프로그래밍

Software Engineering At Google

Jon · 2019. 3. 23. 0:04

URL 복사 이웃추가
본문 기타 기능
공유하기

이 글은 퍼거스 헨더슨의 2017년 논문을 번역한 것입니다.

Software Engineering at Google

We catalog and describe Google's key software engineering practices.

arxiv.org

원문링크 : https://arxiv.org/abs/1702.01715

개요

우리는 구글의 핵심 소프웨어 엔지니어링 실천사례를 목록화하여 설명한다.

작가소개

퍼거스 헨더슨은 구글에서 소프트웨어 엔지니어로 10년 이상 근무했다. 꼬마였던 1979년부터 프로그래밍을 시작하여 프로그래밍 언어의 설계와 구현분야를 전공하였다. 멜번대학교에서 박사학위 지도교수와 함께 연구그룹을 창립하여 Mercury 프로그래밍 언어를 개발하였다. 그는 8개의 국제 컨퍼런스의 프로그램 운영위원으로 활동했으며 50만 라인 이상의 오픈소스 코드를 릴리즈하였다. 유즈넷의 comp.std.c++ 뉴스그룹의 전 의장이었으며 ISO의 C/C++ 커미티의 공인 "기술 전문가"였다. 구글에서 그는 구글에서 널리 사용되는 빌드툴인 blaze 의 원개발자 중 하나였으며 음성인식과 음성명령(시리보다 이전에!), 그리고 음성합성에 사용되는 서버측 소프트웨어 개발에 종사했다. 그는 현재 구글의 TTS (text-to-speech) 엔지니어링 팀을 관리하고 있으며, 여전히 많은 양의 코드를 작성하고 리뷰한다. 그가 작성한 소프트웨어는 10억이상의 기기에 설치되어 있으며 하루에 10억회 이상 사용된다.

1. 도입

구글은 비상한 성공을 계속해 온 회사다. 구글 검색과 애드워즈의 성공에 더하여, 구글은 여러가지 두드러진 제품들을 내놓았는데 구글 맵, 구글 뉴스, 구글 번역, 구글 음성인식, 크롬 그리고 안드로이드가 그것들이다. 구글은 상대적으로 소규모회사들을 인수하여 소유하게 된 많은 제품들을 - 예를 들어 유튜브를 - 크게 개선하고 범위를 확장하였다. 또한 광범위한 오픈소스 프로젝트에 중요

한 기여를 하였다. 구글은 또한 몇가지 놀랄만한 – 예를들어 자율주행 차량 같은 - 미래제품들을 데모하였다.

구글의 성공에는 여러가지 이유가 있을 것이다 – 전문지식을 갖춘 리더십, 뛰어난 인재들, 높은 채용기준, 시장선점에 따른 수익성을 기반으로 한 재정적인 강점 등. 그러나 간과할 수 없는 구글의 성공요인은 **구글이 훌륭한 소프트웨어 엔지니어링 실천사례를 개발해 왔다**는 점이다. 이 실천들은 세계에서 가장 뛰어난 소프트웨어 엔지니어들이 장기간에 걸쳐 쌓아올린 지혜의 산물이다. 우리는 우리의 실천사례들을 세계와 공유하고 싶다. 또한 우리가 그 과정에서 저질렀던 실수에서 배운 교훈들을 공유하고자 한다.

이 논문의 목적은 구글의 핵심 소프트웨어 엔지니어링 실천사례들을 리스트업하고 간략히 설명하는 것이다. 다른 기업이나 개인들은 이것들을 자신들의 소프트웨어 엔지니어링 실천사례와 비교하여 그들에게도 적용가능할지 고려할 수 있을 것이다.

많은 작가들이 구글의 성공과 역사를 분석하는 책과 논문을 썼다. 그러나 대부분은 비즈니스, 관리 그리고 기업문화에 대한 것이었다. 일부의 문서들에서 소프트웨어 엔지니어링 부분을 다루었지만, 단편적인 실천사례를 다루는 것에 그쳤다. 본 논문과 같이 구글의 전체 소프트웨어 엔지니어링 실천사례를 개괄하는 문서는 없었다고 할 수 있다.

2. 소프트웨어 개발

2.1 소스 리파지토리

대부분의 구글 코드는 하나의 통합 소스코드 리파지토리에 저장되어 모든 구글 소프트웨어 엔지니어가 접근가능하다. 언급할만한 예외는 두개의 큰 오픈소스 프로젝트로 (크롬과 안드로이드) 각각은 독립된 오픈소스 리파지토리를 사용한다. 그리고 고부가가치 또는 중대보안 코드들은 읽기권한이 제한되어 있다. 그러나 대부분의 구글 프로젝트는 동일 리파지토리를 사용한다. 2015년 1월 현재, 86TB의 리파지토리에는 10억개의 파일이 있으며, 900만개 이상의 소스코드 파일에 **총 소스라인은 20억라인이다**. 누적된 커밋수는 3500만건이며 매일 4만건의 커밋이 발생한다. 쓰기권한은 통제된다. 각 서브트리의 소유자들만이 자기 서브트리의 변경을 승인할 수 있다. 그러나 어떤 개발자도 어떤 코드에나 접근할 수 있으며, 자신의 로컬변경을 추가해 빌드해 볼 수 있다. 테스트를 돌려보고 코드소유자에게 체인지 리뷰를 신청할 수 있다. 만약 코드소유자가 승인을 하면 해당 변경사항이 반영된다. 기업문화적으로 담당 프로젝트 경계를 무시하고 엔지니어들 각자에게, 문제가 있고 자신이 고칠 수 있다고 생각되는 코드를 고치려고 시도하는 것이 장려된다. 이런 문화가 엔지니어들에게 권한을 부여하게 되고 사용자들의 요구에 따라 더 나은 품질의 인프라스트럭처를 낳게 된다.

거의 모든 개발은 리파지토리의 브랜치가 아니라 "head" 에서 진행된다. (메인 브랜치에서 개발한다는 의미) 이는 코드 통합시의 문제를 조기에 발견할 수 있게 하여 merge 비용을 줄여준다. 이는 또한 보안 픽스를 빨리 만드는 것에도 도움이 된다.

