 스스로도 불확실하기에, 정중한 고집이 부족하다.
  + 일정 진척도가 제대로 모니터링되지 않는다.
  + 일정이 어긋날 경우의 일반적인 대응이 인력 증원으로, 이는 사실 사태를 악화시킬 뿐이다.
* 낙관적인 프로그래머
  + 낙관론자들은 “이번에는 돌아갈 겁니다.”, “방금 마지막 버그를 수정했습니다.”라 하고는 한다.
  + 가장 잘못된 가정은 “모든 작업이 예정 내로 완료될 것”이라는 것이다.
  + 프로그래밍의 재료인 생각은 다른 창작의 목재, 물감처럼 물리적 한계를 주지 않고, 이에 프로그래머들 사이에 낙관주의가 만연하게 된다.
* 맨먼스
  + 프로그래머들은 추정 및 일정 관리에서 투입된 노력을 맨먼스라는 단위로 계산하는 실수를 저지른다.
  + 작업 진척도는 투입된 사람의 수와 달수의 곱에 비례하지 않는다.
  + 작업에 순서가 있어 나누기 힘들다면 쏟아붓는 노력은 도움이 될 수 없다.
  + 분할은 가능하나 하위 작업의 처리에 소통이 필요하다면, 소통을 위한 노력은 사람의 수가 증가할수록 n(n-1)/2배가 되고는 한다.
  + 사람의 투입은 때로는 일정을 단축시키기는커녕 늘어지게 만들 수 있다.
* 시스템 테스트에는 일정에 다음과 같은 일정 분배는 꽤 효율적이다.
  + 계획 수립 1/3 코딩 1/6 구성 요소 및 초기 시스템 1/4 준비 후 1/4
  + 계획 수립에는 많은 시간을 배정하며, 추정하기 쉬운 코딩의 부분에는 비교적 적은 일정을 배정한다.
  + 시스템 테스트는 처음부터 오랜 시간이 계획되지 않지만, 불가피하게 정작 테스트가 시작되었을 때 더 많은 시간을 잡아먹으며 모두를 불안에 빠트리고는 한다. 따라서 초기 일정 수립 시에 테스트에 충분한 시간을 할애하는 것은 중요하다.
* 소심한 추정
  + 일을 맡긴 고객이 결과를 재촉할 때, 일이 예정보다 앞당겨 끝날 순 있지만 항상 올바른 결과물이 더 빠른 시간에 완성되는 것은 아니다.
  + 고객이 바라는 날짜에 맞추려고 잘못된 일정을 세우는 일은 생각보다 프로그래머들이 자주 하는 실수 중 하나이다. 고객이 바라는 날짜가 정량적이지 못한 그들의 예감에 의한 희망 사항이라면, 프로그래머들은 생산성 수치, 버그 발생률 등의 근거를 바탕으로 당당하게 스스로의 추정치를 방어해야 한다.
* 일정 붕괴의 악순환
  + “늦어진 소프트웨어 프로젝트에 인력을 추가로 투입하면 더 늦어지게 된다”는 브룩스의 법칙으로 맨먼스 미신을 걷어낼 수 있다.
  + 프로젝트의 소요 기간은 순서가 있는 내부 요소에 좌우되며, 필요한 인원수는 독립된 하위 작업의 수에 좌우된다.
  + 위의 두 정보를 파악한 관리자는 일정을 짤 때 더 적은 수의 사람과 더 긴 기간은 고려할 수 있으나, 더 많은 사람과 더 짧은 기간에 대해서는 실행 가능한 일정을 만들 수 없다.
  + 부족한 시간 탓에 망가진 프로젝트는 잘못된 프로젝트의 원인 중 최다를 차지함을 잊지 말아야 한다.

