MR3 개발 사례로 Mythical Man-Month 검증하기

SIGPL 여름학교

2019. 8. 27

POSTECH 박성우

The Mythical Man-Month – **맨먼스 미신** (1974**년 출판**)

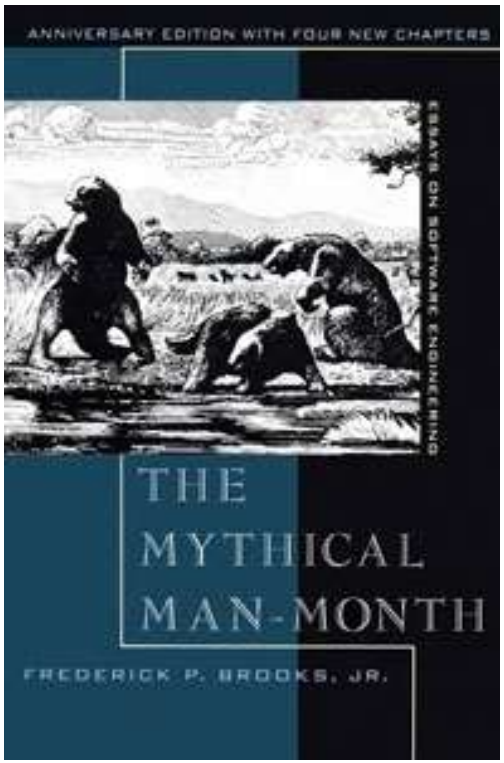
- Book on:
  - software engineering and project management
  - human elements of software engineering
- 소프트웨어 공학의 성경책
  - Everybody quotes it.
    – 소프트웨어쟁이라면 책 내용을 인용하곤 한다.

    "No Silver Bullet"

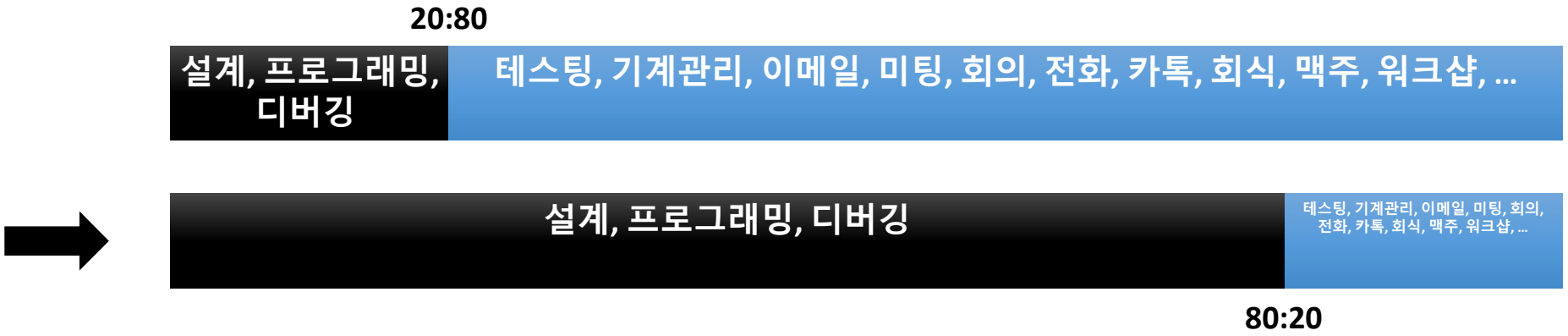
    

  - Some people read it.
    – 소프트웨어쟁이 중 몇은 읽기도 한다.
  - A few people go by it.
    – 소프트웨어쟁이 중 소수만 실천한다.