자동화 시스템이 빈번하게 테스트를 수행한다. 해당 테스트가 의존성을 가지는 모든 파일변경에 대해 수행하는 경우가 많은데 이런 방식이 항상 가능한 것은 아니다. 자동화 시스템은 테스트 실패시에 변경사항 작성자와 리뷰어에게 알림을 보내는데, 대개 변경반영후 수분 이내에 보낸다. 대부분의 팀은 자기 모듈의 빌드상황이 화면상 눈에 잘 띄도록 색상을 사용한다. (모든 테스트를 통과하면 초록색, 테스트 실패가 있으면 빨간색, 빌드가 깨지면 검은색) 이렇게 하면 개발자들이 항상 초록색을 유지하려 노력하는 데 도움이 된다. 좀 더 큰 팀단위에서는 "빌드캅"을 두어 항상 "head"에서 테스트가 패스되는지 확인 후, 실패하는 변경 작성자에게 알려서 고치거나 롤백하게 한다. (빌드캅의 역할은 팀원 모두 또는 경험많은 멤버들이 돌아가면서 맡는다.) 이런 노력이 매우 큰 팀에서도 항상 빌드가 테스트를 통과할 수 있도록 해 준다.

코드 소유권. 각 서브트리에는 해당 서브트리의 "소유자"의 아이디 리스트가 저장되어 있다. 서브디렉토리는 부모 디렉토리의 소유자를 상속받는 것이 원칙이고, 예외적으로 상속받지 않는 경우도 있다. 각 서브트리의 소유자는 다음에 나오는 코드리뷰 섹션에서 기술한 바와 같이 해당 서브트리의 쓰기 권한을 제어한다. 각 서브트리에는 적어도 두명의 소유자가 있어야 하는데 대개의 경우 2명보다 많다. 특히 지역적으로 분산된 팀에는 많을 필요가 있다. 모든 팀원이 소유자 파일에 들어있는 경우도 많이 있다. 서브트리에 대한 변경사항은 소유자 뿐 아니라 구글 직원 누구나 만들 수 있다. 다만 소유자가 승인을 해야 반영된다. 이를 통해 모든 변경사항은 수정되는 소프트웨어를 이해하는 엔지니어의 리뷰를 받게 된다.

2.2 빌드 시스템

구글은 Blaze 라고 알려진 분산 빌드시스템을 사용한다. Blaze 는 컴파일과 링크, 그리고 테스트 수행을 담당한다. 이는 전체 리파지토리에 걸쳐 빌드와 테스트를 수행하는 표준 명령을 제공한다. 이 표준화된 명령과 최적화된 구현이 의미하는 것은 **구글 엔지니어는 리파지토리 상의 어떤 소프트웨어도 쉽고 빠르게 빌드와 테스트할 수 있다**는 것이다. 이런 일관성이 엔지니어들이 프로젝트 경계를 넘나들며 변경작업을 하는 것이 가능하도록 한다.

프로그래머들은 담당 소프트웨어에 대해 blaze 빌드에서 사용되는 "BUILD" 파일을 작성한다. 라이브러리나 프로그램, 테스트와 같은 빌드대상들은 **고수준의 빌드 스펙**으로 선언되는데 스펙의 내용에서는 각 대상의 이름과 소스파일, 그리고 그것이 의존하는 라이브러리 등의 빌드대상을 기술한다. 이러한 빌드 스펙은 빌드룰로 구성되는데 빌드룰은 "여기에 C++ 라이브러리가 있는데 소스파일은 이것들이며 다음과 같은 라이브러리에 의존한다" 라는 식의 개념적인 내용을 담고 있다. 빌드룰들을 빌드 단계로 어떻게 매핑할 것인가의 문제는 – 즉, 각 소스파일을 컴파일하는 단계와 링크하는 단계, 그리고 어떤 컴파일러를 쓰고 어떤 플래그를 사용할 것인지 등은 - 빌드시스템에 달려있다.

빌드시스템의 구현은 구글의 분산 컴퓨팅 인프라를 활용한다. 각 빌드작업들은 대개 **수백 수천개의 머신으로 분산된다**. 이는 매우 큰 프로그램이 빨리 빌드되고 수천개의 테스트가 병렬적으로 수행될 수 있게 한다.

각 빌드단계는 "밀폐되어" 있어야 한다 : 선언된 입력값 이외의 의존이 없다는 말이다. 빌드를 분산하기 위해 필연적으로 모든 의존성이 선언되어 있을 수밖에 없다 : 특정 빌드 단계가 수행되는

머신에는 선언된 입력값만 전달된다. 결과적으로 빌드시스템이 실제의 의존관계를 완벽히 알고 있다고 볼 수 있다. 심지어 빌드에 사용되는 컴파일러도 입력으로 받는다.

각각의 빌드 스텝은 결정성을 갖는다. 그래서 빌드시스템은 빌드결과물을 캐시할 수 있다. 소프트웨어 엔지니어는 자기 워크스페이스를 예전의 CL 에 맞추어 빌드하면 정확히 동일한 바이너리를 얻을 수 있다. 그리고 캐시는 서로다른 사용자가 접근 가능하다.

빌드 시스템은 신뢰성이 있다. 빌드 시스템은 빌드룰 자체의 변경도 트래킹하여 입력값이 변하지 않았더라도 산출방법이 달라진 타겟은 재빌드한다. 예를 들어 컴파일러 옵션이 바뀌는 경우가 그러하다. 또한 빌드가 중간에 중단된 경우나 빌드도중에 소스파일이 수정된 경우도 제대로 처리한다 : 이런 경우에는 빌드명령을 재실행하면 된다. 어떤 경우에도 "make clean" 과 같은 명령을 실행할 필요는 없다.

빌드 결과물은 "클라우드에" 캐시된다. 중간 결과물도 포함되는 이야기다. 만약 다른 빌드 요청이 동일한 결과물을 낳는 상황이라면 빌드시스템은 재빌드하는 것이 아니라 자동적으로 캐시된 결과물을 재사용한다. 동일한 사용자가 빌드 요청을 한 경우가 아니라도 캐시된 파일은 활용된다.

증분 재빌드(incremental rebuild)가 빠르다. 빌드 시스템은 메모리에 상주하고 있어서 재빌드시에 가장 마지막 빌드시와 비교해 다른 파일만 증분적으로 분석하여 빌드한다.

프리서브밋 체크. 구글은 코드리뷰를 시작할 때나 변경사항을 리파지토리에 커밋 준비할 때 자동으로 테스트 묶음을 실행하는 도구를 갖고 있다. 리파지토리의 각 서브트리에는 코드리뷰 시점이나 코드 커밋 직전 시점에 어떤 테스트가 수행되어야 하는지 정한 설정파일이 들어 있다. 이 테스트들은 리뷰를 보내기 직전이나 변경을 커밋하기 직전에 수행하는 동기식 테스트가 있을 수 있고, 결과가 리뷰 토의 메일 스레드로 전송되는 비동기식 테스트가 있다. 리뷰 스레드는 코드리뷰가 수행되는 메일 스레드이며 해당 스레드의 모든 정보는 웹기반의 코드리뷰 툴에서도 조회할 수 있다.

