Mythical Man-Month (맨먼스 미신) Summary

POSTECH CSE
20190218 김민결

Chapter2

이 챕터에서는 소프트웨어 시스템 개발 협업에 있어서 사람들이 주로 하는 착각에 대해 기술하고 있다. 이전까지 읽은 Clean Code 책은 한 명의 팀원이 좀 더 유지보수가 쉬운 코드를 작성하는 방법을 다루고 있었다면, 맨먼스 미신에서는 팀 단위의 협업 차원에서 나타나는 현상과 문제들을 다루고 있다.

우선 대규모의 소프트웨어 개발 프로젝트를 엉망으로 만드는 가장 큰 요인은 시간의 부족이라는 문장으로 챕터의 장을 시작하고 있다. 사실, 개인적인 개발 경험이 있다고 해도 어떤 회사의 인턴을 해본 적도 없으며, 1~4인의 프로젝트만 경험해본 사람으로서, 더 큰 규모에서 장기간동안 이루어지는 프로젝트의 분위기가 어떤 지는 정확히 감이 오지 않는다. 하지만, 늘 과제와 시간에 쫓겨 사는 포스테키안으로서, 데드라인이 정해진 모든 일의 수행 퀄리티는, 주어진 시간을 얼마나 잘 쓰는가에 달려 있음을 뼈저리게 경험해온 바 있다.

또한, 일이 밀린 프로젝트에 사람을 더 추가하는 것이 오히려 상황을 더 악화시킨다는 책의 주장은 꽤 신선하게 다가왔던 동시에, 개인적으로 크게 공감이 된 부분이기도 하였다. 과거 한 과목에서의 4인 팀프로젝트 경험을 되새겨보면, 처음엔 모든 팀원이 균등히 프로젝트에 참여하려 했지만, 결국 코드 자체에 결정적으로 기여한 인원은 2명 정도밖에 되지 않았다. 절대로 나머지 한 두명의 팀원이 이기적이라서 그랬던 것이 아니다. 그냥, 한 가지의 일을 두 명 이상에게 분배하는 것이 상당히 비효율적이라고 느껴졌기 때문이다. 당장 구현할 것은 남아있고 데드라인은 다가오는 상황에서, 일을 분배하는 것은 물론, 이미 혼자서 몇백 줄씩 써 놓은 코드를 남에게 설명하는 것이 시간 아깝게 느껴졌던 것뿐이었다. 어차피 혼자서 고생하면 하루 이틀 안에 끝낼 일을, 굳이 남의 상황까지 고려해가며 질질 끌고 싶지 않았던 것이다. 사실 처음부터 일의 분배가 제대로 이루어지지 않았던 점, 그리고 과거의 내가 다소 독단적이었다는 점에서 그랬던 것도 있긴 했으나, 개인이 쓴 코드를 하나씩 보고하고, 다른 사람의 진행 상황에 내 진행 상황을 하나씩 맞춰가야 한다는 것은 여전히 번거롭게 느껴지는 부분이다.

이 챕터에서도 이와 비슷한 이야기를 하고 있다. 보통 흔히들 하는 착각이, 소프트웨어 개발 프로젝트에서도 man-month의 관계, 즉 프로그래머의 수와 프로젝트 수행 기간이 서로 반비례 관계를 나타낸다고 생각하는 것이다. 이러한 착각 때문에, 예상보다 일이 밀리게 되면 도중에 인원을 더 투입하는 실수를 저지르게 된다. 이 책에서 설명하길, 기존에 잘 알려진 man-month의 관계는 오직, 단순 노동과 같이 팀 내 별도의 소통을 필요로 하지 않는 조건 하에서 일이 제대로 분배될 수 있을 때에만 적용될 수 있다. 그러나, 디버깅을 포함하여 모든 일이 순차적으로 이루어져야 하는 소프트웨어 개발의 특성 상, 서로의 소통이 반드시 이루어져야 하기에, 개개인에게 분배된 일에 더하여, 일에 적응하기 위한 개별적인 훈련 기간과 함께 서로 간의 소통을 위한 시간도 'month'에 고려되어야 한다. 인원 수가 늘어날수록 소통에 써야 하는 시간은 점점 늘어나게 되며, 결국 팀 내 인원이 늘어나는 어느 시점부터는 프로젝트 수행에 필요한 기간도 함께 늘어나게 될 것이다. 이 때문에, 일이 계속 밀리고 있는 프로젝트에 무작정 새 인원을 계속 투입하는 것은 오히려 불 난 집에 부채질하는 격이 되어버리는 것이다.