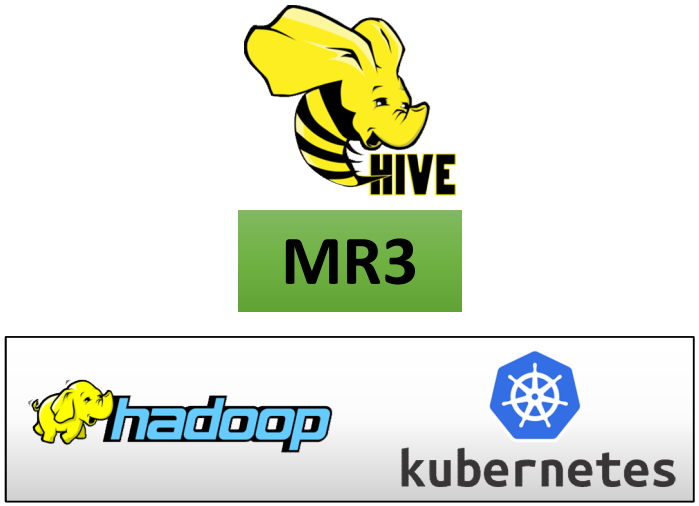
Mythical Man-Month **검증하기**

- 질문
  - “Mythical Man-Month에서 주장하는 내용은 얼마나 사실일까?”
- 검증 방법
  - 실제 프로젝트를 수행하며 Mythical Man-Month 가르침대로 따라해 본다.
- 목표
  - 시간의 80퍼센트를 창의적 활동에 투자한다.
  - 시간의 20퍼센트만 부수적 활동에 투자한다.



MR3 프로젝트

- MR3
  - Hadoop 및 Kubernetes용 계산 엔진
- Hive on MR3
  - MR3를 계산 엔진으로 이용하는 Hive
  - 데이터 웨어하우스 대상 SQL 질의 처리 시스템
- MR3 특징
  - 분산 처리 시스템
    - 부분 부분 설계하고 이해하기는 쉬움
    - 전체 시스템 설계하고 이해하기는 어려움
    - **일관되게 시스템 설계하는 것이 중요**
  - 테스트가 어려움
    - Everything is non-deterministic.
    - Unit test는 거의 쓸모 없음
  - Logging 없이 디버깅 불가능



MR3 **개발 내역**

- 2015년 1월 Java 8으로 개발 시작
  - 주먹구구
- 2015년 7월부터 Scala 언어로 다시 개발
  - 설계부터 다시 시작
  - **성경책 가르침대로 따라하기로 함**
- 2018년 3월 0.1 release
- 2019년 7월 0.9 release

- 현재 4년 2개월간 개발 중

질문

Q. 당신의 프로그래밍 실력은 충분합니까?

Optimism = **프로그래밍에서 만악의 근원** (Root of All Evils)

All programmers are optimists.

So the first false assumption that underlies the scheduling of systems programming is that *all will go well,* i.e., that *each task will take only as long as it "ought" to take.*

- **자신의 optimism 근거가 충분하지 않음을 받아들이기 어려움**
- 팀원 능력에 optimism을 가지기도 어려울 수 있음

- 근거 없는 optimism을 없애면:
  - 아무리 노력해도 **(특히 첫 번째)** 설계와 구현에는 문제가 있음을 받아들이게 됨
  - 결국 설계 단계에 시간을 더 많이 쓰게 됨으로써 **전체 작업 시간은 줄이게 됨**

코딩에 필요한 시간, 테스팅에 필요한 시간

- Optimism 때문에 테스팅에 시간을 적게 할애함

> Because of optimism, we usually expect the number of bugs to be smaller than it turns out to be. Therefore testing is usually the most mis-scheduled part of programming.

- 성경책에서 추천하는 시간 배분

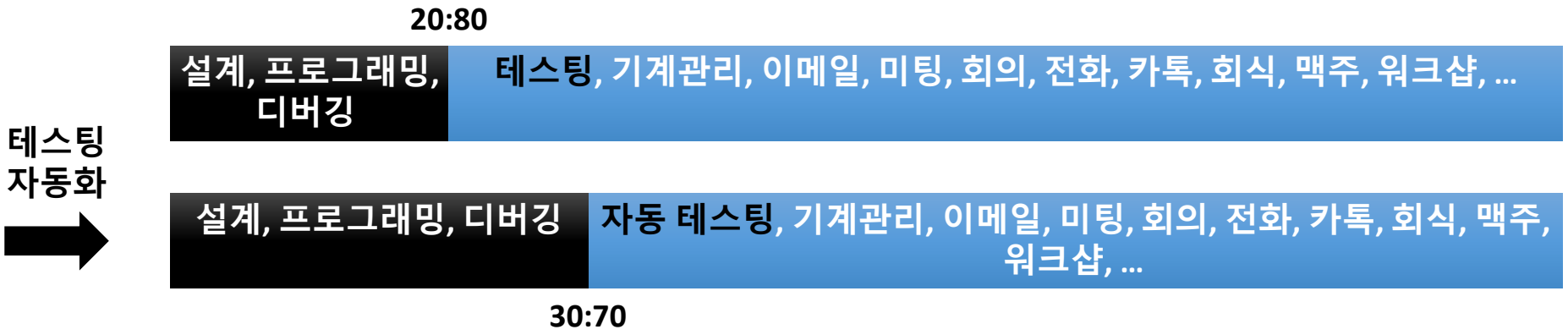
> ⅓ planning
> ⅙ coding
> ¼ component test and early system test
> ¼ system test, all components in hand.

