INFCON 2023

지속 가능한 소프트웨어 개발을 위한 경험과 통찰

케이타운포유 백명석

소개

- **(주)케이타운포유**, 2022.06~: **K-POP 글로벌 온오프라인 통합 플랫폼**, CTO
- **11번가(주)(SKPlanet)**, 2016.11-2021.12: Portal개발그룹장
- **Daum(Kakao)**, 2006.03-2016.10: 검색개발유닛장
- **(주)트랜스넷**, 2002.1 – 2006.03: VoIP
- **O1 Inc**, 1999.09 – 2001.12: 인터넷 투자은행
- **LG-CNS**, 1996.10 – 1999.08: 미들웨어
- https://linktr.ee/codetemplate

목차

- FAQ
- 개발하며 배운 교훈
- 일하는 방법에 대한 발견

규율(Discipline)

- 본질적인 부분(**Essential**)
  - 규율의 **존재** 이유
  - 규율에 **권위** 부여
- 임의적인 부분(**Arbitrary**)
  - 규율에 **형태**/**실체** 부여
  - 이 부분 없이는 규율 존재 불가

How To Perform Surgical Hand Scrubs

지속 가능한 소프트웨어 개발 - 1

- 변화하는 **요구사항**을 지속적으로 수용
  - SW 비용: 서비스 오픈 이후가 **80%**
- **품질**과 **비용**의 관계
  - **제조업**: 품질 ↑ → 비용 ↑ ← **직관적**
  - **SW**: 품질 ↑ → (향후 변경) 비용 ↓ ← **비직관적**
- **비직관성** 때문에...

지속 가능한 소프트웨어 개발 - 2

- 변화하는 **환경**에 맞는 개발 방법
  - 클라이언트/서버: 터미널, vi ← **지도(사전 설계)**
  - 인터넷: PC/노트북, IDE ← **네비게이션(리팩터링)**
    - Inside Out, Outside In **TDD**
    - **모의객체**(점진적으로 협력 인터페이스 발견)
    - **수직 슬라이스**(Walking Skeleton)

https://github.com/msbaek/presenting-tdd

FAQ

우리나라는 왜 "백발의 개발자"가 없나 ?

- Robert C. Martin: 1952년생
- Martin Fowler: 1963년 생
- Kent Beck: 1961년생
  - 9살. 실리콘밸리. 개발자 아버지의 책

할아버지 개발자를 만나고 싶다

이직하면 해결되나 ?

- 회사를 다니는 이유
  - **기여** / **배울 것** / **미래의 나**에게 도움
- 이직
  - 회사, 조직, 리더 - **방향성**(Align)
  - **회사**가 아니라 **할 일**을 보고
  - **처우**: 적응, 성과 - 부작용(Side effect)

주니어와 시니어는 어떻게 다른가 ?

- 주니어
  - **주어진 일**을 잘해야(기능, 일정 + 품질)
  - 매번 물어라
- 시니어
  - 업무의 **완결성**, **품질**
  - 주위 동료들이 잘하도록 **도와야**
- 그리고 …

좋은 줄은 알겠는데 환경이 ?

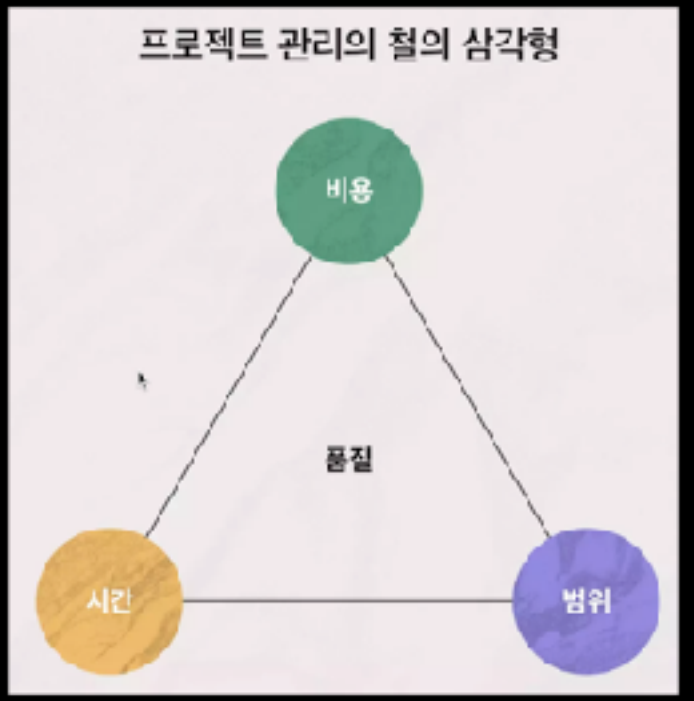
- **"내 관리자가 ~을 허용하지 않을 것"**
  - 정말 문제가 있는 조직도 존재
- 우리의 **전문적인 일**에 **허락**이 필요한가 ?
  - 요리사: 칼 갈기, 설겆이
- **무엇, 언제까지** vs **어떻게**
  - TDD, 리팩터링...
  - 일정내에 원하는 기능을 제공하면

좋은 줄은 알겠는데 환경이 ?

