K DEVCON

프로그래머는 겸손한가?

생즉고(生卽苦)

삶은 괴로움이다.

“어떤 사람은 자기의 문제와 고통스러운 것을 피해 쉬운 길을 찾으려다가 오히려 건전하고 지각 있는 길에서 아주 멀리 벗어나게 된다.” – 스캇 펙

발표자 소개

중 2 (1982년) 때 코딩을 시작해서 컴퓨터를 중심으로 삶을 엮어가는 사람으로 소프트웨어와 데이터 기술을 접목해서 현실의 문제를 해결하는 것을 좋아합니다.

- **컬리, 풀필먼트 프로덕트 본부장 겸 데이터 그룹장**
- 전 우아한형제들, 딜리버리 플랫폼 실장 (배달 플랫폼, 배민 커넥트, B마트 담당)
- 전 SK 플래닛, 데이터 플랫폼 본부장
- 전 Gen128, 젠장(대표이사)
- 전 한국 스프링 사용자 모임 2대 큰일꾼 (현 고문)

프로그래머는 겸손한가?

50여년 전 대선배의 목소리에 귀 기울여 보기

마중물

“겸손한 프로그래머”

겸손한 프로그래머

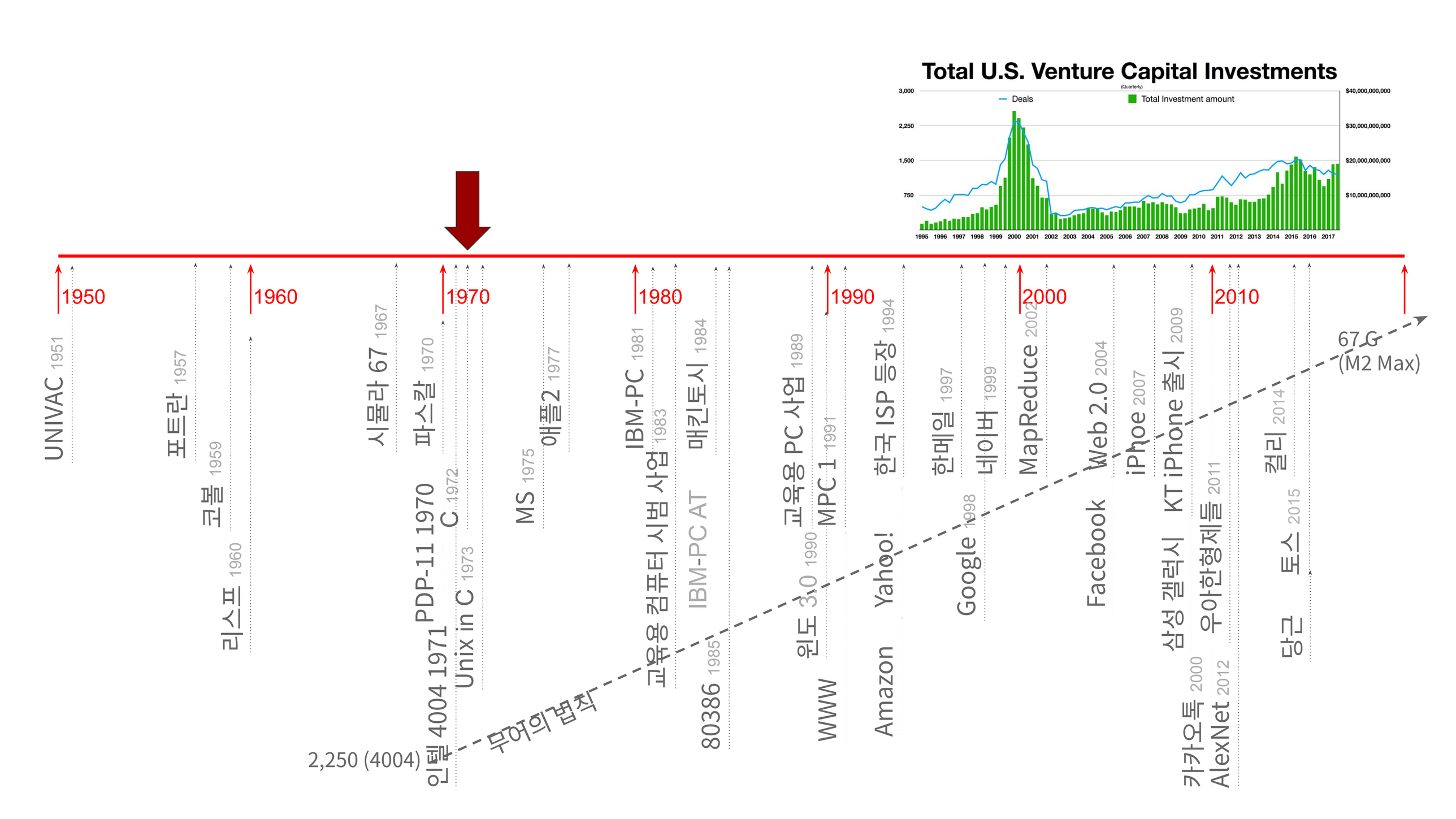
- 에츠허르 데이크스트라의 1972년 제 7회 튜링상 수락 연설
  - 구조적 프로그래밍(“Go To 문의 해로움Go To Statement Considered Harmful”)
  - 주류 고급 프로그래밍 언어의 모태인 ALGOL 개발에 기여
  - 그래프 이론을 포함한 컴퓨터 과학 구축에 폭 넓게 공헌
- 인간의 지적 한계를 적시하고 그에 비해 지나치게 도전적인 프로그래밍이라는 작업의 문제를 “위기”로 선언, 해결 방안 제시