2.3 코드 리뷰

구글은 웹기반의 뛰어난 코드리뷰 툴을 개발하였는데, 이메일과 통합되어 있어서 작성자는 리뷰요청을 보낼 수 있고 리뷰어는 깔끔한 컬러코딩의 diff를 보면서 커멘트할 수 있다. 코드 작성자가 코드리뷰를 요청하면 리뷰어는 이메일을 받게되고 해당 변경사항의 웹리뷰툴로 가는 링크가 제공된다. 리뷰어가 자신의 리뷰 커멘트를 달면 이메일 알림이 전송된다. 그리고 자동화된 툴은 자동화 테스트 및 정적분석 툴의 결과를 포함한 알림을 보낼 수 있다.

메인 소스 코드 리파지토리에 적용되는 모든 변경 반영을 위해서는 반드시 적어도 한 명의 다른 엔지니어의 리뷰가 있어야 한다. 그리고 변경 작성자가 변경되는 파일의 소유자가 아닌 경우에는 적어도 한명의 소유자가 리뷰하고 승인하여야 한다.

예외적인 경우에 한해, 서브트리의 소유자는 긴급 변경을 리뷰 *이전에* 반영할 수 있다. 그러나 여전히 리뷰어가 지명되어야 하고 해당 변경이 리뷰되어 승인될 때까지 작성자와 리뷰어는 자동 알림으로 괴롭힘을 당하게 되어 있다. 이 경우에 이미 리뷰가 반영된 후이므로 리뷰 커멘트 자체가 별도의 변경사항이 된다.

구글은 주어진 변경사항에 대해, 코드의 소유권과 작성자를 참고하여, 최근 리뷰어의 기록이나 가능한 후보 리뷰어가 현재 처리대기중인 리뷰요청수를 알려주는 툴을 갖고 있다. 특정 변경사항이 영향을 미치는 각 서브트리별로 적어도 한명의 소유자가 리뷰를 하고 해당 변경을 승인하여야 한다. 그와는 별개로 작성자는 자신이 원하는 대로 리뷰어를 선택할 수 있다.

코드리뷰에 있어서 하나의 잠재적인 문제는 리뷰어가 너무 늦게 응답하거나 승인을 꺼리는 경우인데, 이런 경우에 개발이 느려지게 된다. 작성자가 리뷰어를 선택할 수 있다는 사실이 이런 문제를 피하게 해 준다. 예를 들자면 너무 오래 리뷰를 붙잡고 있는 리뷰어를 피하거나, 간단한 변경 사항은 덜 철저한 리뷰어에게 보내고, 복잡한 변경 사항은 경험이 많은 리뷰어에게 보내거나 다수의 리뷰어에게 보내는 식이다.

각 프로젝트의 코드리뷰 논의는 자동적으로 프로젝트 관리를 위한 메일링 리스트로 복사된다. 누구라도 자신이 리뷰어로 지명되었냐의 여부와 무관하게 커밋 전후에 커멘트를 달 수 있다. 버그가 발견되면 버그를 만들어낸 변경을 추적하여, 원래 작성자와 리뷰어가 인지할 수 있도록 해당 변경의 코드리뷰 스레드에 커멘트를 남기는 것이 일반적이다.

코드리뷰를 여러명의 리뷰어에게 보내서 그들 중 한 명이 승인하는 순간, 다른 리뷰어가 커멘트를 달기 전에 반영되도록 하는 것도 가능하다. (물론 이 경우에 작성자 또는 첫번째 리뷰어가 소유자라야 한다) 이 경우에 이후에 달리는 커멘트는 별개의 추가 변경사항으로 취급된다. 이렇게 하면 리뷰시간으로 인한 시간 손실을 줄일 수 있다.

리파지토리의 메인 섹션과는 별도로 **"experimental" 섹션이 있어서 여기에서는 코드리뷰가 강제되지 않는다.** 그러나 양산 코드는 반드시 메인 섹션에 존재해야 하며 개발자들은 "experimental"에서 개발하다가 메인으로 옮기는 것보다는 메인 섹션에서의 직접 개발을 권장받고 있다. 왜냐하면 코드리뷰는 개발시점에 수행되는 것이 바람직하기 때문이다. 실제로는 experimental 에서도 개발자들은 코드리뷰 요청을 자주 하고 있다.

개별적인 변경은 되도록 작아야 한다고 권장하고 있다. 큰 수정사항을 작은 변경으로 쪼개어 리뷰하도록 하는 것이 큰 덩이를 리뷰하는 것보다 리뷰 용이성이 좋다. 이는 또한 작성자 입장에서도 리뷰과정에서 받는 수정요청에 쉽게 대응할 수 있게 해 준다. 큰 변경사항의 경우는 리뷰어의 제안 변경을 적용하기가 어렵다. 변경 사항을 작게 유지하는 방법 중 하나는 변경줄수에 따라 코드리뷰 툴이 명명하도록 하는 것이다. 예를 들어 30-99 라인의 변경을 "medium" 사이즈라고 하고 할 때,

큰 사이즈의 변경에는 부정적인 이름을 붙이는 방식이다. 300-999 라인의 경우 "large", 1000-1999 라인의 경우 "freakin huge" 와 같이 명명한다.

2.4 테스팅

유닛테스트는 구글 내에서 강하게 권장되며 광범위하게 활용되고 있다. 모든 양산 코드는 유닛 테스트를 가져야 하며 코드리뷰 도구는 소스파일이 유닛 테스트가 없이 추가된 경우 하이라이트 표시를 해 준다. 코드 리뷰어들은 대개 새로운 기능을 추가한 변경사항에는 새 기능을 테스트하는 테스트를 추가해야 한다고 요구한다. (무거운 라이브러리에 의존하는 코드를 가볍게 테스트할 수 있도록 하는) 모킹 프레임워크가 아주 많이 사용된다.

통합 테스트와 회귀 테스트가 널리 활용된다.

"프리서브밋 체크" 항목에서 말했듯이, 테스트는 코드리뷰나 커밋 절차의 일부로 자동화 강제화할 수 있다.

구글은 또한 자동화된 테스트 커버리지 측정도구를 갖고 있다. 그 결과는 소스코드 브라우저의 옵션 레이어로 제공된다. (브라우저가 테스트 커버된 소스 여부를 표시한다는 의미인듯)

배포전에 테스트하라 는 것이 구글의 절대명제다. 개발팀은 인입요청의 양에 따르는 시간지연이나 에러율과 같은 키 메트릭을 보여주는 테이블과 그래프를 작성하도록 요구받는다.

