맨먼스 미신 ch1/ch2

Ch1. 타르 구덩이

지난 10여년간 대규모 시스템 프로그래밍 분야는 타르 구덩이 같았다.

프로그램은 그 자체로 완결적이고 개발된 장비에서 개발한 사람에 의해 실행 가능한 상태이다.

이러한 프로그램을 좀 더 유용한 개체로 바꾸는 방법이 두가지 있다.

- 프로그래밍 제품(실행 / 테스트 / 보수 / 확장 가능)

→ 일반화된 방식으로 작성

→ 기본적인 알고리즘이 허용하는 한도 내 입력 데이터의 범위와 형식 일반화

→ 신뢰성 확보를 위한 철저한 테스트

→ 빈틈없는 문서화

→ 프로그램 보다 최대 세 배 이상의 비용

- 프로그래밍 시스템

→ 상호 작용하는 프로그램 묶음이 합쳐진 상태에서 대규모 작업 수행할 수 있도록 기능 조율

→ 입출력이 시스템에 정의된 인터페이스를 따라야함

→ 규정된 만큼의 자원만 사용 (메모리 공간/ 입출력장치 / CPU 시간)

→다른 구성 요소들과 조합 가능한 모든 경우에 대해 테스트

프로그래밍 제품 + 프로그래밍 시스템 ⇒ 프로그래밍 시스템 제품

프로그래밍 시스템 제품은 대다수의 시스템 프로그래밍이 목표로 하는 것이다.

프로그래밍은 왜 재미있는가?

> **첫째**는 무언가를 만드는 데서 오는 기쁨이고,

> **두 번째**는 다른이들에게 쓸모 있는 사물을 만드는데서 오는 기쁨이다.

> **세번째**는 서로 맞물려 돌아가는 부속품으로 이루어진 복잡한 퍼즐 같은 사물을 만들고 거기 숨겨진 법칙이 미묘한 순환 속에서 펼쳐지는 것을 바라보는 매혹적인 경험에서 오는 기쁨이다.

> **네번째**는 지속적인 배움에서 오는 기쁨이다. 마지막은 유연하고 다루고 쉽기에 이로부터 오는 기쁨이다. 프로그래밍은 이렇게 마음 깊은 속에 내재한 창작의 갈망을 충족시키기에 재미있다.

그러나 프로그래밍이 고달픈 점도 있다.

> **첫째** 프로그래밍에는 완벽함이 요구된다.

> **둘째**, 나 아닌 다른이들이 내 목표를 설정하고 이로부터 자원과 정보가 제공된다. 특히나 다른이가 만든 엉망진창인 프로그램(설계 ,구현, 테스트 케이스 방면에서의 문제 등등)에 대한 의존성으로부터 오는 문제점이 많이 발생한다.
> **셋째,** 장대한 개념을 설계하는 것은 재미있지만 버그를 잡는 것을 힘들어 하는 사람들도 많다.

> **네번째**는 오랜시간 동안 만든 제품이 old fashioned로 치부되는 것이다. 그러나, 아이디어를 실현하는 과정에는 단계적인 작업이 필요하므로 실재하는 제품들을 만드는 데는 오랜 시간이 걸린다. 그래서 어떤 제품의 구현체가 한물 갔는지를 논의하려면 실현되지 않은 개념과 동급으로 봐서는 안되고 실재하는 구현체와 비교해야한다.

프로그래밍은 많은 이들을 허우적 거리게 만드는 타르 구덩이임과 동시에 그 나름의 즐거움과 고달픔을 담고 있는 창조적 활동이다.

ch2. 맨먼스 미신

소프트웨어 프로젝트가 망가지는 이유는 무엇일까?

> **첫째** 우리의 추정 능력이 형편없어서 사실과 거리가 먼 가정을 사실로 치부해 버리기 때문이다.

> **둘째** 추정을 하면서 인원과 기간이 상호 교환 가능하다고 보는 것이다.

> **셋째** 자기가 내린 추정에 대해 확신을 내리지 못하기에, 고집이 결여되어있다.

> **넷째**는 일정의 진척도가 제대로 모니터링되지 않는 것이다.

> **다섯째**는 일정이 어긋나는 것을 감지했을 때의 대응이 인력 추가 투입에 그치는 것이다.