테스팅과 디버깅에 절반 할애

- Optimism이 남아 있을 때 잘 맞는 규칙
  - **프로그래밍 실제 작업 시간 = 예상 시간 * 2배 * 25%**

테스팅 시간 줄이기

- 테스팅 문제 자체는 줄일 수 없음 (줄여서도 안됨)
- 그러나 테스팅에 실제 할애하는 시간은 줄일 수 있음
  - 테스팅 자동화
  - 작업 병렬화



테스팅 자동화 예

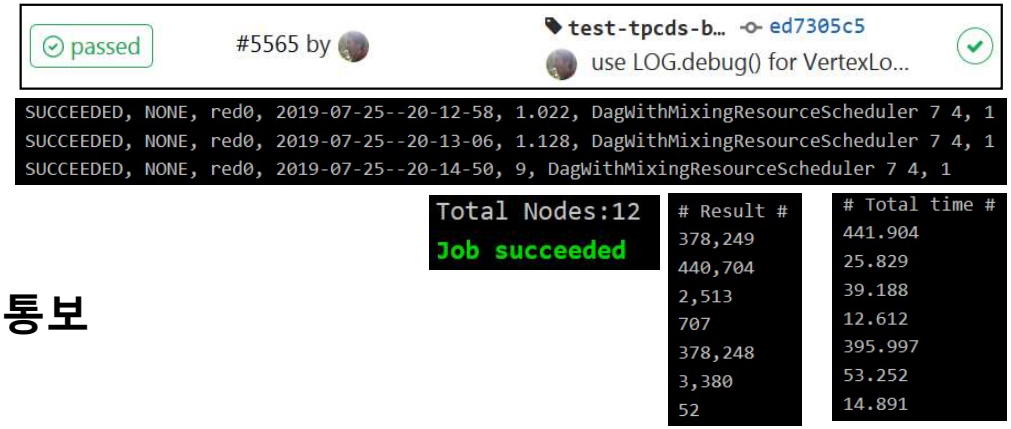
1. git으로 tag push

```
test-tpcds-indigo-amprocess-crossdag-beeline1-hivesrc5-basememory8192-combine10-querystart11-queryend11-kill80-numrun5
```

2. 실험 setting 기록, 실험 수행, 진행 기록

```
Output Directory:
/home/gitlab-runner/mr3-run/hive/hiveserver2-service-result/hive-mr3-4f74269-2019-07-29--12-08-04-b2c68aef
```

3. 실험 결과 분석 및 저장



4. Slack으로 통보

Grey (1) | PLvi (1)

Indigo (21)

Gold (42)

Red (11)

Blue (13)

White (4)

Navy (4)

질문

소프트웨어 프로젝트에서 가장 중요한 자원은 ___이다.

답: 사람???

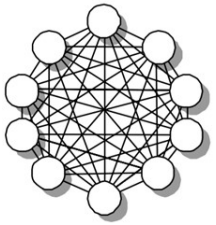
Brooks's Law: *Adding manpower to a late software project makes it later.*

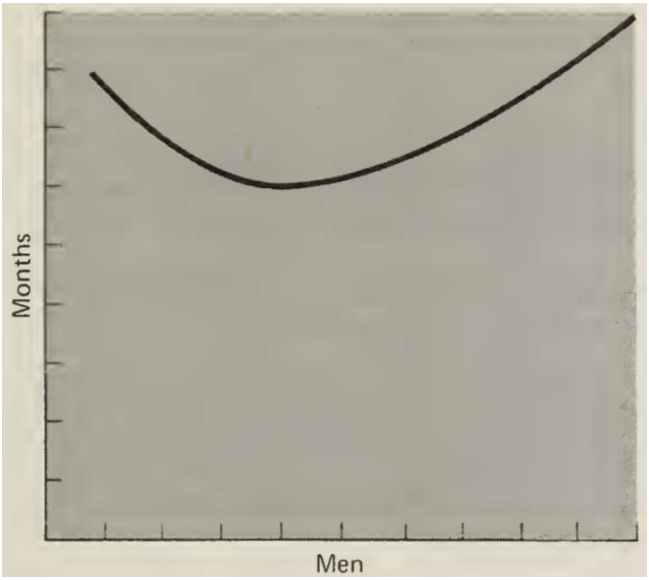
- Man-month 개념은 '**No communication among workers**'일 때만 통함
- 소프트웨어 프로젝트에서는 통하지 않음