2.5 버그 트래킹

구글은 버그, 기능요구, 사용자 이슈, 릴리즈나 클린업같은 작업과 같은 이슈 트래킹을 위해 Buganizer 라는 시스템을 사용한다. 버그는 구조적 컴포넌트로 분류되고 각 컴포넌트는 디폴트 수신인과 디폴트 메일리스트를 가질 수 있다. 소스변경 리뷰를 보낼 때, 개발자는 변경사항이 어떤 이슈넘버에 대응한 것인지 기입하여야 한다.

구글의 팀에서는 정기적으로 자기 팀의 오픈 이슈를 검토하여 우선순위를 부여하고 적절한 엔지니어에게 할당하는 것이 일반적이다. 몇몇 팀은 버그 분류를 전담하는 담당자가 있기도 하고, 팀 미팅에서 버그 분류 (triage) 를 수행하는 팀도 있다. 많은 팀들은 버그에 대해 레이블을 붙여서 triage 를 거쳤는지를 표시하고, 어떤 릴리즈에서 해당 버그가 수정될지도 표시한다.

2.6 프로그래밍 언어

구글의 소프트웨어 엔지니어들은 다음의 5개 언어중 하나를 사용하도록 강하게 요구받는다 : **C++, Java, Python, Go, JavaScript**. 사용되는 언어의 숫자를 최소화하는 것이 코드 재사용과 프

로그래머 사이의 협력에서의 장애를 줄인다.

각 언어마다 구글 **스타일 가이드**라는 것이 있어서 회사내의 코드가 비슷한 스타일과 레이아웃, 네이밍 컨벤션으로 작성되도록 하고 있다. 그리고 전사적인 **코드 가독성** 훈련 절차가 있어서 언어별로 가독성을 담당하는 숙련된 엔지니어가 다른 엔지니어들에게 가독성있고 관용어법에 맞는 코드를 작성하는 법을 가르친다. 그들은 실제 변경사항들을 리뷰하면서 작성자가 해당 언어의 어법에 맞게 코딩하는 법을 익힐 때까지 리뷰를 해 준다. 뻔하지 않은 신규 코드를 추가하는 변경사항들은 반드시 해당 언어의 "가독성" 과정을 수료한 사람이 승인하여야 한다.

앞에서 언급한 5개 언어말고도 많은 **도메인 특화된 언어**들이 특정한 목적으로 사용된다. (예를 들어, 빌드타겟과 의존관계를 표시하는데 사용되는 빌드 언어)

서로 다른 프로그래밍 언어 사이의 상호작용은 주로 **프로토콜 버퍼**로 수행된다. 프로토콜 버퍼는 데이터 구조를 효과적이고 확장가능한 형태로 인코딩하는 방법이다. 여기에는 구조화된 데이터를 표현하는 도메인 특화 언어가 있으며, 이 언어로 작성된 스펙으로부터 C++, Java, Python 에서 생성하고 읽고 쓰며 시리얼라이즈할 수 있는 코드를 생성하는 컴파일러가 있다. 구글의 프로토콜 버퍼 버전은 구글의 RPC 라이브러리와 통합되어 있어서 쉽게 언어간 RPC 가 가능하도록 해 준다. RPC 프레임웍에서 요청/응답의 시리얼라이즈와 관련된 작업을 자동적으로 처리해 준다.

프로세스의 공통성이 방대한 코드베이스와 다양한 언어 환경에서도 개발이 어렵지 않게 해 주는 열쇠다. 소프트웨어 엔지니어링 작업 차원에서는 동일한 명령 셋이 지원된다고 할 수 있다. (체크아웃, 편집, 빌드, 테스트, 리뷰, 커밋, 버그리포트 등) 이런 명령들은 프로젝트와 무관하고 사용 언어에 무관하다. 개발자들은 프로젝트가 달라지거나 언어가 달라졌다고 개발 절차를 새로 익힐 필요는 없다.

2.7 디버깅 및 프로파일링 도구

구글 서버는 실행중인 서버를 디버깅할 수 있는 도구를 제공할 수 있는 라이브러리와 링크되어 있다. 서버 크래시가 발생하면 시그널 핸들러가 스택 트레이스를 덤프하고 코어파일을 저장한다. 만약 힙메모리 부족으로 크래시가 발생했다면 라이브 힙 개체들의 샘플 부분집합을 이용하여 스택 트레이스를 덤프한다. (뭔말일까?) 디버깅을 위한 웹 인터페이스가 있어서 인입/인출 RPC를 검사할 수 있으며 (타이밍, 에러율, 레이트 제한 등) 명령행 플래그 값을 바꿀 수 있으며 (특정 모듈에 대한 로깅 verbosity 변경 등) 리소스 사용 검사나 프로파일링도 제어할 수 있다. 이런 툴들이 전체적인 디버깅 용이성을 높여주어 gdb 와 같은 전통적인 디버거를 사용할 일은 드물다.

2.8 릴리즈 엔지니어링

몇몇 팀은 전업 릴리즈 엔지니어를 두고 있지만, 대부분의 구글 팀에서는 릴리즈 엔지니어링은 일반 소프트웨어 엔지니어들이 수행한다.

대부분의 소프트웨어에서 **릴리즈는 빈번히 일어난다**: 주간 또는 격일 릴리즈가 일반적이고 매일 릴리즈하는 팀들도 있다. 이것이 가능한 이유는 **대부분의 릴리즈 엔지니어링 작업이 자동화되어 있기** 때문이다. 자주 릴리즈를 하는 것은 엔지니어들에게 동기부여가 되며 (지금 하는 작업이 몇달 후 또는 일년 후에나 릴리즈된다면 그 작업이 흥겹기 힘든 게 당연하다), 반복횟수가 많아지므로 전체적인 개발속도가 빨라지고, 많은 피드백을 받고 수정할 기회가 생긴다.

릴리즈 작업은 보통 신규 작업공간에 가장 최근의 'green' 빌드 (즉, 가장 마지막으로 모든 자동테스트를 통과한 수정사항까지) 를 가져오는 것으로 시작한다. 릴리즈 엔지니어는 체리픽할 추가 수정사항을 선정한다. 즉, 메인 브랜치에서 릴리즈 브랜치로 머지할 수정사항을 고른다. 그러고 나서 빌드를 진행하고 테스트도 수행된다. 만약 테스트 실패 항목이 있으면 문제해결을 위해 메인 브랜치에서 추가 수정을 머지한다. 만약 메인브랜치에 필요한 수정이 없으면 메인 브랜치에 수정작업을 한 후 릴리즈 브랜치로 머지한다. 그리고는 재빌드하고 테스트가 실행된다. 모든 테스트가 패스되면 빌드 바이너리와 데이터 파일이 패키지화된다. 이 스텝은 자동화되어 있어서 릴리즈 엔지니어는 간단한 명령만 실행하면 되고, 좀더 고도화된 경우에는 메뉴 기반의 UI 상에서 기능을 선택해 들어간 후 체리픽할 수정사항을 선택하는 것으로도 가능하다.