- **대립/갈등** 상황
  - 대부분의 개발자는 꺼림. 하지만 익혀야
- **불필요한 승인**을 요청한 것은 아닌가 ?
  - 리더에게 너무 상세히 알려주면

시간이 부족하다

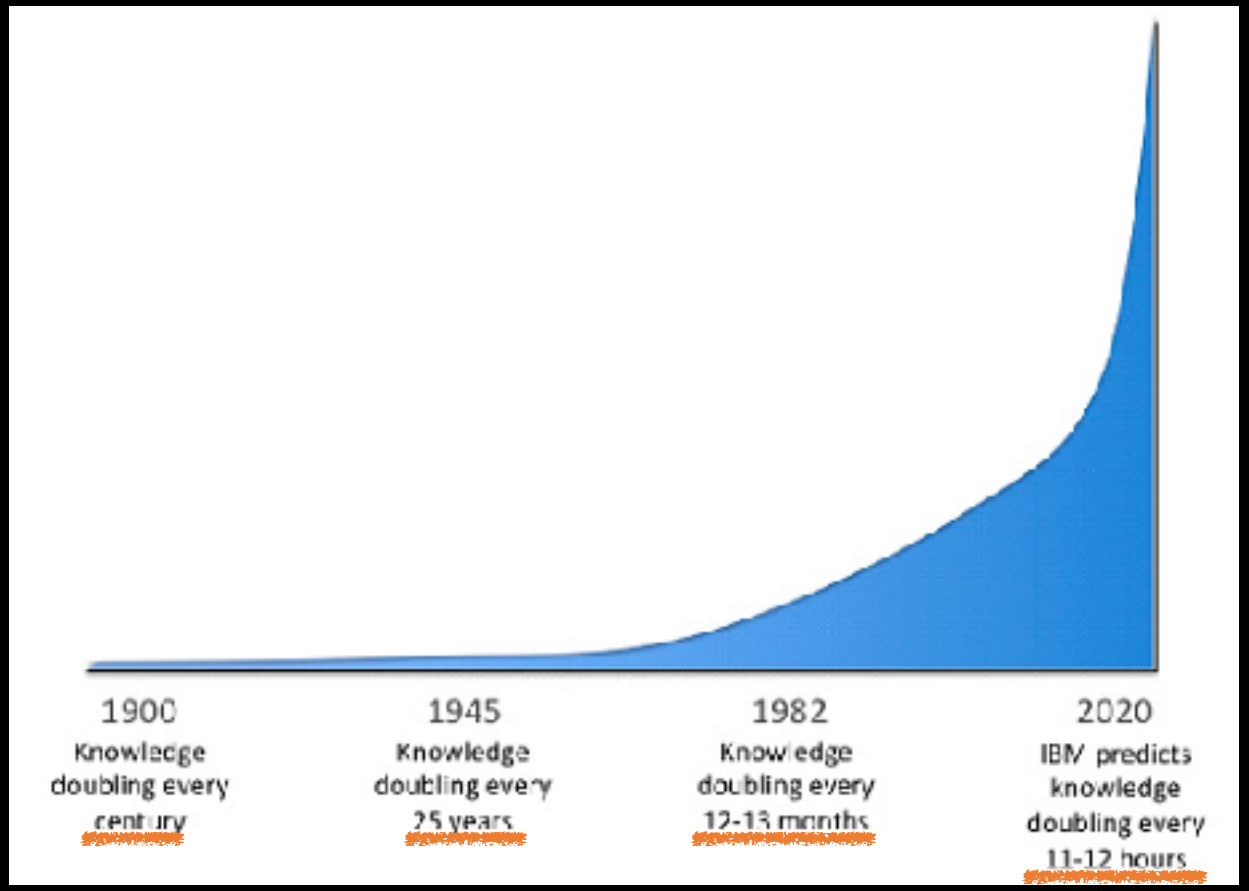
- **시간** vs **실력**
- 시험 시간
  - 시험 시간 안에 답안을 내야 함 ← 실력
  - 부족한 실력은 **學習**으로 보충해야
  - **"아는 만큼 보인다"**
- 품질은 항상 100점이여야 하나 ?
  - **Trade-off**, **실용적**이여야



https://asana.com/ko/resources/project-management-triangle

무엇을 공부해야 하나 ?

