How Google Engineers Work

https://product.kyobobook.co.kr/detail/S000061352347

바위 위에 지어지는 것은 없습니다. 모든 것은 모래 위에 지어지죠. 하지만 우리는 모래가 바위라고 생각하고 지어야 합니다.

프로그래밍과 소프트 엔지니어링의 가장 큰 차이 : 시간, (규모) 확장, 실전에서의 트레이드오프

소프트웨어 엔지니어링은 흐르는 시간 위에서 순간순간의 프로그래밍을 모두 합산한 것이다.

지속 가능성

시간과 변경 - 기대 수명, 업그레이드,

- 하이럼의 법칙 : API 사용자가 충분히 많다면 API 명세에 적힌 내용은 중요하지 않습니다. 시스템에서 논에 보이는 모든 행위(동작)을 누군가는 이용하게 될 것이기 때문입니다.
- 이용하는 API 명세레 명시되지 않은, 즉 언제든 변할 수 있는 기능을 사용하는 코드 = 임시방편적인 혹은 기발한 코드 / 모범 사례를 따르고 미래에 대비한 코드 = 클린 혹은 유지보수 가능한 코드
- 갈아타기 규칙

확장 가능한 정책들

- 비욘세 규칙 = 네거 좋아했다면 CI 테스트를 준비해뒀어여지

코드 베이스에 유연성을 주는 여러 요인들 - 전문성, 안정성, 순응, 익숙함, 정책

원점 회귀 : 보안성 점검을 개발 프로세스의 마지막으로 연기하면 안된다고 호소하며 보안을 고려하는 시점을 왼쪽으로 이동시켜라

금융 비용, 리소스 비용, 인적 비용, 거래 비용, 기회 비용, 사회적 비용

제번스의 역설 : 효율이 좋아지면 자원의 소비가 늘어난다

분산 빌드 시스템

인간 = 간헐적 버그들의 집합

기술 전문가 셀럽 현상

버스 지수 : 몇 명의 팀원이 버스에 치어서 일을 할 수 없게 될 때 프로젝트가 망하게 되는지를 나타내는 지수

눈이 많아야 버그가 줄어든다.

눈이 많아야 프로젝트가 탈선하지 않고 옳은 방향으로 나아간다

홀로 일하기는 함께 일하기 보다 본질적으로 더 위험합니다.

사회적 상호작용의 세 기둥

- 사회적 스칼의 세 기둥 : 겸손, 존중, 신뢰

  > 자존심 버리기, 비평하고 비평받는 법 배우기, 빠르게 실패하고 반복하기

훌륭한 포스트모템 문화

- 사건의 개요, 사건을 인지하고 해결에 이르기 까지의 타임라인, 사건의 근본 원인, 영향과 피해 평가, 문제를 즉시 해결하기 위한 조치 항목, 재발 방지를 위한 조치 항목, 해당

경험에서 얻은 교훈

> 인내심을 기르자, 마음을 열고 받아들이자

모호함을 뚫고 번창한다, 피드백을 소중히 한다, 저항(항상성)을 극복한다, 사용자를 우선한다, 팀에 관심을 기울인다, 옳은 일을 한다

전문가, 지식을 전할 메커니즘 < 배움의 문화, 심리적 안전의 기반 필수

정보의 섬 - 정보 파편화, 정보 중복, 정보 왜곡

단일 장애점

전부 아니면 전무 전문성

앵무새처럼 흉내내기

유령의 묘지

체스터슨의 울타리 원칙

가독성 제도 : 단순한 코드 가독성 + 구글 전사 차원의 표준 멘토링 프로세스

> 가독성 자격증

가치 / 결과

- 자신을 솔직하게 바라보고 성찰하기, 모두와 함께 만들기, 제품을 이용하기 가장 어려운 이들을 위해 설계하자, 가정하지 말고 시스템 전반의 공정성을 측정하자, 변할 수 있다

관리자 / 테크 리드

관리자염

피터의 법칙

섬기는 리더십

전통적인 관리자는 일을 어떻게 처리할지 고민, 훌륭한 관리자는 무슨 일을 처리할지를 고민합니다.