The added burden of communication is made up of two parts, training and intercommunication. Each worker must be trained in

This training cannot be partitioned,

Intercommunication is worse. If each part of the task must be separately coordinated with each other part, the effort increases as $n(n-1)/2$. Three workers require three times as much pairwise





- **결론: 개발 프로세스에서 최우선 목표 = Communication Overhead 줄이기**

Communication Overhead **줄이기**

- **1. 프로젝트 내용 문서화**
  - 1. 주별 목표
  - 2. 일별 progress/solution/decision/question/TODO/실험 결과/발견/…
    - 자기와 팀원에 도움이 될 내용
    - **일종의 technical writing**
      - Title = Topic sentence, Body = Paragraph
  - 3. 설계 문서
- **2. 문서 실시간 공유**
  - Dropbox 이용
  - Benign distraction -- 팀원 문서 보는 즐거움
- **3. Interrupt 최소화**
  - 미팅은 만나서 얘기하는게 나을 때만
  - **그러나 미팅을 하면 가능한 길게 (코드 리뷰, 설계 리뷰 등)**
  - 이메일은 부득이한 경우에만
  - 전화는 쓰지 않음

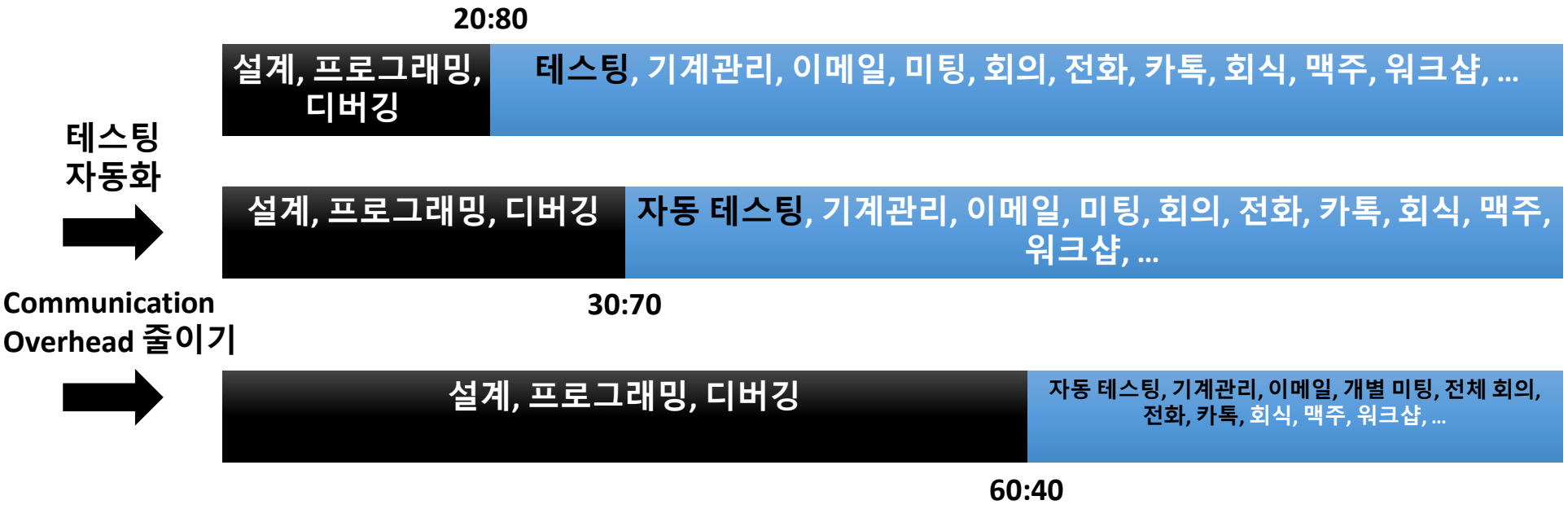
- **결과: 시간의 대부분을 집중해서 일할 수 있음**

자신과의 Communication Overhead **줄이기**

- 자신:
  - **1. with myself**
  - **2. with ourselves**
- 추후 **시간**을 절약해 줄 **수 있는** 내용은 모두 문서화
  - 설계/구현에서 왜 이런 결정을 내렸는가
  - 버그를 어떻게 고쳤는가
  - 소프트웨어/OS/기계/클러스터 설치는 어떻게 했는가
  - 고장난 기계를 어떻게 고쳤는가
  - …

- **결과: 창의적이지 않고 할 줄 아는 일에 다시 쓰는 시간 최소화**

'Communication Overhead 줄이기' 결과



지금 생각

소프트웨어 프로젝트에서 가장 중요한 자원은 시간이다.

