소프트웨어 엔지니어링이란? - 1

이 글은 "구글 엔지니어는 이렇게 일한다" 책을 보면서 요점 정리를 하는 글 입니다.

프로그래밍과 소프트웨어 엔지니어링의 가장 큰 차이

- 시간
- (규모) 확장
- 실전에서의 트레이드오프(trade-offs)

시간이 프로그램에 미치는 영향

"이 코드의 예상 수명은?" 이라는 질문을 던져보면 좋습니다.

- 단순히 구동 수명(execution lifetime)이 아니라 유지보수 수명 (maintenance lifetime)을 말합니다.

수명이 길어질수록 변경이라는 요소가 점점 중요해집니다.
십 년 이상 살아남게 된다면 간접적이든 직접적이든 프로그램의 거의 모든 의존성 (외부 라이브러리, 기반 프레임워크, 운영체제 등)이 처음과는 달라질 것입니다.
우리가 생각하는 소프트웨어 엔지니어링과 프로그래밍을 가르는 핵심은 이 사실을 인식하는데서 시작됩니다.

이 차이가 우리가 말하는 소프트웨어의 "지속 가능성"의 핵심입니다.
기술적이든 사업적이든 소프트웨어의 기대 생애 동안 요구되는 모든 가치 있는 변경에 대응 할수 있다면 "그 프로젝트는 지속 가능하다" 라고 말합니다.

규모라는 관점에서 본 소프트웨어 엔지니어링

소프트웨어 엔지니어링은 팀 업무입니다.
협업은 그 자체로 새로운 문제를 유발하지만, 한명이 개발하는 것보다 가치 있는 시스템을 만들어낼 잠재력 또한 가지게 됩니다.

소프트웨어 엔지니어링은 의사결정의 복잡성과 이해관계 측면에서도 프로그래밍과 차이가 납니다.
구글에서 진행하는 프로젝트는 대부분 영원히 생존하리라 가정해야 합니다.
즉, 의존성이나 언어 버전 업그리에드가 필요 없어지는 시기를 예츨할 수 없습니다.

하이럼의 법칙

다른 엔지니어들이 사용 중인 프로젝트를 유지보수하고 있다면 "동작한다"와 "유지보수 가능하다" 를 구분 짓는 가장 즁요한 요인은 바로 "하이럼의 법칙" 입니다

하이럼의 법칙 : API 사용자가 충분히 많다면 API 명세에 적힌 내용은 중요하지 않다. 왜냐하면 시스템에서 눈에 보이는 모든 행위(동작)를 누군가는 이용하게 될

것이기 때문이다.

시간의 흐름에 따른 변경과 유지보수를 논하려면 하이럼의 법칙을 알아야 합니다. 소프트웨어 유지보수에 하이럼의 법칙이 적용된다고 해서 계획 세우기를 등한시하거나 소프트웨어를 더 잘 이해하려는 노력을 포기해서는 안됩니다. 문제를 완벽하게 제거할 수는 없어도 완화는 가능하기 때문입니다.

현실에서는 API 사용자가 명세서에는 없는 기능을 찾아 활용하기도 하며, 그 기능이 유용해 널리 쓰이면 추후 API 를 변경하기 어렵게 됩니다.
이러한 사용자가 한 명도 없다면 API 를 훨씬 수월하게 변경이 가능합니다.
따라서 변경이 얼마나 유용할지를 분석할 때는 이러한 충돌을 조사, 식별, 해결하는 데 따르는 비용도 고려해야 합니다.

"당장 돌아가야 한다" 라는 생각으로 작성한 코드와 "언제까지고 작동해야 한다" 라는 생각으로 작성한 코드의 차이를 생각해보면 어떤 관계인지 분명하게 그려질 것입니다.

수명의 관점에서 본 코드

- 이용하는 API의 명세에 명시되지 않은, 즉 언제든 변할 수 있는 기능을 사용하는 코드는 "임시방편적인" 혹은 "기발한" 코드
- 모범 사례를 따르고 미래에 대비한 코드는 "클린"하고 "유지보수 가능한" 코드

위의 두개의 코드는 나름의 목적이 있지만, 어느 쪽을 선택할지는 코드의 기대수명에 크게 좌우됩니다.