- **지식 배가 곡선**
  - 지금 무엇을 알고 있나 ?
  - 얼마나 잘 배울 수 있나 ?



Buckminster Fuller's Knowledge Doubling Curve, with post-1982 addition by IBM

무엇을 공부해야 하나 ?

- SW Development
  - Software development is a process of **discovery** and **exploration**; therefore, to **succeed** at it, software engineers need to become **experts at learning**
- **공부하는 방법**

https://www.amazon.com/Modern-Software-Engineering-Better-Faster/dp/B0BLXCXT3R

어떻게 공부해야 하나 ?

- **루틴**
- RSS, 뉴스레터, SNS, 유튜브 강연
  - 제목 / 소개 / 전체 / 튜토리얼, 책 - todo
  - **근시일** 내 할 일에 깊게 투자

개발자 동기 부여

- How do you inspire your team to adopt … ?
  - **좌절** 준비
  - 자신만 제어 가능(힘듦). 타인은 제어 불가
  - **영감은 부산물**
    - **모범**이 되라 - “나를 보고 **따라하고 싶어야**”
    - 단축키, 툴, 테마 등에서 시작

https://www.youtube.com/watch?v=ajpa7COOptg

개발자 동기 부여

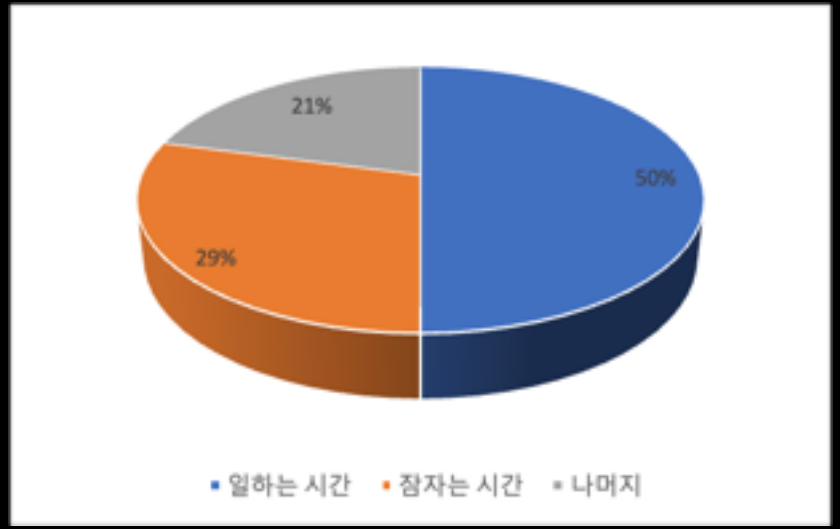
- 동기 부여를 위해 하지 말아야 할 결정을 …
  - **회사** / **업무**를 위한 결정 vs 개발자의 **경력**을 위한 결정
  - 예. **리텐션**
    - 다양한 신기술을 도입한 레거시 운영툴
    - 과한 기술 도입(REST vs ETL)
    - 떠날까봐 승인 → 경력 추가 → 이직 → 운영의 어려움

왜 성장해야 하나 ?

- 직업이란 ?
  - **Job**, **Occupation**, **Do for a Living**
  - "사회인은 **전문가**여야 한다", Daum 부사장님
    - **Professional**
  - 드라이퍼스 모델, "백발의 개발자가 되기 위한 커리어 패스"
    - 초급자, 초중급자, 능숙자, 숙련자, **전문가**

왜 성장해야 하나 ?

- **워라밸**
  - 근무시간: 8 + 1(점심시간) = 9
  - 출퇴근 + 준비: 1 + 1 + 1 = 3
  - 24 - 7(취침) - 12 = 5
  - **어떤 시간이 즐거워야 하나 ?**
  - "행복은 기쁨의 **강도**가 아니라 **빈도**다", 행복의 기원