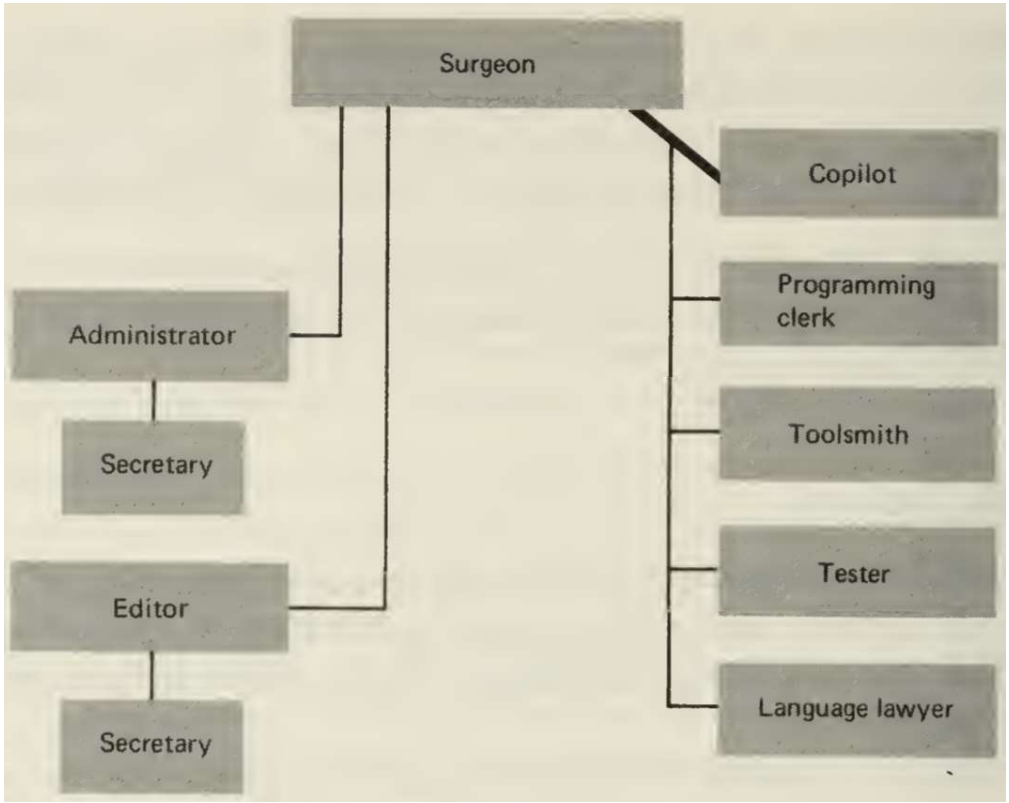
질문

소프트웨어 개발할 때 중요한 결정은 _____ 방식을 따라야 한다.

답: 민주주의???

Surgical Team -- **팀 구성과 역할 분담은 어떻게 해야 하나**?

- Surgeon: Chief Programmer로서 모든 최종 결정 내림



⬅ **반드시 필요**

⬅ **문서화 통해서 시간 최소화**

⬅ **테스트 코드 직접 작성**
테스트 자동화

Surgical Team **모델은 일종의 문화**

- 팀 멤버가 모두 Surgical Team 철학을 이해하고 따라야 함
  - 소프트웨어 개발에서 민주주의는 치명적임
- Surgeon의 판단 능력이 결정적으로 중요함
- 일관적 설계를 위해서도 절대로 필요함

> user's interest. If a system is to have conceptual integrity, someone must control the concepts. That is an aristocracy that needs no apology.

내가 결정한데로 하라우

➔ No Communication Overhead

팀원들 의견대로 진행하자!