그래서? 왜?

- 지금 우리 모습을 이해하고
- 어느 방향으로 나아가야 할지 논의하기 위해서

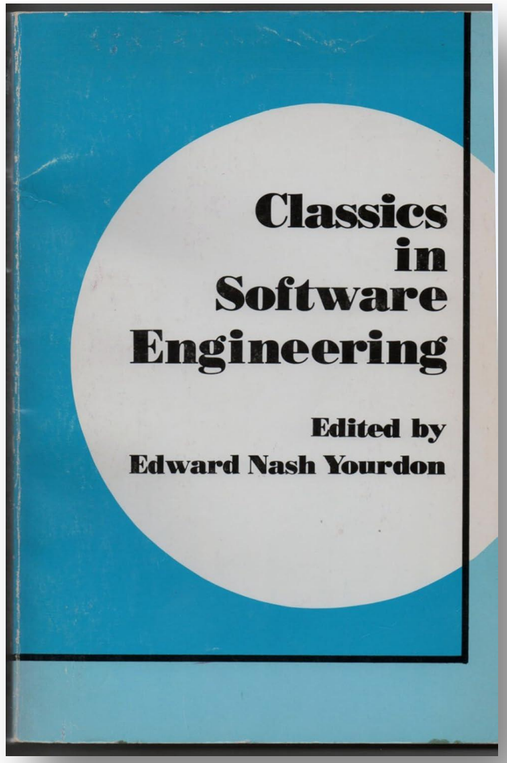
겸손한 프로그래머의 구성

- 당시까지의 컴퓨터 역사를 개인 이야기와 함께 정리하며 얼마나 컴퓨터 과학이 미숙한 수준인지 고발
- 하드웨어 중심 문화와 기계의 제약을 극복하는 지적 만족감에 천착하는 프로그래머
- 프로그래밍하기 어려워지는 하드웨어 발전 방향과 컴퓨터의 보급과 함께 점차 과도해지는 사용자의 요구
- 프로그래밍의 본질적인 어려움
- 과거의 다양한 시도와 평가
- 고품질의 저비용 프로그래밍 기술을 향한 비전 제시와 기대
- 비전 달성을 위한 사회적 조건 세 가지
- 여섯 가지 비전 달성 방안


Total U.S. Venture Capital Investments
(Quarterly)
Deals
Total Investment amount
3,000
2,250
1,500
750
$40,000,000,000
$30,000,000,000
$20,000,000,000
$10,000,000,000
1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017
1950
1960
1970
1980
1990
2000
2010
UNIVAC 1951
포트란 1957
코볼 1959
리스프 1960
시물라 67 1967
파스칼 1970
PDP-11 1970
C 1972
Unix in C 1973
MS 1975
애플2 1977
IBM-PC 1981
교육용 컴퓨터 시범 사업 1983
매킨토시 1984
IBM-PC AT 80386 1985
교육용 PC 사업 1989
윈도 3.0 1990
MPC 1 1991
WWW
한국 ISP 등장 1994
Amazon
Yahoo!
한메일 1997
Google 1998
네이버 1999
MapReduce 2002
Facebook
Web 2.0 2004
iPhoe 2007
삼성 갤럭시
KT iPhone 출시 2009
카카오톡 2000
AlexNet 2012
우아한형제들 2011
컬리 2014
당근
토스 2015
인텔 4004 1971
2,250 (4004)
무어의 법칙
67 G
(M2 Max)

겸손한 프로그래머의 시대 배경

- 최초의 폰 노이만 아키텍처 컴퓨터인 EDSAC이 출시된지 24년 후
- 3세대 컴퓨터의 폭발적인 보급(PDP-11, System/370)
  - Unix 탄생(개발 1971/11, 발표 1973/10)
- 고급 프로그래밍 언어 ALGOL 연구와 다양한 언어 탄생(C, 파스칼, 시뮬라 등)
- 소프트웨어 위기, 나토 소프트웨어 학회(1968, 1969)
- 구조적 프로그래밍 논의
  - "인간 활동으로 여겨지는 프로그래밍Programming Considered as a Human Activity", 1965
  - “Go To 문의 해로움Go To Statement Considered Harmful”, 1968
  - “구조적 프로그래밍 논문집Notes on Structured Programming”, 1970



“겸손한 프로그래머”

글의 구성

01

현실 진단

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

02

소프트웨어

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

03

비전 제시

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

현실 진단

'

응? 뭐라고? 프로그래머? 이걸 직업이라고 할 수 있나?
무엇보다 프로그래밍이 뭐지?
지적으로 존중할 만한 학문임을 뒷받침하는 지식 체계가 있기는 한 건가?'
이런 질문에 대해 나는 빈손으로 서 있는 느낌이었다.