모든 프로그래머들은 **낙관론자**다. 그러나 시스템 프로그래밍의 일정 관리의 바탕을 이루는 잘못된 가정은 바로 '모든 작업이 예상된 시간 내에 완료될 것'이라는 가정이다.

프로그래밍의 재료는 유연하고 다루기 쉽기에 가능할 거라는 낙관주의가 발생하게 된다.

추정 및 일정관리에서 투입된 노력을 셈할 때 쓰는 단위는 **맨먼스(man-month)**이다.

프로젝트 비용은 투입된 사람 수와 달 수의 곱에 따라 변하지만 작업 진척도는 그렇지 않다. **맨먼스 단위로 작업량을 추론하는 것은 매우 위험한 일이다**. 사람과 일정을 교환하려면 어떤 일을 하면서 서로 간의 소통이 필요 없는 경우이다. 그러나 프로그래밍은 이에 전혀 해당되지 않는다. **작업의 성격상 순서가 있어 나누기 어렵고 하위작업을 분할하더라도 커뮤니케이션이 필요**하기에 맨먼스 미신에 들어맞는 일이 아니다.

커뮤니케이션으로 추가되는 부담은 **훈련과 의사소통**으로 나뉜다.

모든 작업자는 **기술적 내용, 작업목표, 전반적 전략, 업무 계획에 훈련**을 받아야한다. 훈련 비용은 **추가되는 사람 수에 비례해 증가**한다.

의사소통은 각파트가 모든 파트와 개별적으로 조정을 해야한다면 **n(n-1)/2**배가 된다. 따라서 여려명이 모여서 문제를 해결해야 한다면 일이 더욱 심각해진다.

구성요소 디버깅과 시스템 테스트에서는 순차 처리라는 제약 사항의 절대적 영향 하에 놓인다.

저자가 추천하는 나름의 법칙은 다음과 같다.

> 계획 수립 1/3

> 코딩 1/6

> 구성 요소 테스트와 초기 시스템 테스트 1/4

> 모든 구성 요소가 준비된 후의 시스템 테스트 1/4

이 방식을 일반적인 방식과 비교 해보았을 때 계획 수립, 디버깅 및 테스트에 많은 시간을 할당하고 있음을 알 수 있다. 시스템 테스트에 충분한 시간을 배정하지 않는 것은 아주 참담한 결과를 초래한다. 따라서 초기에 일정을 수립할 때 **시스템 테스트에 충분한 시간을 할애해야 한다.**

또한, 소프트웨어 쪽 관리자들이 일정관리에서 자신감을 갖기 위해서는 생산성 수치, 버그 발생률 추정원칙 같은 것을 마련해야한다. 또한, 추정의 기초가 견고해지기 전까지 소프트웨어 관리자들은 스스로의 추정치를 방어해야한다.

어떤 중요한 프로젝트가 일정에 뒤처지고 있다면 어떤 조치를 취할 것인가

예시 :12MM (3명 4개월간 배정) 각 달이 끝나는 시점마다 A,B,C,D의 마일스톤 세움

2달이 지났는데도 첫번째 마일스톤을 완료하지 못함

1. 작업을 일정에 맞춰 끝내야 함. 9MM가 남았으므로 두달이므로 4.5명이 더 필요하다. 3명에 2명을 추가로 더 투입한다.
2. 작업을 일정에 맞춰 끝내야함. 18MM가 남았다고 예상되므로 두달이므로 9명이 더 필요하다. 3명에 6명을 추가로 더 투입한다.
3. 일정을 재 수립한다. 새로 수립한 일정에서는 일이 신중하고 빈틈없이 완료되도록 충분한 시간을 배정하여 재조정이 없도록 한다.
4. 작업 범위를 축소한다.

첫번째와 두번쨰는 재앙을 초래한다. 달라진 규모는 조직 구성이나 업무 분배에 있어서 정도의 차이를 넘는 본질적 차이를 가져온다.

다음과 같은 **브룩스의 법칙**을 제시할 수 있다.

> “늦어진 소프트웨어 프로젝트에 인력을 추가로 투입하면 더 늦어지게 된다.”

프로젝트에 소요되는 기간은 순서대로 처리해야하는 내부 요소에 좌우되며

필요한 최대 인원수는 독립된 하위작업 개수에 좌우된다.

이를 기반으로 관리자는 더 적은 수의 사람과 더 긴 기간에 기초한 일정을 수립해야할 것이다.