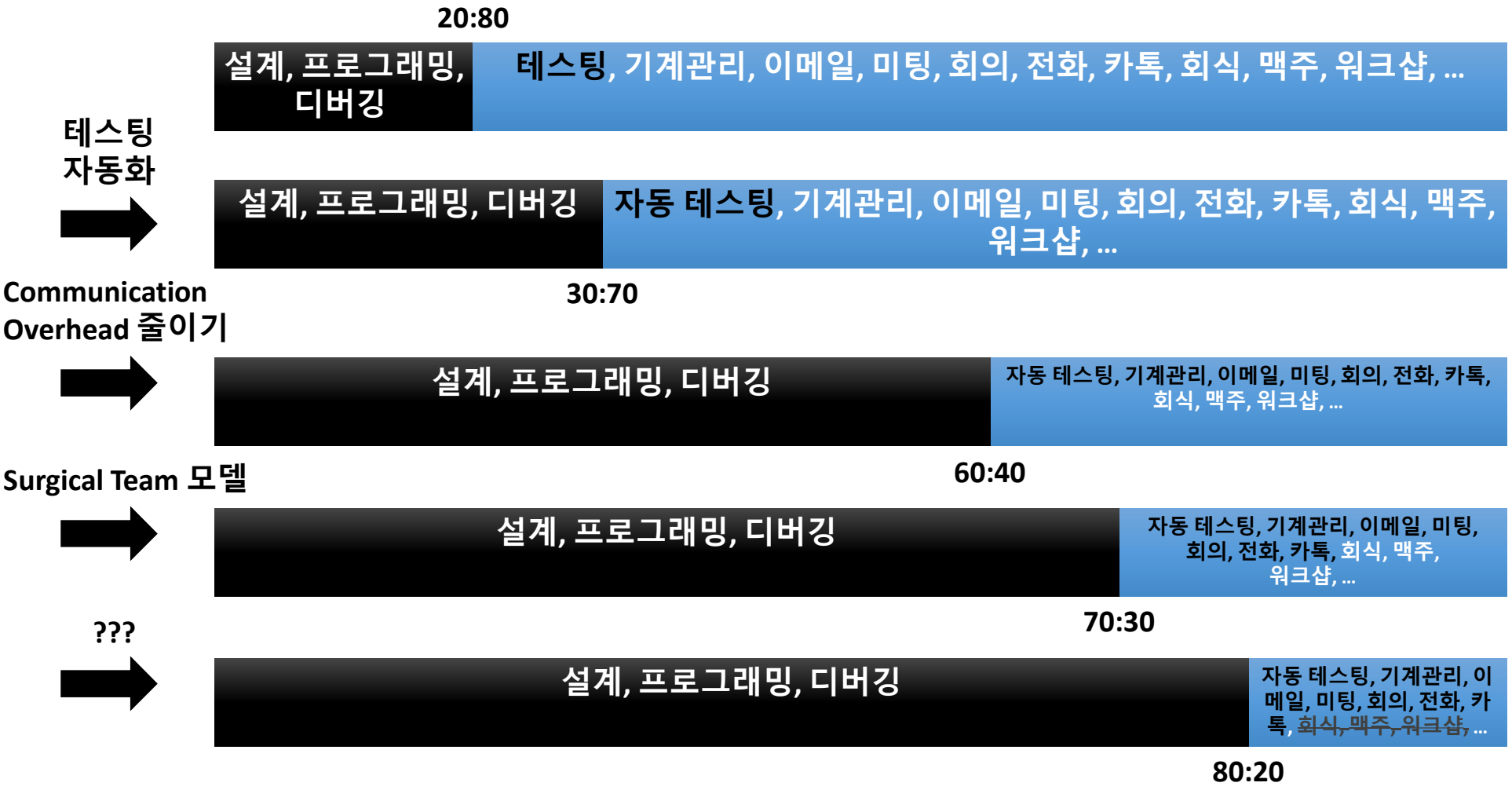
➔ Communication Overhead Hell

Scaling Up: **사람이 100명 이상이면?**

- 예: Amazon 모델
  - 팀 크기 최소화
    - 5명 - 8명 정도 엔지니어로 구성
  - 팀 내에서 모든 결정 및 구현
    - design, build, test, deploy
    - 별도의 테스트 팀이 없음
  - 팀간 coordination problem 최소화

- **결론: 팀끼리도 Communication Overhead를 최소화 해야 함**

Surgical Team 모델 결과

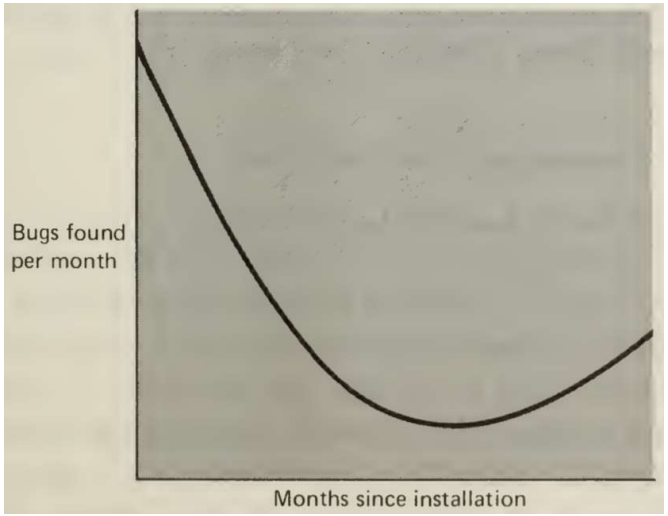


질문

함수형 프로그래밍이 중요한 이유는 _____ 때문이다.