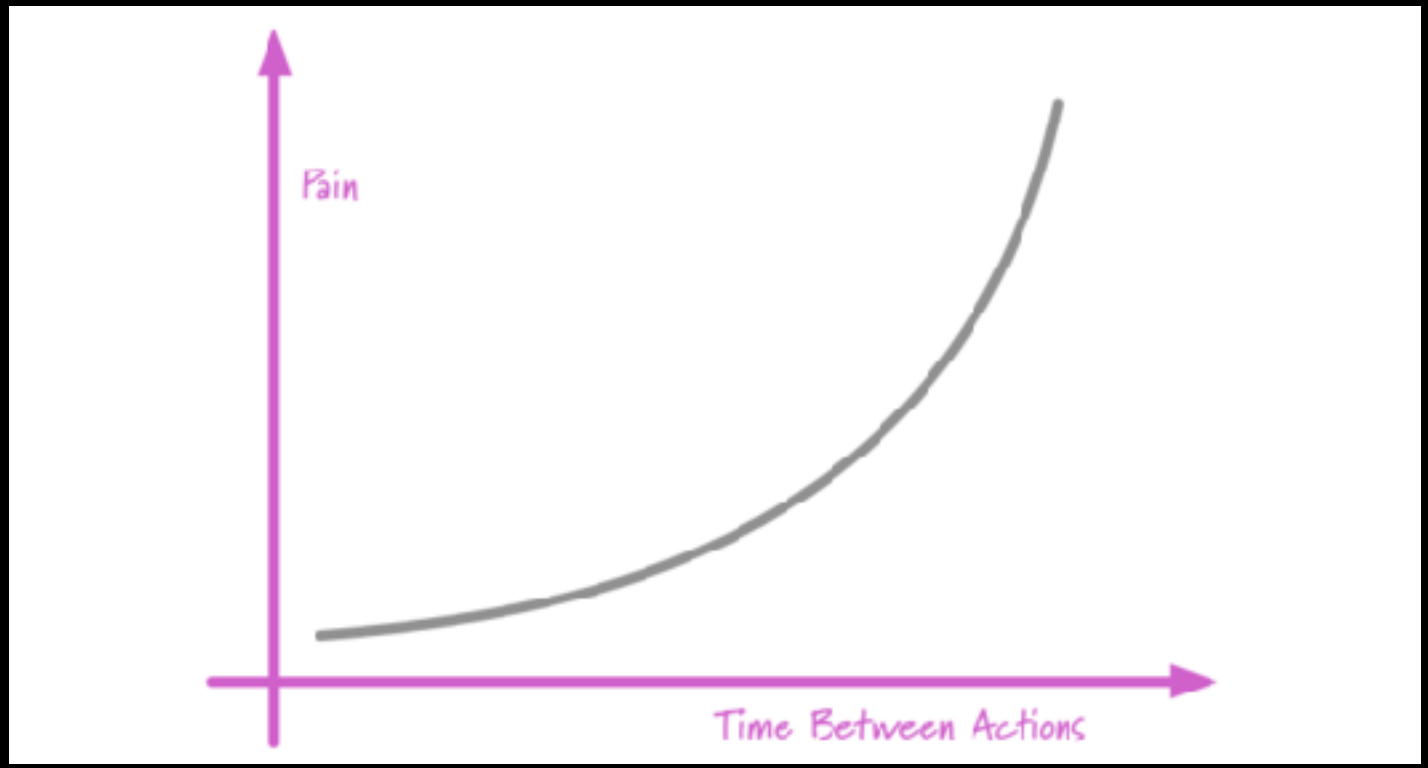
어려운 기술을 배우는 방법

- **쉬운 문제로**
  - **한번 푼 문제**를 TDD로 풀면서 TDD를 익히는 방법

https://twitter.com/kentbeck/status/1421257648914137090

어려운 기술을 배우는 방법

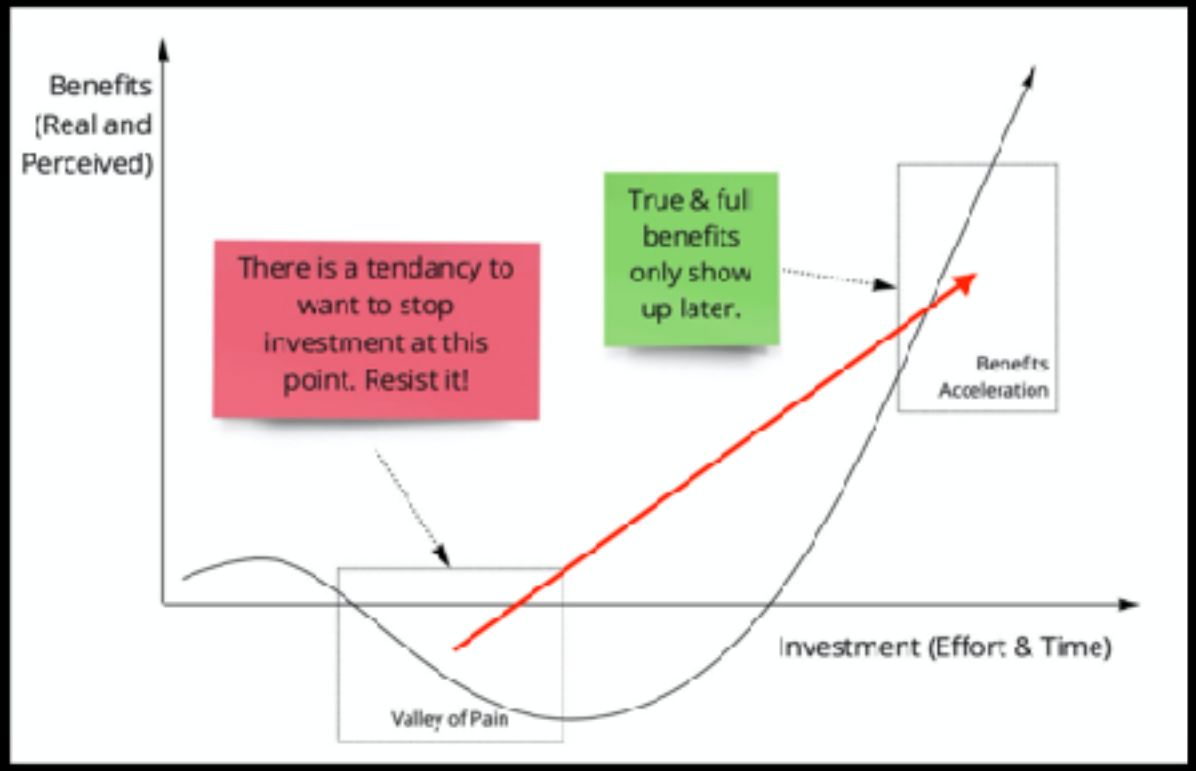
"if it hurts, do it more often", Continuous Delivery