'

프로그래머가 된 개인적 경위

- 뒤늦게 도입된 컴퓨터, 최초의 네덜란드 직업 프로그래머가 됨 (1952)
- 이론 물리학 전공과 프로그래밍 병행(수학 센터) 중 선택
- 상사의 조언

  '컴퓨터는 잠깐 왔다 사라질 게 아니고 단지 시작 점에 있을 뿐이네. 자네가 앞으로 프로그래밍을 존경 받을 만한 분야로 만드는 사람 중 하나가 될 것이네.'
- 결혼식을 하면서 정부에서 프로그래머란 직업을 인정하지 않음
- 대부분의 나라가 시기만 조금 다를 뿐 비슷한 상황

하드웨어에 묶인 프로그래머

- 워낙 도전적인 기기어서 동작한다는 것 자체가 대단한 1세대 컴퓨터
- 전 세계에 한 대 뿐인 컴퓨터를 정상 동작 상태로 유지하는 것이 최우선 과제
- 미국 컴퓨터 학회(ACM Association for Computing Machinery) 이름에 기계가 언급된 이유
- 존재감 없는 프로그래머
  - 기계를 개발한 곳이 곧 기계를 사용하는 장소
  - 거대하고 황홀한 기계 앞에서 보잘것 없어 보이는 프로그래머의 작업
  - 프로그래머 스스로 기계 덕에 자신의 일이 의미 있다고 생각
  - **성능이 빈약한 컴퓨터의 한계를 극복하는 기교에서 지적 만족감을 얻음**

프로그래밍에 대한 두 가지 견해

1. 유능한 프로그래머라면 복잡하게 꼬인 문제를 즐기고 교묘한 기교를 좋아해야 한다.
2. 프로그래밍이란 어떤 식으로든 계산 절차의 효율성을 최적화하는 일에 불과하다.
    - 더 강력한 기계가 나오면 해결될 문제
    - 프로그래밍은 별 어려운 일이 아닌 것이 될 것이라는 전망
    - 그 사이 10^n배 더 강력한 기계가 나왔지만 여전히 어려움에 허우적거리는 프로그래머, 왜?

소프트웨어

'

심하게 말해, 컴퓨터가 없었을 때는 프로그래밍에는 전혀 문제가 없었다.
느린 컴퓨터 몇 개 뿐이었을 때는 프로그래밍이 조금 문제가 되었고,
이제는 거대한 컴퓨터에 프로그래밍도 비슷하게 거대한 문제가 되었다.

'

소프트웨어 위기

1. 어떻게 사용할지 무관심한 채 천 배 이상 강력한 기계를 만든 무책임한 전자 업계
2. 천 배 이상 강해진 사회적 야망
3. 목적과 수단의 긴장이 폭발하는 영역에 놓인 불쌍한 프로그래머
4. 선각자들이 이미 경고한 “소프트웨어 위기", 수 년 뒤에나 알게된 업계
5. 보급을 목표로 만들어진 낮은 가격의 제 3세대 컴퓨터
    - 그 가격이 하드웨어에 국한되었다는 것이 문제
    - 많이 보급되는 만큼 견고한 설계가 필요. 하지만, 설계상 심각한 결점으로 컴퓨터 과학의 발전을 가로 막음
    - 3세대 컴퓨터의 엄청난 성공으로 결함 많은 설계가 당연한 것으로 여겨짐
6. 하드웨어가 만들어낸 거대한 문제를 해결하도록 내던져진 프로그래머

소프트웨어 영역의 시도와 평가

- 서브루틴 라이브러리
  - EDSAC에서 등장, 폐쇄 서브루틴, 최고의 소프트웨어 발명품 중 하나
  - 추상화의 기본 패턴 중 하나가 구현 가능해짐, 앞으로도 살아남을 개념
- 포트란
  - 대담하고 존경 받을만한 프로젝트
  - 뒤늦게 결점이 발견되었으나 성공적인 코딩 기법으로 평가해야 함
  - 사고 전달 수단으로 적절하지 않아서 이제는 빨리 잊어야 할 과거
  - 두뇌의 힘을 낭비하고 위험이 크고 사용이 비싼 반면 너무 많이 사용되고 있음
- 리스프(Lisp)
  - 전혀 다른 흥미로운 프로젝트, 뛰어난 안정성 제공
  - “컴퓨터를 오용하는 가장 지능적인 방법"이란 평가, 대단한 찬사로서 사고의 자유를 의미
  - 언어의 의미와 동작 방식이 기묘하게 혼합된 형태