5년 전 답: (멋있고) 간결하기 ...

소프트웨어 출시 후 Bug



⬅ 고치기 매우 어려운 Bug

- Bug-fix 때문에 Bug 생김:

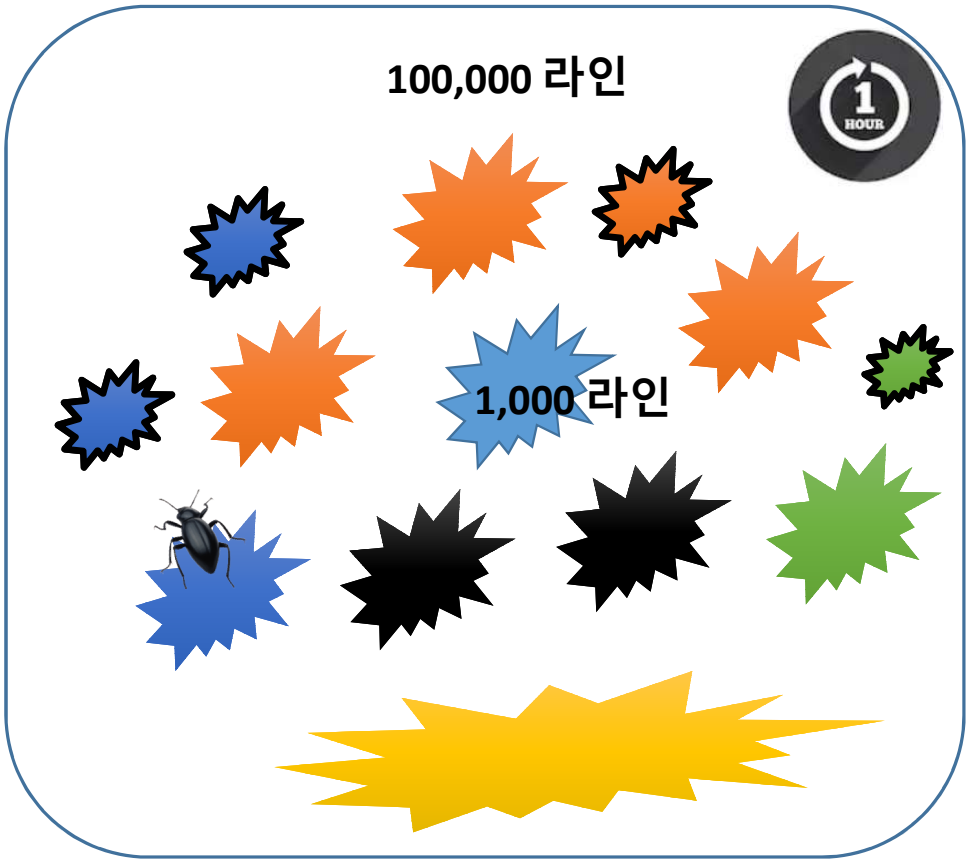
> The fundamental problem with program maintenance is that fixing a defect has a substantial (20–50 percent) chance of introducing another. So the whole process is two steps forward and one

- 함수형 프로그래밍 강점:

> Clearly, methods of designing programs so as to eliminate or at least illuminate side effects can have an immense payoff in maintenance costs. So can methods of implementing designs with

함수형 스타일 프로그래밍이 왜 중요한가?

명령형 스타일



함수형 스타일

지금 생각

함수형 프로그래밍이 중요한 이유는
시간을 줄여주기 때문이다.

예전 생각, 지금 생각

복잡한 시스템을 **구현**해 내는 것이
진짜 능력이다.

➔ 시스템을 **간단하게 설계**하는 것이
진짜 능력이다.