후보 빌드가 패키지화되고 나면 보통은 "**스테이징**" 서버에 로드되어 **소규모의 사용자들이 통합테스트를 하게** 된다. (때때로 개발팀내에서 수행된다)

상용 트래픽의 일부를 복사하여 스테이징 서버로 인입시키는 것이 유용한 테스트 방법이다. (이는 테스트용 복제트래픽일 뿐이며 실제 서비스는 상용서버가 수행해야 한다) 스테이징 서버의 응답이 사용자에게 전달되면 안되고 상용서버의 응답만이 사용자에게 전달된다. 이런 테스트를 통해 서버를 상용으로 전환하기 전에 심각한 문제 (예를 들어 서버크래시와 같은) 가능성을 사전에 감지할 수 있다.

다음 단계는 보통 "**카나리**" 서버에 설치하는 것으로 이는 **실제 트래픽의 일부를 담당**한다. 스테이징 서버와 달리 이 서버들은 실제 사용자의 요청을 처리하고 사용자에게 응답한다.

마지막 단계로 특정 릴리즈가 모든 데이터 센터의 모든 서버로 배포된다. 매우 트래픽이 많고 신뢰도가 높아야 하는 서비스에 대해서는 며칠에 걸쳐서 **점진적으로 배포**가 된다. 이를 통해 이전 단계에서 잡아내지 못했던 버그로 인한 서비스 비가용 가능성을 최소화한다.

2.9 출시 승인

사용자가 인지할 수 있는 변경이나 상당한 디자인 변경의 출시는 구현을 담당한 엔지니어링 팀이 아닌 외부의 사람들로부터 승인을 받아야 한다. 특히 코드가 법적계약들을 충족하고, 개인정보보호에 문제가 없으며, 보안성 요구사항을 만족하는지, 안정성은 확보되는지 (예를들어, 서버 한계상황의 자동 모니터링을 하여 담당 엔지니어에게 알림이 전달되는지) , 사업적 요구사항에 부합하는지 등을 검토해야 한다.

출시 프로세스는 주요 신제품이나 신기능이 출시될 때마다 사내의 적절한 사람들에게 알림이 전달되도록 만들어져 있다.

구글은 내부 출시 승인 툴을 가지고 있어서 각 제품에 대해 필요한 리뷰들과 승인들을 기록하고 지정된 출시 절차가 준수되도록 하고 있다. 이 툴은 쉽게 커스터마이징할 수 있어서 제품 또는 제품군마다 별도의 리뷰와 승인 세트를 운용할 수 있다.

2.10 포스트 모텀

상용 시스템에서 심각한 장애가 발새한 경우, 관련 인력들은 포스트 모텀 문서를 작성해야 한다. 이 문서는 사고에 대해 다음과 같은 항목들을 기술해야 한다: 제목, 요약, 영향도, 주요 타임라인, 근본 원인, 유효했던 조치/유효하지 않않던 조치, 향후 보완계획. **이 문서의 초점은 문제 자체와 미래의 재발을 어떻게 피할 것인가이며, 누가 잘못했느냐가 아니다.** 영향도 섹션에서는 장애 시간이나 서비스되지 못한 쿼리의 갯수 (또는 실패한 RPC 수), 또는 매출손실 등으로 사고의 영향을 수치화하는 것이 좋다. 타임라인 섹션에서는 장애가 발생할 때까지의 주요 이벤트와 진단을 위해 취한 작업들, 그리고 해결 조치들을 기술하면 된다. 유효 조치/무효 조치의 섹션에서는 대응과정에서 배운 교훈을 기술한다 - 즉, 어떤 조치들이 빠르게 이슈와 원인을 감지하고 해결에 도움이 되었으며, 무엇이 잘못되었으며, 어떤 구체적인 조치가 미래의 비슷한 문제 발생과 심각도를 줄일지 기술한다.

2.11 빈번한 재작성

대부분의 구글 소프트웨어는 몇년마다 재작성된다.

이 말을 들으면 엄청난 비용이 드는 일이라 생각될 것이다. 실제로 구글의 리소스 중 상당한 부분이 이 작업에 투입된다. 그러나 이 작업은 중요한 효과를 가지고 있어서 구글의 민첩성과 장기적인 성공의 열쇠가 된다. 몇년의 시간이 지나면 어떤 제품의 요구사항은 아주 심하게 바뀌어 있는 경우가 많다. 이는 소프트웨어 환경이나 다른 주변 기술이 변하기 때문이기도 하며 기술과 시장의 변화가 사용자의 요구와 기대에 영향을 주기 때문이기도 하다. 과거의 요구사항에 따라 구현된 몇년전의 소프트웨어는 현재의 요구사항을 충족하도록 설계되어 있지 않다. 그리고 시간이 지남에 따라 복잡도가 증가해 있는 것이 보통이다. 코드를 재작성하는 것은 이제는 별로 필요없어진 요구사항에 맞추느라 누적된 복잡성을 제거해 준다. 또한 코드 재작성은 새로운 팀 멤버에게 지식을 전달하고 코드의 주인이 된 느낌을 부여하는 방법이다. 이 소유감은 생산성에 필수 불가결하다. 엔지니어는 코드를 "자기 것" 이라고 느낄 때, 더 열성적으로 기능을 구현하고 문제를 고치게 마련이다. 빈번한 재작성은 엔지니어들이 다른 제품으로 옮겨다니는 것이 수월하도록 해 주고, 이는 서로 다른 프로젝트의 아이디어들이 긍정적 영향을 주고 받게 만든다. 또한 빈번한 재작성은 코드가 최신의 기술과 방법론으로 작성되도록 해 준다.