https://martinfowler.com/bliki/FrequencyReducesDifficulty.html

어려운 기술을 배우는 방법

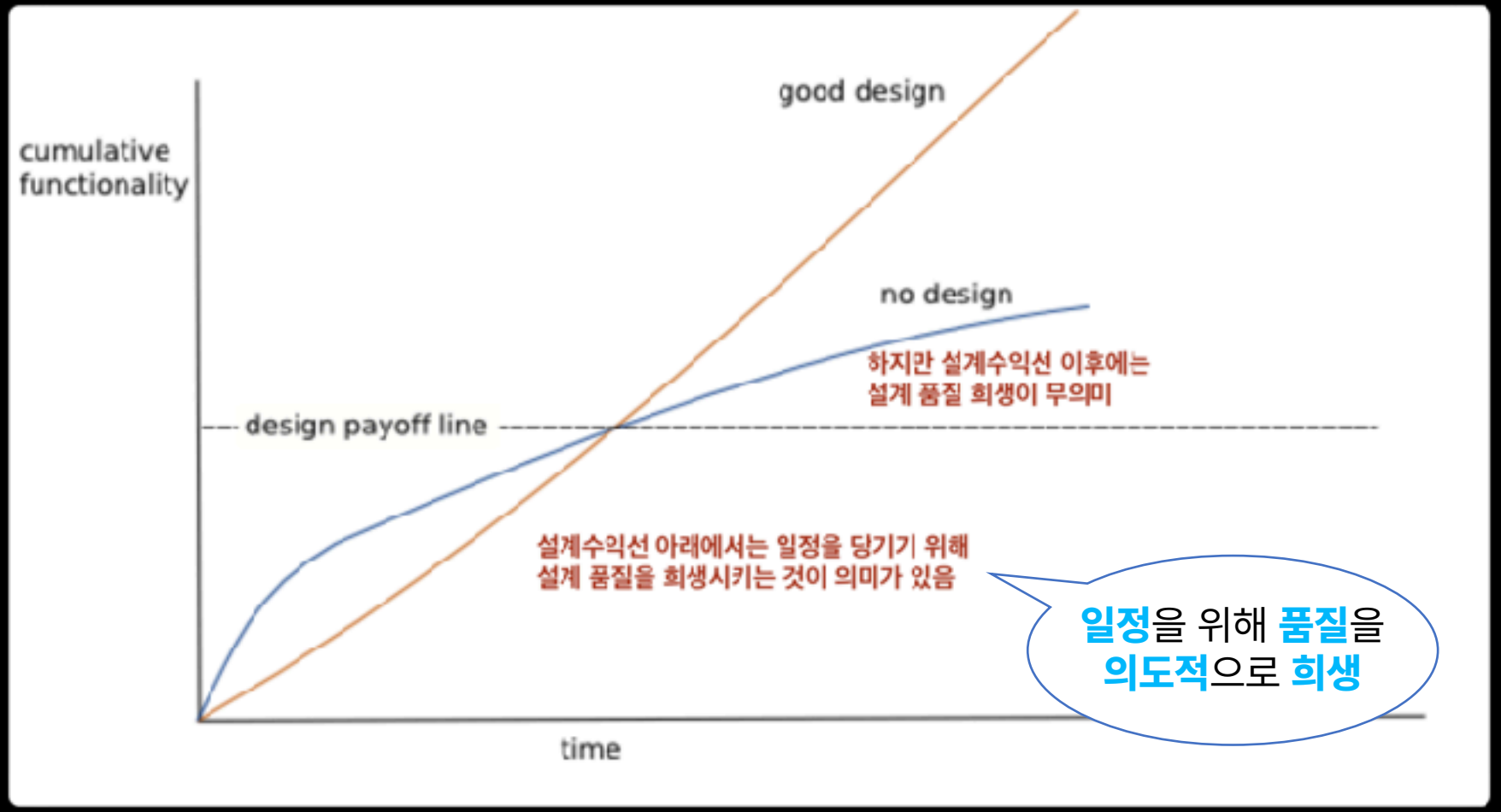
- 유의사항: 계단 성장 - 고통의 계곡



https://www.itworld.co.kr/news/206739

개발

기술 부채



https://martinfowler.com/bliki/DesignStaminaHypothesis.html

기술 부채

- **의도적 희생**
  - 충분히 잘 할 수 있어야
  - 제대로 관리 못하면 시간이 지남에 따라
    - 개발 리소스 ↑ 개발 생산성 ↓
- **췌장암**
  - 아키텍처의 부족은 너무 늦었을 때만 측정할 수 있음

Make it Work, Make it Right

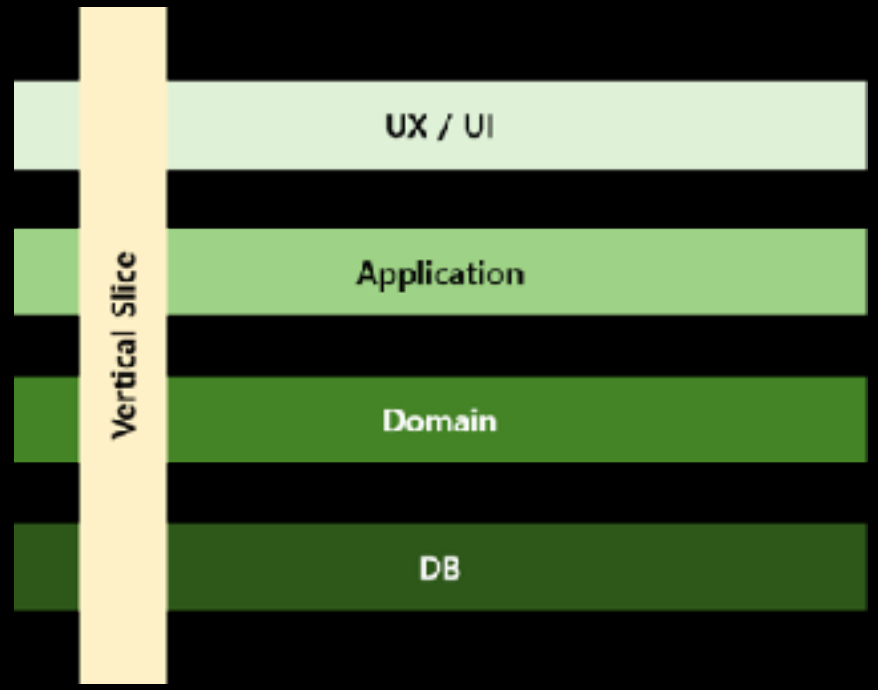
- **SW**의 **2가지 가치**
  - 현재의 요구사항을 만족하는 SW: **행위**
  - 향후 요구사항 변경을 수용할 수 있는 SW: **구조**
- **“Make it Work, Make it Right”**, Kent Beck
- **모순**되는 **우선순위**

Make it Work, Make it Right

- 해결책
  - **동작하게** 한 다음 **반드시 올바른 구조**로
    - **올바른 구조**가 만들어질 때까지는 **다음 작업**을 하지 않음
  - 아주 **작은 단위**로(revert 가능한)
  - "**TDD**의 **R → G → B** 주기의 정신적인 기반"

Make it Work, Make it Right

- **수직 슬라이스**(Vertical Slice)
  - 계층별로 구현 X
  - 한가지 기능씩 E2E로 완벽하게 구현
    - 언제나 배포 가능
  - 복잡해지기 시작하면 **리팩터링**
    - **췌장암**



https://m.blog.naver.com/gwaei324/221506613479

https://www.amazon.com/Growing-Object-Oriented-Software-Guided-Tests/dp/0321503627