안티패턴

위임하되, 곤란한 일은 직접 처리하자

파도를 일으켜야 할 타이밍을 알자

실패해도 쉽게 되돌릴 수 있는 일에는 해보세요 라고 말하자

외적 동기, 내적 동기

자율성

3A 리더십 : Always Be Depending, Always Be Learning, Always Be Scaling

결정 하고 반복 하기 > 분석 마비 조심

분석 > 분투 > 견인 > 보상

데이터 주도 회사

GSM 프레임워크 : 목표와 신호를 뒷받침하는 의미있는 지표 선정하기

가로등 효과

QUANTS = 코드 품질 , 엔지니어들의 몰입도, 지적 복잡성, 박자와 속도, 만족도

경험 표집법 (ESM)

규칙과 지침 - 프로그래밍 스타일 가이드

기본 원칙 : 규모와 시간 양쪽 측면에서 탄력적인 엔지니어링 환경이 지속되도록 하는 것

스타일 가이드 : 위험을 피하기 위한 규칙, 모범 사례를 적용하기 위한 규칙, 일관성을 보장하기 위한 규칙

스타일 중재자

코드 포맷터 - 프리서브밋 검사

코드 리뷰 : 정확성과 용이성, 코드 소유자로부터 변경 토드의 적절성, 가독성

그린필드 코드 리뷰 : 완전히 새로운 코드를 대상으로 하는 가장 드문 유형의 코드 리뷰

글쓰기와 프로그램을 같이 가는 기술로 보기

독자 유형 : 경험 수준 + 도메인 지식 여부 + 목적

: 탐색자 or 배회자

문서 자료의 리뷰 : 정확성, 명확성, 일관성
좋은 문서 자료의 특징 : 완전성, 정확성, 명확성

문서의 신선도 보증 기간

테크니컬 라이터

자동 테스트 : 테스트 작성 > 테스트 수행 > 실패
한 테스트에 대한 조치
테스트 스위트

모든 테스트 케이스의 두 가지 독립된 요소 : 크기 + 범위

작은 테스트 : 빠르고 결정적, 제약 가장 엄격, 하

나의 스레드까지만 포함되는 경우

중간 크기 테스트

큰 테스트

> 모든 테스트는 밀폐되어야 함. 셋업, 실행, 테어다운 하는데 필요한 모든 정보 포함

단위 테스트

마이크 콘의 테스트 피라미드

테스트 스위트 안티 패턴 .. 아이스크림 콘

코드 커버리지 : 어느 테스트가 기능 코드의 어떤 라인을 실행하는지 측정하는 수단

모노리포, 하나의 리포티지에 관리

오리엔테이션 수업

테스트 인증 프로그램

화장실에서도 테스트

탐색적 테스팅 : 창의력 요구

깨지기 쉬운 테스트 : 실제 버그가 없음에도, 검증 대산 코드와 관련 조차 없는 변경때문에 실패하는 테스트

상태 테스트

상호작용 테스트

명확한 테스트

완전한 테스트

간결한 테스트

DAMP = 서술적이고 의미 있는 문구

테스트 인프라 : 가끔 다른 테스트 스위트와도 코드를 공유하면 유용할때가 존재

테스트 대역 -> 테스트 용이성, 적용 가능성, 충실성의 장점 제공

테스트하기 쉽다 = 단위 테스트를 고려해 짜인 코드

이어주기 .. 의존성 주입

모의 객체 프레임워크 : 테스트 대역을 쉽게 만들어주는 소프트웨어 라이브러리

테스트 대역 활용 방법

- 속이기 (가짜 객체
- 뭉개기 (스텁 -> 과용의 위험성 : 불명확, 깨지기 쉬움, 테스트 효과 감소
- 상호작용 테스트

고전적 테스트 : 실제 구현을 선호하는 테스트

모의 객체 중심주의 테스트 : 모의 객체 프레임 워크를 선호하는 테스트

객체 - 실행 시간, 결정성, 의존성 생성

계약 (명세) 테스트

변경 검출 테스트

단위 테스트의 손대기 어려운 영역

- 부정확한 테스트 대역
- 설정 문제
- 과부하 시 나타나는 문제
- 예기치 못한 동작, 입력, 부작용

- 창발적 행위와 진공 효과

큰 테스트의 구조 : 테스트 대상 시스템 확보 > 필요한 테스트 데이터 준비 > 대상 시스쳄을 이용해 동작 수행 > 행위 검증

SUT - 밀폐성, 충실성

: 단일 프로세스 ~, 단일 머신 ~, 다중 머신 ~, 공유 환경, 하이브리드

대규모 테스트는 시드 데이터와 테스트 트래픽 필요

: 도메인 데이터, 현실적인 기준성, 데이터 기록 API, 손수 가공한 데이터, 복사한 데이터, 샘플링한 데이터

검증 : 수동 검증, 단정문, A/B 비교

탐색적 테스팅 == 퍼즈 테스트

버그 파티

사용자 인수 테스트 (UAT) : 공개 API를 통해 제품을 조작하면서 특정 사용자 여정이 의도한 대로 이루어지는지를 보장하는 테스트

프로버 : 프로덕션 환경을 대상으로 단정문을 수행하는 기능 테스트

카나리 분석 : 프로버와 우시, 대신 신버전을 프로덕션 환경에 언제 배포할 지 주된 관심

재해 복구 테스트 (-DiRT)