3. 프로젝트 관리

3.1 20% 시간

엔지니어는 관리자를 비롯한 어느 누구의 승인없이 자신의 업무시간 중 20%를 임의의 프로젝트에 사용할 수 있다. 엔지니어에 대한 이러한 신뢰는 몇가지 이유로 매우 가치있다. 첫째로, 좋은 아이디어를 가진 누구나 - 다른 사람들이 그 아이디어가 가치있다고 생각하지 않는 경우라도 - 프로토타입을 개발할 수 있는 충분한 시간을 제공받는다. 둘째로, 관리자들이 이 제도가 없었다면 볼 수 없었던 활동들을 관찰하게 해 준다. 20% 시간을 제공하지 않는 다른 회사에서는 엔지니어들이 관리자가 모르게 '비밀개발(skunkwork)' 프로젝트에 매달리곤 한다. 비록 관리자들이 이런 프로젝트의 가치에 동의하지 않더라도 엔지니어가 공개적으로 이런 프로젝트를 수행할 수 있는 편이 훨씬 낫다. 세번째로, 엔지니어들에게 업무시간의 일부 동안 신나는 업무를 할 수 있게 하면, 본 업무에서도 동기 부여가 되고, 덜 신나는 업무에 100% 시간 투입시에 쉽게 목격되는 번아웃이 발생하지 않는다. 동기부여된 엔지니어와 번아웃된 엔지니어의 생산성 차이는 20%보다 훨씬 크다. 마지막으로, 이 제도는 혁신의 문화를 북돋운다. 다른 엔지니어들이 재미있고 실험적인 20% 프로젝트를 수행하는 것을 목격하면, 다른 엔지니어들도 동화된다.

3.2 목표 및 핵심 결과 (OKR)

구글에서 개인과 팀은 자기 목표를 문서화하고 자신의 진척사항이 이 목표와 부합하는지 점검하여야 한다. 팀들은 분기별 및 연간 목표를 정하는데, 여기에는 목표에 얼마나 도달했는지 측정가능한 "핵심 결과"라는 지표가 포함된다. 이 작업은 회사의 모든 레벨에서 수행되어, 위로는 전체 회사 차원의 목표도 정의된다. 개인과 작은 팀의 목표는 그들이 속하는 상위 팀의 목표와 조응해야 한다. 분기말에는 측정가능한 핵심 결과까지의 진척도가 기록되는데, 각 목표에 대해 0.0 (0%달성)에서 1.0 (100%달성) 까지의 값을 부여한다. 이 OKR 점수는 구글 내의 누구나 조회 가능하다. (예외적으로 민감한 정보는 제외된다. 예를 들면 기밀 프로젝트 같은 것들) 그러나 이 점수는 개인의 평가에 직접 활용되지는 않는다.

OKR 값은 높게 설정되어야 한다 : 전체 평균이 65%가 되는 것을 목표로 한다. 즉, 스스로 달성할 것이라고 예측하는 값보다 50% 높게 설정하도록 장려하고 있다. 만약에 어떤 팀이 65% 보다 훨씬 높은 점수를 얻었다면, 그 팀은 다음 분기에 좀더 야심찬 OKR 목표값을 잡도록 권하고 있다. (반대로 65% 보다 많이 낮은 점수를 얻었다면 다음 분기에는 낮추어 잡기를 권할 것이다.)

OKR 들은 회사의 각 부분이 어떤 업무를 하고 있는지 소통하는 핵심 메커니즘이고, 직원들이 좋은 실적을 낼 수 있도록 하는 사회적 인센티브이기도 하다. 엔지니어들은 자기 팀이 OKR 평가를 하는 미팅을 가질 것이라는 것을 알고 있으며, OKR 점수가 직접적인 평가나 급여에 영향을 주지 않는다는 것을 알면서도 자연스럽게 점수를 높게 받으려 노력하게 된다. 객관적이고 측정가능한 핵심 결과를 정의하는 것은 일을 잘 하고 싶은 사람들의 욕구가 목표에 측정가능하게 기여하는 방향으로 발현되도록 돕는다.

3.3 프로젝트 승인

출시 승인을 위해서는 잘 정의된 절차가 있지만, 구글은 프로젝트의 시작이나 취소를 위한 절차는 갖고 있지 않다. 10년 넘게 구글에서 근무했고, 이제는 나 스스로 관리자이지만, 나는 아직도 그런 결정이 어떻게 내려지는지 잘 모르겠다. 하나의 이유는 회사 내에서 절차가 동일하지 않기 때문일 것이다. 각 급의 관리자들은 자기 팀이 어떤 프로젝트를 담당하는지에 책임지고, 자기 재량권을 갖는다. 어떤 경우에는 바텀 업 방식으로 엔지니어가 팀내에서 어떤 프로젝트에서 일할지 스스로 결

정할 자유가 주어지기도 한다. 다른 경우에는 그 결정이 탑다운으로 매니저가 어떤 프로젝트에 우선권을 주고 더 많은 리소스를 투입하고, 어떤 프로젝트를 취소할지 결정하기도 한다.

3.4 회사 재조직

때때로 경영진의 결정에 따라 큰 프로젝트가 취소되기도 한다. 이런 경우 해당 프로젝트에 투입되었던 많은 엔지니어가 새로운 팀에서 새로운 프로젝트를 찾아야 한다. 비슷한 식으로 가끔씩 "조각합치기" 가 진행되는데, 지역적으로 많은 곳에 흩어져 있는 프로젝트를 조정하여 적은 수의 지역으로 조정하는 것이다. 이러면 몇몇 지역의 엔지니어는 팀과 프로젝트를 바꿔야 한다. 이 경우, 엔지니어는 보통 자기 지역에 이미 존재하는 범위내에서 새로운 팀과 역할을 선택할 자유가 주어진다. 또는 자기 팀을 유지하기를 원하는 직원에게는 지역을 옮길 수 있는 옵션이 주어지기도 한다.

팀 통합과 분할이라든가 보고선 변경과 같은 조직 변경은 상당히 자주 일어난다. 솔직히 나는 구글이 다른 회사와 비교해 조직 변경이 더 잦은지는 잘 모르겠다. 매우 큰 기술 기반의 조직에서는 기술이나 요구사항 변경에 따른 비효율을 피하기 위해서 빈번한 조직 변경이 필요한 것 같다.

4. 인력 관리

4.1 직역

구글은 엔지니어링 커리어와 관리자 커리어를 구분하고 있으며, TL 역할은 관리자 역할과 구분되어 있고, 연구기능은 엔지니어링에 합쳐져 있고, 엔지니어 조직을 프로덕트 매니저, 프로젝트 매니저, 사이트 신뢰성 엔지니어 (SRE) 등이 지원하고 있다. 이러한 실행방식의 적어도 일부는 구글에서 개발되고 유지된 혁신문화에 중요하다고 생각한다.