알골 60

- 프로그래밍 언어를 구현과 분리된 방식으로 정의하려는 진정한 노력의 결실
- 성공적으로 분리된 나머지 구현이 가능할지 걱정이 될 정도
- 알골 정의 문서(“Report on the Algorithmic Language ALGOL 60”)는 BNF 표기법 덕에 짧으면서도 학계에 심오한 영향을 미치고 알골이란 이름을 유행하게 만듬
- 알골 60의 패러미터 구조는 지나치게 많은 조합을 허용, 강한 절제가 필요

PL/I

- 끔찍하게 길고 복잡한 언어 정의 문서
- 조종실에서 버튼, 스위치, 핸들이 7,000 개 달린 비행기를 조작하는 느낌
- 기본 도구인 프로그래밍 언어가 그 자체로 인간의 지적 통제를 벗어나서 프로그램을 지적 한계 내로 붙잡을 방법이 없음
- PL/I 옹호자는 이미 많은 기능에 끝 없이 더 많은 기능을 요구하는 약물 중독자
- 포트란은 유아기 장애, 악성 종양의 증식 특징을 가진 PL/I은 치명적인 질병

비전 제시

실수에서 배운 교훈

- 프로그래밍은 조만간 완전히 다른 활동이 되어 있을 것, 충격적인 큰 변화
  - 전망 1: 몇 %의 비용으로도 시스템을 프로그래밍 할 수 있을 것
  - 전망 2: 시스템의 버그가 거의 없을 것
- 소프트웨어에서 비용과 품질은 상호 보완적인 관계, 어느 한 쪽을 추구하면 자연스럽게 다른 쪽을 달성하게 됨

혁명을 가능하게 만들 세 가지 조건

1. 사회 전반적인 변화 필요성 인정
   - 더 높은 신뢰성의 소프트웨어가 필요하다는 인식은 이견이 없을 것으로 예상
   - 과거에는 “소프트웨어 위기"를 말하는 것이 신성 모독 수준의 금기
   - 나토 소프트웨어 공학 학회를 계기로 인식 전환 발생
2. 충분히 강력한 경제적 필요성
   - 60년대 프로그래머 보수가 너무 많았고 앞으로 낮아질 것이라는 전망이 흔함
   - 사회적으로 프로그래머의 성과와 제품의 성능에 불만을 느끼고 있음
   - 하드웨어 가격이 떨어지면 소프트웨어가 비싸다고 느껴 사회적인 거절이 발생할 것
3. 가능하게 만들 기술
   - 가능할 것이라고 예상하며 여섯가지 논거 제시

지적 관리성intellectual manageability

- 프로그램의 종류와 구현마다 지적 관리성이 천차만별일 수 있음이 밝혀짐
- 지적 관리성을 지키는 규칙이 밝혔고, 이것을 깰 때 지적 관리성이 훼손됨
- 규칙은 두 가지 종류로 나뉠 수 있음
  - 첫 번째 종류의 규칙은 기계적으로 쉽게 규제할 수 있음, 적절한 프로그래밍 언어의 선택으로 규제 가능
  - 두 번째 종류의 규칙은 아직 (어쩌면 영원히) 규제 방법이 없어서 프로그래머가 스스로 준수하려고 해야 함

이제 **지적으로 관리 가능한 프로그램의 설계와 구현에만 우리 스스로를 한정할 것을 제안**한다. 이 제한이 너무 심해서 감당할 수 없다고 우려하는 사람이 있다면, 안심해도 된다. 지적으로 관리 가능한 프로그램의 범주는 알고리즘으로 풀 수 있는 모든 문제에 대해 매우 현실적인 프로그램을 포함할 만큼 충분히 풍부하기 때문이다.

6가지 논거

논거 1

- 프로그래머가 지적으로 관리 가능한 프로그램만 고려하면 되므로 선택하는 대안이 다루기 훨씬 쉽다

논거 2

- 지적 관리 가능한 프로그램의 하위 집합으로 제한하기로 결정하는 즉시, 고려해야 할 해법의 범위가 단번에 대폭 줄어든다

논거 3

- 프로그램 정확성correctness 문제에 관한 건설적인 접근 방법
- 일반적인 방법은 프로그램을 만들고 후에 테스트를 수행, 버그가 있음을 보여주는 효과적인 방법
- 버그가 없음을 보여주는 데는 전혀 적절하지 않음
- 정확성에 대한 설득력 있는 증명을 제공해야 함
- 프로그램 제작 후 증명은 부담만 증가시킬 뿐
- “프로그래머는 정확성 증명과 프로그램을 함께 자라나게 해야 한다"
  - 타당한 증명의 구조를 찾고
  - 증명 요건을 충족하는 프로그램을 구축하는 것을 반복
- 우리 범위를 지적으로 관리용이한 프로그램으로 제한하는 경우에는 적용 가능