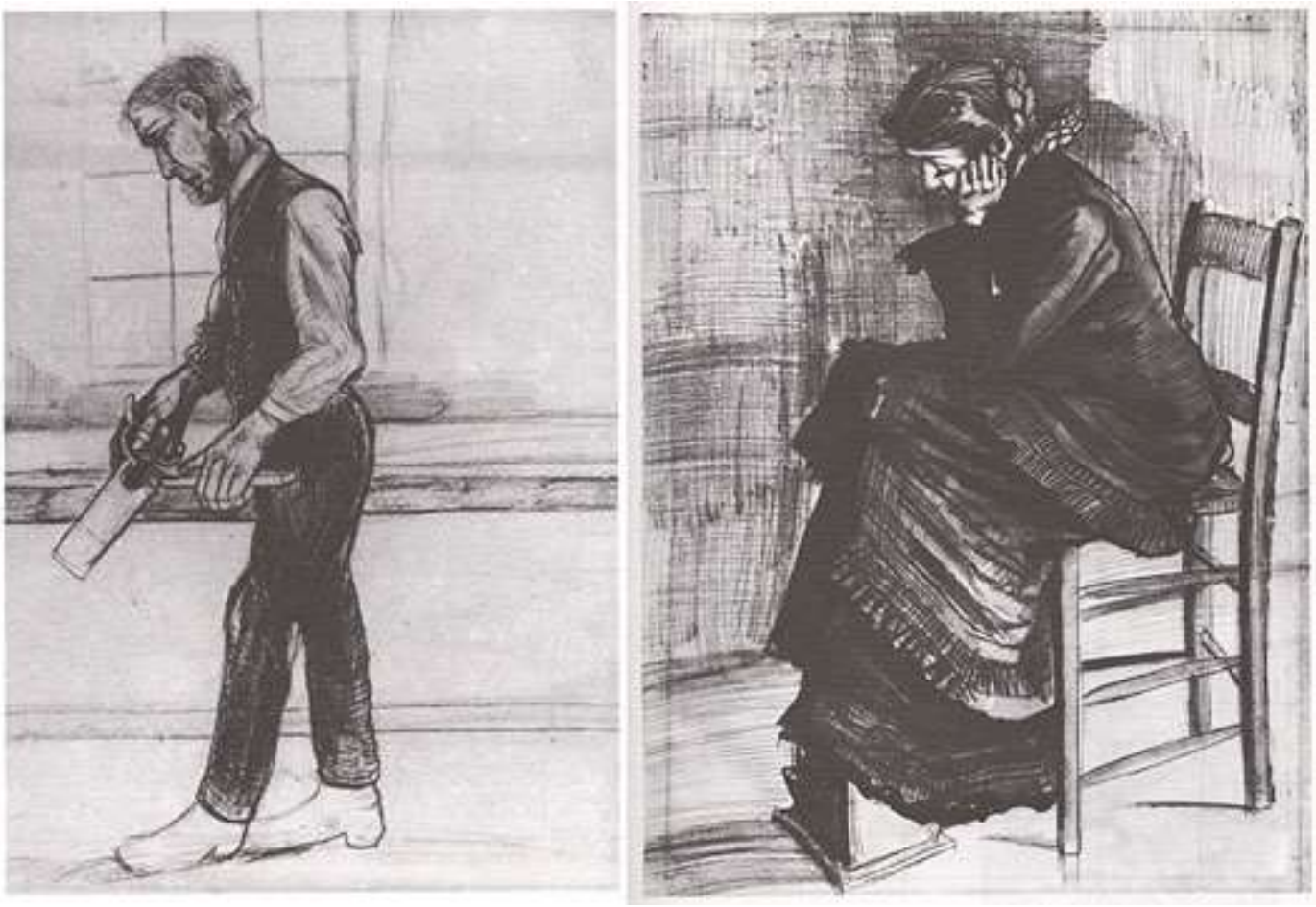
프로그래밍에서 '설계하기 = 꿰뚫어 보기'

- **Designing** is **Seeing** in the **Programming Dimension**.

소프트웨어 설계 능력은
설계가 복잡함을 간파하는 능력이다.

Out of the Tar Pit (Ben Moseley and Peter Marks, 2006)

Simplicity is *Hard*



Vincent Van Gogh Carpenter, 1880 and Woman Mourning, 1882

예전 생각, 지금 생각

코딩 실력이 제일 중요하다.

➔ **Communication 능력이 제일 중요하다.**
(Technical Writing, Presentation)

예전 생각, 지금 생각

경험 많은 개발자가 좋다.

➔ 경험은 상관없이 빠릿빠릿한 개발자가 좋다.
(빠릿빠릿 = 명쾌한 소통 능력 + 빠른 습득 능력)

예전 생각, 지금 생각

코드 새로 짜는게 어렵다.

➔ 있는 코드 읽는게 어렵다.

예전 생각, 지금 생각

주석 달린 코드가 좋다.

➔ 주석 없는 코드가 좋다.
(예외: Why를 설명한 주석)

감사합니다

Q & A