구글은 엔지니어링 내에 몇가지 직역을 정의하고 있다. 각 직역별로 커리어 개발이 가능하며, 여러 단계의 직급이 존재하고 다음 직급의 업적치를 보이는 경우 승진이 가능하다. (승진시에는 월급과 같은 보상이 높아진다. ~~뻔한 말을 왜 하지?~~)

주요 직역은 다음과 같다

* 엔지니어링 매니저

이 리스트에서 유일한 인사관리 직역이다. 소프트웨어 엔지니어와 같은 다른 역할은 사람을 관리할 *수도* 있지만, 엔지니어링 매니저는 *항상* 사람을 관리한다. 엔지니어링 매니저는 대체로 전직 소프트웨어 엔지니어이며, 기술적인 전문성과 함께 사교 능력을 갖추고 있다.

기술적인 리더십과 인력관리는 구별된다. 엔지니어링 매니저는 프로젝트를 리드할 필요가 없다. 프로젝트는 TL 이 리드한다. TL 이 엔지니어링 매니저인 경우도 있지만 소프트웨어 엔지니어인 경우가 더 많다. 프로젝트의 TL 은 해당 프로젝트의 기술적인 결정사항을 최종 결정권을 갖는다.

TL의 선정과 팀의 업적은 매니저의 책임이다. 그들은 팀원의 커리어 개발, 업적 고과 (나중에 언급하는 상호 평가를 활용한다), 그리고 보상을 담당한다. 채용 절차에도 일부 간여하게 된다.

엔지니어링 매니저는 3명에서 30명까지의 인력을 관리하는데, 대부분의 경우는 8명에서 12명 정도이다.

* 소프트웨어 엔지니어 (SWE ~~스웨덴?~~)

소프트웨어 개발에 종사하는 대부분의 인력은 이 직역에 속한다. 구글에서의 소프트웨어 엔지니어 채용기준은 대단히 높다. 예외적으로 뛰어난 소프트웨어 엔지니어만 뽑기 때문에 다른 회사에서 만연하는 소프트웨어 문제들 중 상당수는 없어지거나 최소화된다.

구글은 엔지니어링 부문과 관리 부문의 승진 경로를 분리해 두었다. 소프트웨어 엔지니어가 인력을 관리하는 것이 가능하고, 엔지니어링 매니저로 직종변경을 하는 것도 가능하지만, (최고위직을 포함해서) 승진했다고 해서 인력을 관리할 의무가 생기는 것은 아니다. 고위직급에서는 리더십을 보여주는 게 요구되지만, 여러가지 형태의 리더십이 있을 수 있다. 예를 들어 큰 임팩트를 가지거나 매우 많은 엔지니어들이 사용하는 위대한 소프트웨어를 개발하는 것도 리더십의 사례로 충분하다. 이런 리더십 관점이 중요한 이유는 위대한 기술 소양을 가졌지만 인력관리의 기술이나 의지가 없는 사람도 승진 경로에 문제가 없으며 관리자 역할을 할 필요가 없다는 의미가 되기 때문이다. 이는 여러 조직에서 발견되었던 폐단, 즉 개발능력으로 인해 승진된 사람들이 조직관리를 소홀히 하여 생기는 문제를 피할 수 있게 해 준다.

* 연구 과학자

이 직역의 채용 방침은 매우 엄격하고 그 기준도 극도로 높다. 위대한 논문 기록을 근거로 하는 특출한 연구능력과 **함께** 코드 작성 능력을 요구한다. 소프트웨어 엔지니어 직역에 적합한 학계의 유능한 인재들 중 대다수는 연구 과학자로는 부적격으로 판정될 것이다. 박사학위를 가진 구글의 직원들은 대부분 연구 과학자가 아니라 소프트웨어 엔지니어이다. 연구 과학자의 성과는 발표 논문을 포함한 연구 기여들로 측정되지만, 이런 직역구분에도 불구하고 소프트웨어 엔지니어와 연구과학자는 회사내에서의 업무상 별다른 차이가 없다. 두 직역 모두 오리지널 연구를 수행하고 논문을 낼 수 있으며, 두 직역 모두 새로운 제품 아이디어와 신기술을 개발할 수 있고, 두 직역 모두 코딩을 하고 제품을 개발할 수 있다. 구글에서 연구 과학자는 소프트웨어 엔지니어와 같은 팀에서 동일 제품이나 연구에서 같이 일하는 것이 보통이다. 이렇게 연구 인력을 엔지니어링 팀에 섞이게 하는 접근법은 새로운 연구결과들이 제품이 쉽게 반영될 수 있게 해 준다.

* 사이트 신뢰성 엔지니어 (SRE)

운용중인 시스템의 유지관리를 위해 시스템 어드민을 두는 전통적인 방식 대신에 구글은 소프트웨어 엔지니어링 팀이 시스템을 관리한다. SRE 를 채용할 때의 요구사항은 소프트웨어 엔지니어 직역을 뽑을 때의 요구사항과 약간 다르다. (SRE의 경우 소프트웨어 개발 능력은 약간 낮을 수 있고, 이를 네트워크나 유닉스 시스템 이해와 같은 기술 보유로 상쇄한다) SRE 직역의 특성과 목표는 SRE 책 (제목 "사이트 신뢰성 엔지니어링") 에서 잘 설명하고 있으므로 여기서 따로 설명하지는 않겠다.

* 프로덕트 매니저

프로덕트 매니저는 제품 관리를 책임진다 : 제품 사용자의 대변자로서 그들은 소프트웨어 엔지니어의 업무를 조정하고, 사용자들에게 중요 기능들을 설파하며, 다른 팀과의 업무 조정을 하고, 버그추적과 일정관리를 하며, 고품질의 제품을 산출하기 위해 필요한 것들을 완비한다. 프로덕트 매니저들은 보통 스스로 **코딩을 하지 않고**, 소프트웨어 엔지니어들이 올바른 코드를 작성하도록 보장한다.

* 프로그램 매니저 / 기술 프로그램 매니저 (TPM)

프로그램 매니저는 프로덕트 매니저와 크게 보아 유사하지만, 제품을 관리하는 것이 아니라 프로젝트나 업무절차 또는 특정업무 (예를들어 데이터 수집) 을 관리한다. 기술 프로그램 매니저도 비슷하지만, - 발화 데이터를 다룰 때 언어학과 같이 - 특정한 기술 전문성을 가지고 있다.

소프트웨어 엔지니어와 프로덕트 매니저/프로그램 매니저의 비율은 조직마다 다르지만, 4:1 부터 30:1 정도의 범위에서 대체로 큰 값에 분포한다.