논거 4

- 프로그램을 설계하는데 필요한 지적 노력의 양은 프로그램 길이에 따라 달라짐
- **추상화**: 매우 유한한 추론으로 무수히 많은 경우를 다룰 수 있는 유일한 정신적 도구
- 유능한 프로그래머의 가장 중요한 활동 중 하나가 추상화 능력을 활용하는 것이어야
- 추상화의 목적은 절대적으로 명확한 새로운 의미 수준을 만드는 것
- 프로그램이 길이에 따라 구상과 이해하는 지적 노력이 비례해서 커지지 않을 수 있음
- 추상화 패턴이라는 부산물의 발견, 프로그래밍 시간을 대폭으로 단축시킬 것

'

유능한 프로그래머는 자기 두개골 크기가 매우 제한적이라는 것을 충분히 인식하고 있어서 프로그래밍 작업에 최대한 겸손하게 접근하고 무엇보다도 전염병과 같은 영리한 트릭을 피한다.

'

논거 5

- 도구, 언어 , 표기법이 생각하거나 표현할 수 있는 것을 결정짓는 주요 요인
- 프로그래밍 언어를 비교하는 새로운 척도
  1. 프로그래밍 언어가 사고 습관에 미치는 영향 분석
  2. 두뇌 능력이 가장 부족한 자원이라는 인식
- 똑똑하다고 자랑하는 사람들이 빠지는, 전염병 같은 교묘한 트릭
  - 경쟁적으로 더 적은 코드로 같은 동작을 하는 코드 작성
  - 암호 같은 코드를 보여주며 알아 맞춰 보라고 함
- “더 풍부한” 또는 “더 강력한” 프로그래밍 언어의 개발은 실수라는 것이 과거의 교훈
- for 문도 바로크적, 반복문이 사고를 제한했던 경험
- 매우 체계적이고 소박하고 추상화가 자연스러운 프로그래밍 언어가 유망할 것

논거 6(주장)

- 잘 팩토링된 해결책의 더 광범위한 적용 가능성
- 계층 구조hierarchy: 잘 팩토링된 해결책을 구현하는 핵심 개념
- “진정 만족스러운 방식으로 해결할 수 있는 유일한 문제들은 잘 팩토링된 해결책을 마침내 수용하는 문제일 것이다”
- 팩토링된 해결책만 시도할 정도로 충분히 겸손해져야 함
- 팩토링하는 인간의 능력을 훼손하는 인터페이스를 거부하다보면 결국 팩토링이 가능해지는 결과로 이어질 것

혁명 반대 세력

- 대기업이나 컴퓨터 업계
- 보수적인 컴퓨터 사용자 그룹
- 교육 기관과 대학
- 평균 프로그래머의 낮은 지적 능력
- 교육 시스템이 획일화된 문화를 구축하는 도구로 사용되는 사회의 정치적 압력