코드리뷰

- 목적
  - 버그, 장애 사전 방지
  - 리팩터링, 설계 기법 등을 통해 구성원들의 역량 증대
- **관점의 변화**
  - 수백 명이 있는 개발 조직의 리더

    →

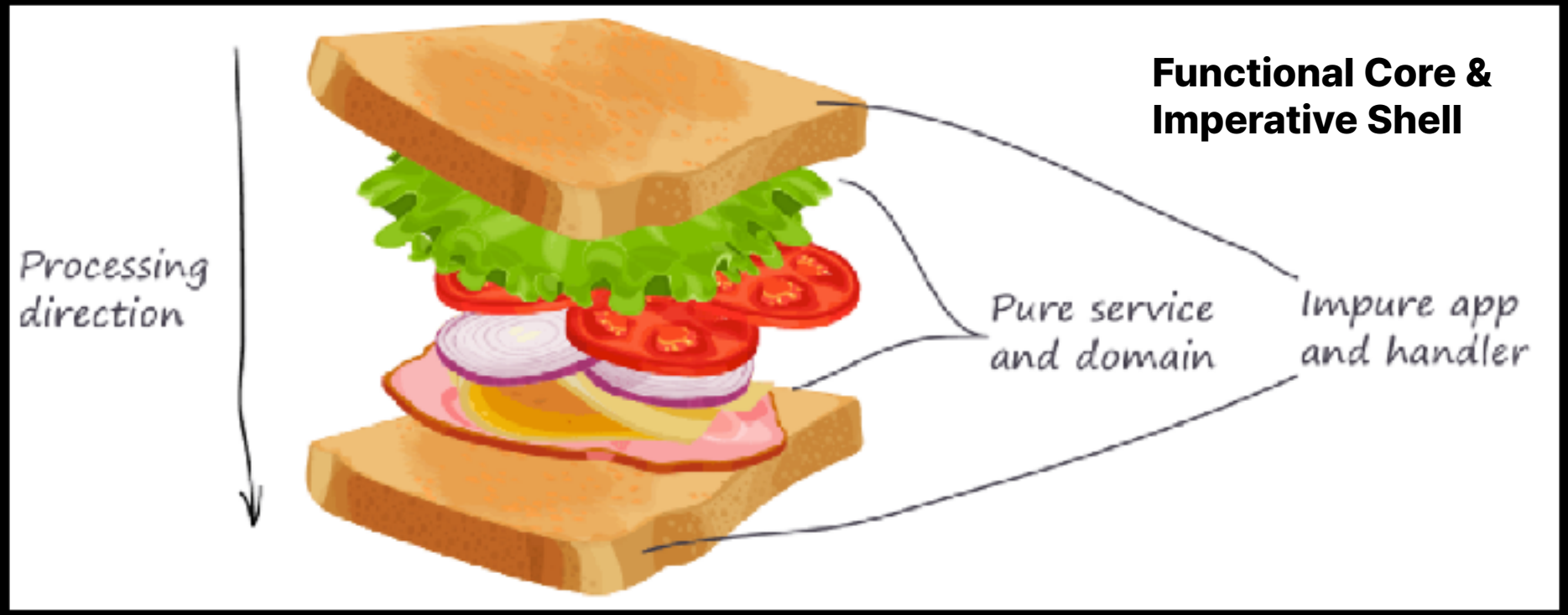
    십여 명이 있는 개발 개발 조직의 리더

코드리뷰

- **가독성**이 최우선
  - 머리 속에 코드가 들어오나
  - Composed Method
  - "Code that fits in your head", Mark Seemann

코드리뷰

- 유지보수 가능성



https://fullsteak.dev/posts/fullstack-rescript-architecture-overview/

코드리뷰

- 이해되지 않으면 거절
  - 항상 거절할 수 있는 단위로 일하기(**매몰비용오류**)
  - 작은 단위로
    - 최대 4시간 이하
    - **하루 2번 리뷰**: 오전, 오후 업무 시작 전

코드리뷰

- **맥락**(Context)
  - 저자와 리뷰어의 코드에 대한 **가장 큰 차이**
    - 저자에겐 있고, 리뷰어에겐 없음
  - 저자도 시간이 지나면
  - 시간이 지나면 결국 남는 것은 **코드** 뿐
    - PR, 코드에 맥락이 남아 있어야