4.2 시설

구글은 건물내에 미끄럼틀이나 당구대, 게임룸과 같은 재미있는 시설을 두는 것으로 유명하다. 이것은 좋은 인재들을 채용하고 유지하는 것을 돕는다. 직원들에게 무료인 구글의 훌륭한 식당들도 비슷한 역할을 하며, 구글직원들이 사무실에 머무르는 것을 간접적으로 지원한다. 배가 고파서 집에 가는 일은 없다는 말이다. 사무동 곳곳에 "마이크로 키친"을 두어, 직원들이 스낵과 음료를 마음대로 즐길 수 있게 한 것도 비슷한데, 한편으로 여기서 많은 대화가 오가므로 이 공간은 비공식적인 아이디어 교환의 장이 되기도 한다. 헬스장과 스포츠센터, 사내 마사지샵은 직원들의 몸관리와 건강, 행복을 돕는다. 이는 생산성을 높이고 이직율을 낮춘다.

구글의 좌석배치는 열린 공간 개념이며, 밀도는 높은 편이다. 논쟁의 여지가 있지만, 이런 배치는 개인들의 집중력을 조금 희생하는 대신 커뮤니케이션을 장려하여 대체로 경제성이 있다.

직원들은 개인 좌석을 할당받는데, 좌석이 꽤 자주 바뀐다 (6개월에서 1년마다, 주로 조직 확장의 결과로 바뀐다) 좌석 배치는 커뮤니케이션을 활성화하기 위한 목적으로 매니저가 정한다. 가까운 좌석간에 대화하기 쉬운 것은 당연할 것이다.

구글은 모든 회의실에 최신의 화상회의 설비를 갖추고 있어서, 다른 조직의 회의 초대에 대해 화면 터치 한번으로 연결할 수 있다.

4.3 훈련

구글은 다음과 같이 직원 교육을 장려한다.

* 신입 구글러 ("Noogler") 에게는 필수 신입교육 코스가 있다.

* 기술 스태프 (SWE 와 연구 과학자) 들은 "Codelabs" 를 하도록 한다 : 개별 기술에 대한 짧은 온라인 교육 코스로 코딩 실습이 포함되어 있다.

* 구글은 직원들에게 다양한 온라인 또는 오프라인 교육 코스를 제공하고 있다.

* 또한 구글은 외부 기관에서 교육받는 것을 지원하고 있다.

그리고 각 신입 구글러는 보통 공식적인 "멘토"를 지정받고, 적응을 도와주는 "버디"도 지정받는다. 매니저와의 미팅이나 팀 미팅, 코드리뷰, 디자인 리뷰, 비공식 모임등에서는 비공식적인 멘토링이 있게 마련이다.

4.4 전배

지식과 기술은 조직의 다른 부분으로 전파하고 조직간의 커뮤니케이션을 활성화하기 위해 **회사의 다른 부서로의 전배가 장려된다**. 한 포지션에서 12개월 이상 일한 후에 프로젝트 전배 또는 오피스 전배가 가능하다. 소프트웨어 엔지니어에 대해 다른 직역에의 임시적인 근무가 장려된다. 예를 들어 사이트 신뢰성 엔지니어로 6개월 정도 로테이션 근무하는 것이 장려된다.

4.5 업적 평가와 보상

구글에서는 피드백이 권장된다. 엔지니어끼리는 서로 "피어 보너스"나 "쿠도"라는 방식으로 공개적으로 긍정적인 피드백을 줄 수 있다. 모든 엔지니어는 임의의 다른 직원을 선택하여 "피어 보너스"라는 이름으로 100달러의 현금 보너스를 매년 2회까지 줄 수 있다. 보통의 업무 범위를 넘어서 기여한 동료에게 줄 수 있는데, 웹 폼에 보너스 지급 사유를 적어넣기만 하면 된다. 피어 보너스를 받는 사람의 팀 동료들도 해당 사실을 알게 된다. "쿠도"를 줄 수도 있는데, 쿠도란 좋은 업적을 조직적으로 인지한다는 의미의 공식적인 칭찬이다. 쿠도에는 따로 금전적인 시상이 없는데, 업무 외적인 기여를 해야 한다는 조건이 없으며 받을 수 있는 회수 제한도 없다.

매니저는 프로젝트 종료시의 스팟 보너스를 포함하여 보너스 부여를 할 수 있다. 다른 회사와 마찬가지로 구글 직원들도 연간 업적 보너스를 받으며 성과에 따라 주식 배분도 받는다.

구글은 매우 조심스럽고 세세한 승진 프로세스를 갖고 있다. 직원 자신 또는 매니저가 승진 후보로 추천을 하면, 자기 리뷰, 동료 리뷰, 매니저 고과 등을 거치게 된다. 이 과정들을 입력으로 하여 승진 위원회에서 실제 승진 결정을 내리게 된다. 이의가 있는 경우 승진 재심 위원회가 열리기도 한다. 맞는 사람이 승진하도록 보장하는 것이 직원들에게 제대로 동기부여를 하는 것이다.

반대로, 저성과 인력에는 매니저 피드백이 있게 된다. 필요한 경우에는 구체적인 업적 목표와 진척도 점검을 포함하는 업적 개선 계획이 피드백과정에 포함된다. 그것도 잘 되지 않으면, 저성과 인력 해고가 가능하다. 그러나 구글에서 해고는 *극도로* 드물다.

매니저의 성과는 피드백 설문으로 측정된다. 모든 직원은 1년에 두번씩 자신의 매니저의 성과에 대한 조사에 답해야 한다. 이 결과는 익명화되고 합산되어 매니저에게 전달된다. 이런 상향 피드백은 조직내에서 관리의 품질을 유지하고 개선하는 데 매우 중요하다.

5. 결론

구글에서 사용되는 대부분의 소프트웨어 엔지니어링 실행 사례들을 간략히 살펴보았다. 물론 구글은 지금 매우 큰 회사라서, 몇몇 부분들은 매우 다른 식으로 일하고 있을 수도 있다. 그러나 여기서 설명한 것들이 구글의 대부분 조직들이 따르는 방식이다.

이렇게 다양한 소프트웨어 엔지니어링 실행 사례가 있고, 또 이런 사례와 무관하게 구글이 성공할 수많은 이유들이 있으므로, 특정 방법이 얼마만큼의 개선을 가져온다고 말하기는 쉽지 않다. 그러나 이러한 실행 사례들이 구글에서 시간의 테스트를 견디며 수많은 일류 소프트웨어 엔지니어의 판단을 거친 방법들이란 것은 의심의 여지가 없다.

이 논문에서 언급된 특정 방법론을 자기 조직에서 설파하려는 다른 회사의 분들에게 내가 도와줄 수 있는 말은 이것이다.

"구글에서는 잘 통했어요"