결론

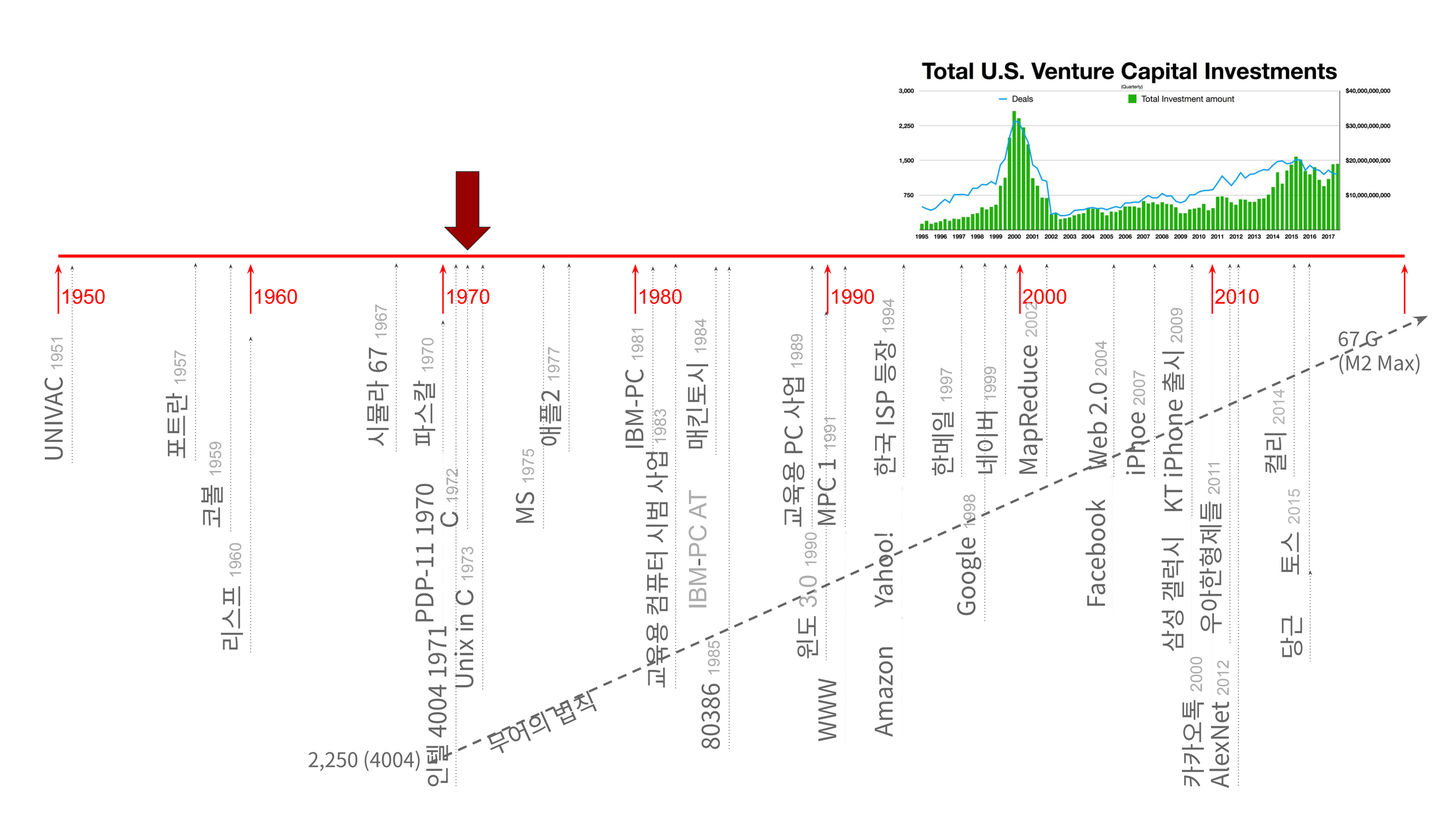
- 컴퓨터가 생긴지 25년, 자동화 도구로 큰 영향을 미쳤지만 잔물결에 불과
- 인류 문화사에 전례 없는 도구로 지적 도전에 심오한 영향을 미칠 것
- 기본 연산은 1μ초 미만이지만 프로그램 연산 시간은 몇 시간이 걸릴 수도 있음
- 10^10 또는 그 이상의 비율을 커버하는 다른 유일한 기술
- 고도의 계층 구조가 가능하고 필수적인 환경을 제공하는 최초의 기술
- 생소한 도전에서 인간에 대해 많은 것을 배울 수 것이고

'

이 도전, 즉 프로그래밍 작업과의 대립은 매우 특이해서 이 새로운 경험을 통해 인간에 대해 많은 것을 배울 수 있다. 설계와 창조의 과정에 대한 우리의 이해는 깊어지고 우리의 생각을 조직하는 작업은 더 잘 제어될 수 있어야 한다. 그렇지 않았다면, 제 의견에 인간은 컴퓨터를 전혀받을 자격이 없다!

'