코드리뷰

- 오프라인 리뷰의 어려움 - **맥락**
  - **저자**: 문제가 있는 부분을 **간과**하도록 재빨리 **설득**
  - **시간**을 가지고 **읽어봐야** 의견을 줄 수 있음
  - **속도가 다름**
  - 승인하기 어렵다면 **거절**해야

코드리뷰

- 조기 최적화(Premature Optimization)
  - 맥락이 있는 저자의 조기 리팩터링 ← **YAGNI**
  - 맥락이 없으면 이해가 안됨
  - **구조**를 잘 만들기보다 이해할 수 있는 **가독성**
- 무엇을 잘하고 싶은가 ?
  - 예측을 잘하는 **예언가** ?
  - 빠르게 변화를 수용할 수 있는 **설계를 잘하는 개발자** ?

https://github.com/msbaek/memo/blob/master/refactoring-tech/composed_method.md

일하는 방법

하고 싶은 일을 하려면

- **신뢰 구축 절차**
  1. (더 잘 할수 있어도)**주어진 대로** 잘하기
  2. 개선을 **제안**하기
     - 처음부터 2로 하면(시간↑, 측정X)
     - 1,2를 비교해서 **개선 정도 측정** 가능
  3. (**안 물어보고**)최선의 방법으로 진행

일을 할당하는 방법

- **일에 사람을 할당** or 사람에 일을 할당
  - n가지 일에 n명 할당
    - **공유**가 안됨
    - **대기** 발생. **병목**
    - 우리가 왜 같은 **팀**인가 ?

일을 할당하는 방법

- 일에 사람을 할당 or **사람에 일을 할당**
  - n명에게 일을 할당
    - n개를 한번에 하나씩
    - 지식 **공유**: 팀에 전문성/지식 **축적**
    - **팀웍**: 인간은 사회적 동물
    - **몰입**의 즐거움
    - **대기** 제거

사람/기억력에 의존하지 말아야

- 머리가 좋아서
  - **기억력**에 의존
- **추적 가능성**이 중요
  - 보잉: 30년 생산, 30년 사용
  - github, jira, wiki
  - 작성→**검색** 확인

놀래키지 말기

- 일이 발생한 후에 공유
  - **놀람** → **화**
- 사전에 이슈가 될 만한 일을 공유
  - **대안**도 미리 공유
  - 이슈가 발생해도 **안정적**으로 **대응**

옳고 그름 vs 혁신

- 국민소득 3만불. **혁신**이 필요
- 규모와 복잡성 증가
  - **똑똑한 소수**가 전체를 감당하는 것 불가(**소품종 대량** 생산)
  - 구성원의 **다양성**과 **전문성**을 존중해야(**다품종 소량** 생산)
- **누가 맞는지**가 중요한 세상이 아님
  - 어떻게 하면 **잘할지**가 중요
  - **일이 되는 방향**으로 일하는게 중요한 세상

옳고 그름 vs 혁신

- 혁신을 위한 도전은 **90%가 실패**
  - 하지만 500% 이상의 성장도 가능
  - **작게**, 그리고 **실패**할 만한 도전을 **초기에**
  - 어차피 할 실패라면 **최대한 빨리**(fail fast)
    - **작게, 반복적, 점진적**

중간 계단(stairstep)

- **최종 목표로 한번에** 가는 것
  - **추측**
  - 난이도↑, 실패↑
  - 진전 어려움
- **아기 발걸음**, 반복/점진/개선

중간 계단(stairstep)

- 여러 **중간 계단**을 둬야
- **Needs Driven**
  - 추측에 기반해 하고 싶은 일 수행 →
    할 필요를 만들고 작게 진행
- 중간 계단은 최종 목표에 도달한 후 삭제
  - 절대 낭비가 아님
  - **일이 되게 하는 방법**

성과

- 가치를 제공하는 코드 작성만 의미
  - 해당 코드가 없었을 때 손실도 고려해야(개인정보 보호)
  - **코드리뷰**, **짝프로그래밍**을 통한 기여
- 사티야 나델라
  - **"당신은 다른 사람의 성공에 어떻게 기여했나요?"**

우리팀 에이스는 무엇을 하나 ?

- 고급 + 초급 **짝프로그래밍**
  - 고급 프로그래머는 속도가 **느려짐**
  - 초급 프로그래머는 **남은 인생에 걸쳐** 속도가 올라감
- **몹프로그래밍**
  - 팀장님. **팀원들의 역량**을 빨리 끌어 올리더라
- 여러 사람들이 성과를 낼 수 있게 **도와야**
  - 가동율 100%면 안됨

케이타운포유는 채용 중

- 이커머스 / 물류 개발자
- 데이터 엔지니어
- 프론트엔드 개발자(경력 >= 4년)
- AWS 기반 DevOps