**3장: “Surgical team” (프로그래머 수백 명보다 일급 멤버 소수가 낫다)**

* 커뮤니케이션, 그리고 잘못된 커뮤니케이션의 수정에 들어가는 비용이 막대함에 따라, 가능한 한 적은 인원수의 팀을 구성하는 편이 보통 낫다.
  + 하지만 적은 인원수로는 정말 큰 시스템을 빠르게 만들 수 없다.
* 밀스의 제안: 전체 업무를 여러 팀에 분배하되, 외과 수술 팀처럼 조직하라. 즉, 함께 문제에 달려드는 것이 아니라 한 명의 해결과정을 지원하도록 움직인다.
  + 외과의 – 수석 프로그래머. 높은 수준의 지식을 가진 사람이 실질적 접속 권한을 가지고 프로그램의 설계, 테스트 등을 담당한다.
  + 부조종사 – 외과의의 분신으로, 업무의 어떤 부분이든 수행 가능하나 비교적 경험은 적은 편이다. 설계 과정에 참여해 같이 고민, 토론 및 평가하는 역할을 한다. 유사시에 대비하는 보험 역할로 프로그램 전체의 구성을 파악해둘 필요가 있다.
  + 행정 담당 – 돈, 사람, 공간, 장비 등에 대한 최종 결정권은 외과의가 가지고 있으니, 이런 일에 시간을 허비하지 않도록 이러한 행정 업무를 담당하는 사람을 배치한다.
  + 편집자 – 외과의가 작성한 초안, 구술 원고를 읽고 비평, 재작업, 보충 및 버전 관리, 감독 등을 담당하도록 한다.
  + 프로그램 사무원 – 컴퓨터에 입력할 자료를 전달받아 필요에 따라 기록을 남긴다. 팀이 생산하는 모든 기술적 산출물을 하나의 프로그래밍 제품 라이브러리로 관리한다.
  + 도구 담당 – 기본 서비스가 팀에 적합한지 확인하고, 필요한 대화식 서비스 등의 도구를 구축, 유지, 업그레이드한다.
  + 테스터 – 적절한 테스트 케이스를 만들고 디버깅 작업에 테스트 데이터를 제공한다.
  + 언어 전문가 – 어렵고 모호하거나 까다로운 일의 처리를 위해 특정 언어를 깔끔하고 효율적으로 사용하는 방법을 찾아낸다.
* 이런 외과 수술 팀 형식의 분업은
  + 외과의와 부조종사 둘 다 전체 설계, 코드를 알고 있으므로 메모리 할당이나 디스크 접근을 조율할 필요가 없고 제품의 일관성 또한 보장된다.
  + 팀 내 멤버들이 동등한 위치에 있을 경우에 비해 이해관계로 인한 차이 없이, 외과의에 의해 결정을 일방적으로 정리할 수 있다.
  + 더 많은 사람들이 요구되는 작업에서도, 설계를 결정하는 사람의 수를 1/7로 줄였기에 더 효율적인 의사소통을 수행할 수 있다.

**16장: “No silver bullet” (소프트웨어 제작을 혁신적으로 발전시킬 하나의 발전은 없다)**

* 기술이든 관리 기법이든 그 자체로 생산성, 신뢰성, 단순성을 자릿수 하나만큼이라도 향상시킬 발전은 10년 후를 내다보아도 나타나지 않을 것이다.
* 현대 소프트웨어 시스템의 축소 불가한 본질에는 복잡성, 호환성, 변경 가능성, 비가시성이 있다.
  + 복잡성: 소프트웨어 개체들은 어느 하나라도 다른 것과 비슷하지 않은, 본질적으로 아주 복잡한 것이다. 이러한 본질적 복잡성과 비선형적 증가 양상은 팀 구성원들 간의 의사소통을 어렵게 하고, 제품 결함, 비용 초과, 일정 지연 등 다양한 고전적 문제들의 근본적 원인이다.
  + 호환성: 물리학도 복잡한 대상을 다루지만, 물리학자들은 무언가 일관된 원칙이 발견될 것이라는 굳은 믿음을 가지고는 한다. 하지만 소프트웨어 엔지니어에게 그런 것은 없고, 단지 여러 명의 인간에 의해 설계된 결과물이 복잡할 뿐이다. 이러한 호환성을 갖추기 위해, 소프트웨어의 복잡성은 결코 단순화될 수 없다.
  + 변경 가능성: 건물, 자동차들은 제조된 이후에 바뀌는 일이 흔하지 않지만, 소프트웨어 개체는 끝없는 변경 요구에 노출되어 있다. 소프트웨어는 순수한 사고의 부산물이며 기능 확장, 새로운 요구 등 가변성이 무한하다.
  + 비가시성: 소프트웨어는 보이지 않고 시각화 할 수도 없다. 이 본질은 인간의 가장 강력한 직관을 무용지물로 만들어, 설계와 의사소통을 보다 어렵게 만든다.
* 소프트웨어의 본질적 어려움 대신, 부수적 어려움을 해결한 성과들은 존재한다.
  + 고급 언어: 프로그래머가 추상적인 프로그램에서 상상하는 모든 구조물을 제공함으로써, 프로그램의 부수적 복잡성을 상당수 제거해주었다.
  + 시분할 방식: 즉시성을 보존하며 그로 인해 복잡성에 대한 개관을 유지할 수 있도록 한다. 일괄 처리 프로그래밍은 전환 시간이 느리기에 프로그램의 컴파일 및 실행 과정에서 세부 사항을 잊어버리게 되고는 한다.
  + 통합된 프로그래밍 환경: 통합된 라이브러리, 파일 형식 등을 통해 여러 프로그램을 함께 사용할 때 발생하는 부수적인 난점들을 해소하였다.
* 이후에도 새로운 고급 언어들, 객체 지향 프로그래밍, 인공 지능, 프로그램 검증, 더 나은 프로그래밍 환경, 워크스테이션 등을 “은 탄환”의 후보로 제시하지만, 그중 어느 것에도 소프트웨어의 본질적 복잡함을 해결할 마법과 같은 효과는 기대하지 않는다.
* 개념적 본질을 겨냥할 수 있는 시도들
  + 고객이 원하는 바를 반복적인 질문을 통해 보다 구체화시킨다.
  + 소프트웨어를 작성하는 대신 구축한다. 모든 소프트웨어 시스템은 점진적인 개발에 의해 성장될 필요가 있다.
  + “탁월한 설계자들”을 사용하고, 육성한다. 소프트웨어를 만든다는 ‘창조적인’ 과정에서 설계자는 아주 중요하다.