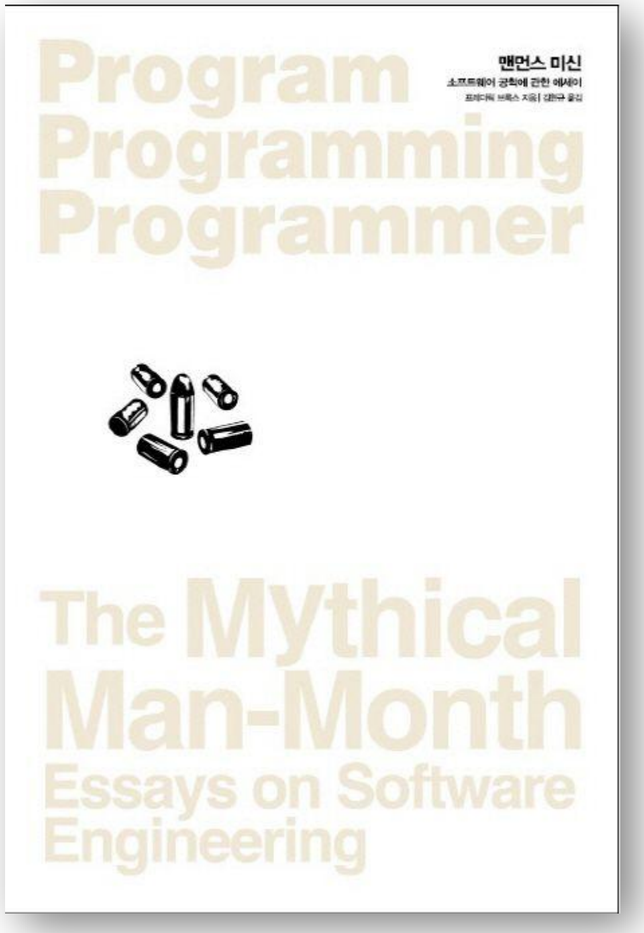
50년이 지난 지금


Total U.S. Venture Capital Investments
(Quarterly)
Deals
Total Investment amount
3,000
2,250
1,500
750
$40,000,000,000
$30,000,000,000
$20,000,000,000
$10,000,000,000
1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017
1950
1960
1970
1980
1990
2000
2010
UNIVAC 1951
포트란 1957
코볼 1959
리스프 1960
시물라 67 1967
파스칼 1970
PDP-11 1970
C 1972
Unix in C 1973
MS 1975
애플2 1977
IBM-PC 1981
교육용 컴퓨터 시범 사업 1983
매킨토시 1984
IBM-PC AT 80386 1985
교육용 PC 사업 1989
윈도 3.0 1990
MPC 1 1991
WWW
한국 ISP 등장 1994
Amazon
Yahoo!
한메일 1997
Google 1998
네이버 1999
MapReduce 2002
Facebook
Web 2.0 2004
iPhoe 2007
삼성 갤럭시
KT iPhone 출시 2009
카카오톡 2000
AlexNet 2012
우아한형제들 2011
컬리 2014
당근
토스 2015
인텔 4004 1971
2,250 (4004)
무어의 법칙
67 G
(M2 Max)

본질적 복잡성과 우발적 복잡성

“은탄환은 없다”- 제거할 수 없는 소프트웨어의 본질적인 복잡성

- 동일한 요소가 전혀 없음
- 많은 수의 상태
- 규모에 따라 증가하는 요소가 비선형적으로 상호 작용



‘브룩스에 따라 우리는 우발적 복잡성과 본질적 복잡성으로 구분하지만, 현대 시스템에 남아있는 대부분의 복잡성이 본질적이라는 그의 전제에는 동의하지 않는다.’

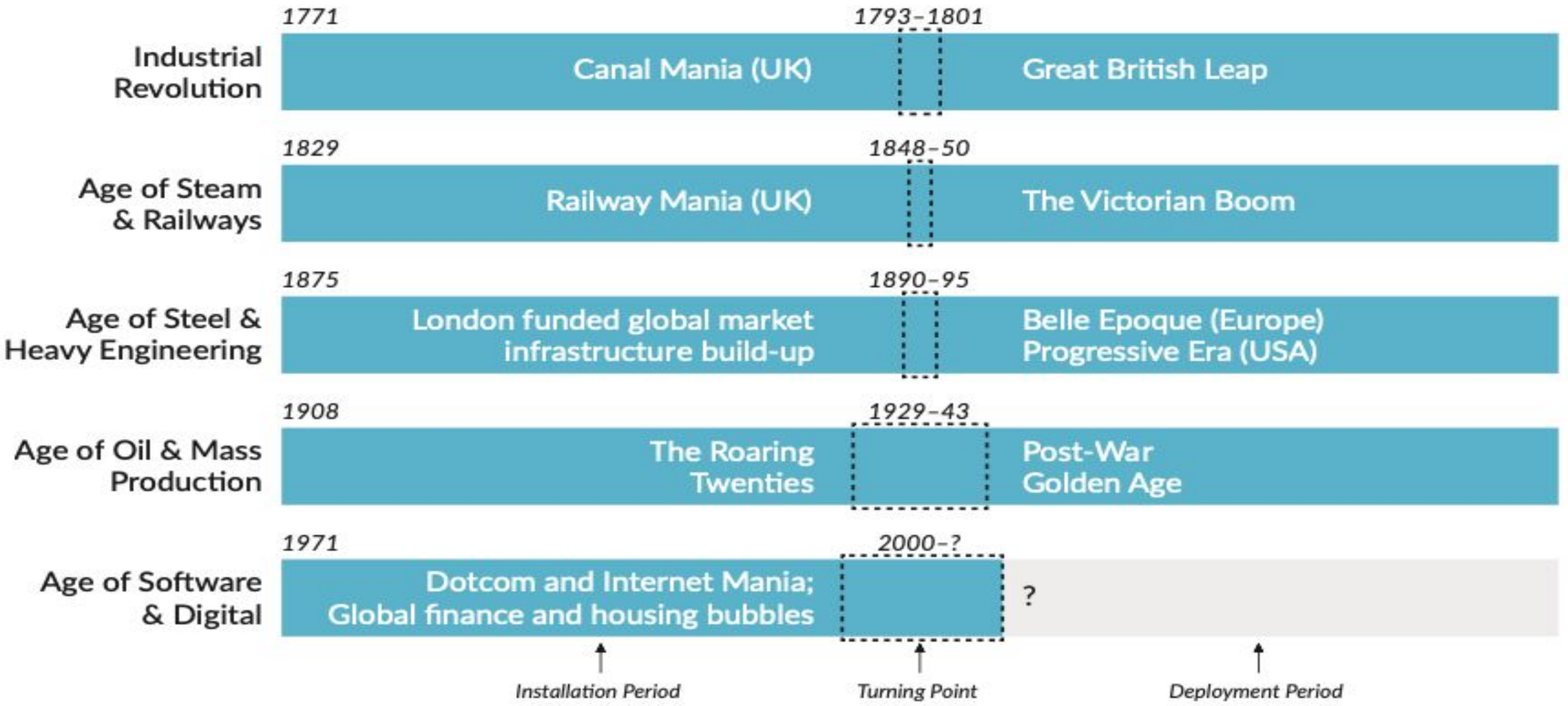
- 벤 모슬리 & 피터 마크스, ‘타르 웅덩이 밖으로’