카오스 엔지니어링 : 시스템에 꾸준히 결함을 심어서 무슨 일이 벌어지는지 관찰하는 테스트

큰 테스트 수행 > 테스트 속도 개선, 불규칙한 결과에서 벗어나기, 이해되는 테스트 만들기,

폐기 : 궁극적으로는 낡은 시스템을 완전히 걷어내는 과정

: 권고폐기 (희망폐기), 강제폐기

: 실행 가능성과 적시성 경고메세지에 담기 필수

발견 > 마이그레이션 (이주) > 퇴행 방지

트렁크 기반 개발 : 확장성 뛰어남

버전 관리 (VCS) : 파일의 시간에 따른 변경 기록 (버전)을 추적하는 시스템, 파일들의 메타데이터 관리

: 중앙 집중 형 VCS / 분산형 VCS

개발 브랜치 : 구현은 다 했지만 커밋하진 않았어요 와 이제부터 이 코드를 기준으로 개발하세요 의 중간 단계

원 버전 : 어느 리포지터리의 어느 브랜치가 진실 공급원인지를 개발자가 명확히 알도록 한다는 단일 진실 공급원의 개념을 확장한 개념

원버전 규칙 : 개발자가 이 구성요소는 어떤 버전을 사용해야 하죠?라고 묻는 상황을 만들지 않아야 합니다.

빌드 호러아즌

Code Search UI

랭킹

: 쿼리 독립적 시그널 / 쿼리 의존적 시그널

: 검출(검색) + 결과 다양성

빌드 시스템의 목적 : 속도, 정확성

태스크 기반 빌드 시스템의 어두운 면

: 빌드 단계들을 병렬로 실행하기 어려움

: 증분 빌드 수행 어려움

: 스크립트를 유지보수하고 디버깅 하기 어려움

아티팩트 기반 빌드 시스템

분산 빌드 : 원격 캐싱, 원격 실행

내부 의존성 / 외부 의존성 / 전이 외부 의존성

때때로는 엔지니어의 힘과 유연성을 제한해야 생산성을 높일 수 있다.

코드 리뷰 도구 원칙 : 간결성, 신뢰 제공, 익숙한 소통 방식, 워크플로 통합

코드 리뷰 흐름 : 변경 생성 > 리뷰 요청 > 댓글 달기 > 변경 수정, 댓글에 답하기 > 변경 승인 > 변경 커밋

관심 집합

정적 분석 : 프로그램을 실행하지 않은 채로 소스 코드를 분석하여 버그나 안티패턴 등의 잠재적인 문제를 찾아내는것 <_> 동적 분석

거짓 양성 비율 -> 유효 거짓 양성

프리서브밋 검사

의존성 관리 의 어려움 : 문제 정의 부터 곤란

ABI 호환성

변경 자체만으로는 파괴적인지 아닌지 판단 할 수 없음

대규모 변경 (LCS)

유령의 묘지 : 나무 오래되고 둔하고 복잡해서 아무도 손대려 하지 않는 시스템

지속적 통합 (CI) : 팀원들이 작업 결과를 자주 통합하는 소프트웨어 개발 방식. 통합할 때마다 자동 빌드 (테스트 포함) 하여 통합 오류를 빠르게 찾아낸다.

빠른 피드백 루프 : 편집/컴파일/디버그 > 프리서브밋 > 포스트서브밋 > 릴리스 후보 (RC) > RC 승격 > 최종 RC

카나리 배포, 버전 왜곡

실험, 기능 플래그

지속적 빌드 (CB) ; 참 헤드, 녹색 헤드

지속적 배포 (CD) : 릴리스 자동화

빠를수록 안전하다

민첩성, 자동화, 격리, 신뢰성, 데이터 중심 의사 결정, 단계적 출시

실패 관리

테스트의 불안정성

밀폐 테스트 : 모든 것을 갖춘 테스트 환경에서 수행하는 테스트
; 거짓 양성, 거짓 음성

TAP

늘 배포하라

저는 컴퓨토를 이해하려고 애쓰지 않아요. 프로그램을 이해해보려는 거죠.

서비스형 컴퓨트 (CaaS)
: 간단한 자동화, 스케줄링 자동화, 적정 규모화와 자동 확장

배치 작업 : 처리량 중요, 수명이 짧음

서빙 작업 : 지연 시간 중요, 수명이 대체로 김. 구동되는데 오래 걸림

멱등성

중앙 관리 / 사용자화

서버리스

트레이드 오프