그러면 대체 왜 일이 밀리게 되는 것인가? 물론 저자의 개인적인 견해라고 생각되긴 하지만, 모든 프로그래머는 낙관주의자이기 때문이라고 한다. '이 코드는 문제없이 돌아갈 것이다.' 혹은 '이 버그만 해결하면 무조건 돌아간다.' 라는 생각은 그저 행복회로에서 기인한 것이 아니다. 그냥 제 아무리 좋은 계획과 아이디어

를 가지고 있더라도, 그것들의 결함은 오직 구현할 때에만 나타나기 때문이다. 또한, 프로그래밍 환경이 비교적 다루기 쉽다(tractable)라는 점도 프로그래머를 낙천적으로 만들기에 충분한 요소이다. 즉, 프로그래머들은 종종 사전에 고려하지 않았던 문제들을 실행 또는 디버깅 중에 버그로 만나게 되고, 이들을 해결하는 데에 예상치 못한 딜레이가 발생하게 된다. 또한, 프로젝트에서 각 팀원의 업무가 순차적으로 이루어지는 경우가 많기에, 개별 팀원이 겪는 딜레이가 모두 더해짐에 따라서 일이 점점 밀리게 되는 것이다.

전체 소프트웨어 개발 프로젝트 업무 중, 순차적인 특징이 가장 두드러지게 나타나는 부분은 System Test (and Component Test)로, 추후 발생할 버그의 수를 과소평가하는 낙관적인 프로그래머들로 인해, 항상 Testing이 계획한 것보다 오래 걸리게 된다고 한다. 이러한 점에서, 책의 저자는 아래와 같은 비율로 프로젝트 스케줄을 조정하여, Testing에만 무려 50%의 시간을 쏟는 것을 강력하게 권장하고 있다.

Planning에 1/3, Coding에 1/6, Component test(and early system test)에 1/4, System test에 1/4

Testing에 충분한 시간을 배분하지 않은 상황에서, 앞서 기술한대로 일이 밀리게 되는 시나리오를 생각해보자. 소프트웨어 출시 몇 일 전에 갑자기 예기치 못한 버그들이 쏟아지면, 데드라인까지 문제를 해결하는 것이 무리일 것이며, 결국 사용자에게 불완전한 프로그램을 제공하거나, 출시 기간을 지연시킴에 따라 막대한 재산상의 손해를 야기하게 된다. 특히, 출시 기간이 지연되는 경우, 해당 소프트웨어가 탑재되고 지원되는 다른 기기들의 사업 계획까지도 지연시키기 때문에, 이로 인해 발생하는 막대한 2차적 손해까지도 생각해야 한다. 이러한 점에서 프로젝트에 차질이 생기지 않도록, 엄밀한 사전 수치 및 규칙에 기반하여 스케줄을 계획하고, 가능한 모든 버그를 잡을 수 있도록 Testing에 충분한 시간을 배정하는 것이 중요하다.

챕터 2의 마지막 섹션, Regenerative Schedule Disaster에서는, 소프트웨어 개발 프로젝트의 일이 밀리고 있다고 하여 인원을 더 추가하게 되면, 일의 재분배, 추가 인원의 개별 훈련, Testing의 증가로 오히려 더 큰 딜레이를 발생시키고 상황을 악화시킨다는 사실을 수치적인 예시로 보여주고 있다.

마지막으로 정리하면, 소프트웨어 개발에서만큼은, 순차적인 일의 특성에 따라 프로젝트 수행 기간(months)이 결정되고, 서로 독립적인 일의 수로 프로그래머의 적정 인원(men)이 결정되며, 이 두 가지를 고려해 최소한의 인원으로 넉넉한 기간을 잡아 프로젝트를 수행해야 함을, 그리고 더 많은 인원을 투입하여 더 짧은 기간동안 프로젝트를 수행하는 것은 절대로 불가능함을 강조하며 챕터가 마무리된다.