변화

- 오류 없는 소프트웨어를 만드는 문제를 넘어 유지보수의 문제
- PC 혁명, IoT, 클라우드 - “네크워크는 컴퓨터다The Network is the Computer”
- 인터넷, 통신 혁명 - 초연결 사회, 소프트웨어로 돌아가는 세상

조건에 대해

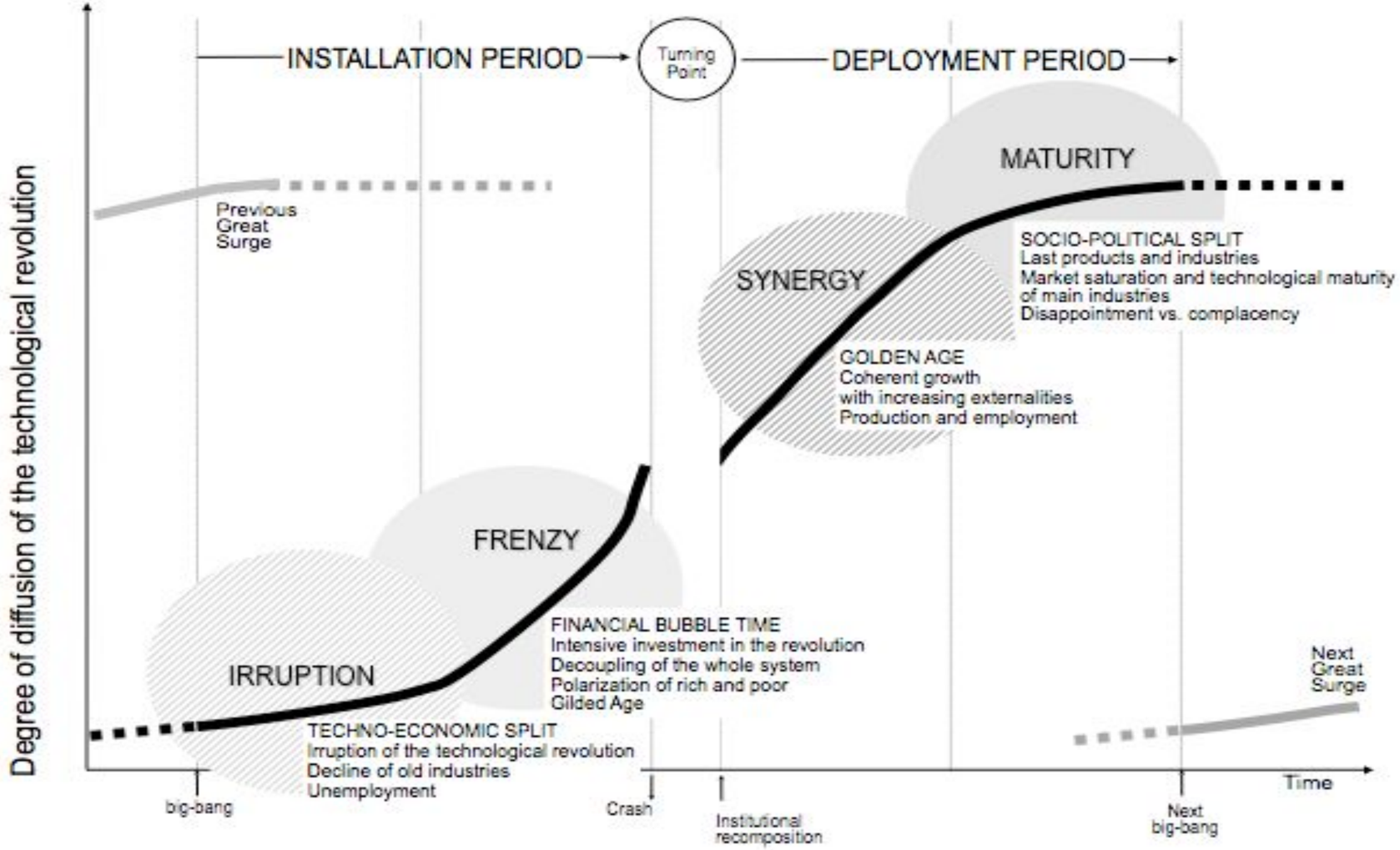
소프트웨어 시대



Digitization → Digitalization → Digital Tx

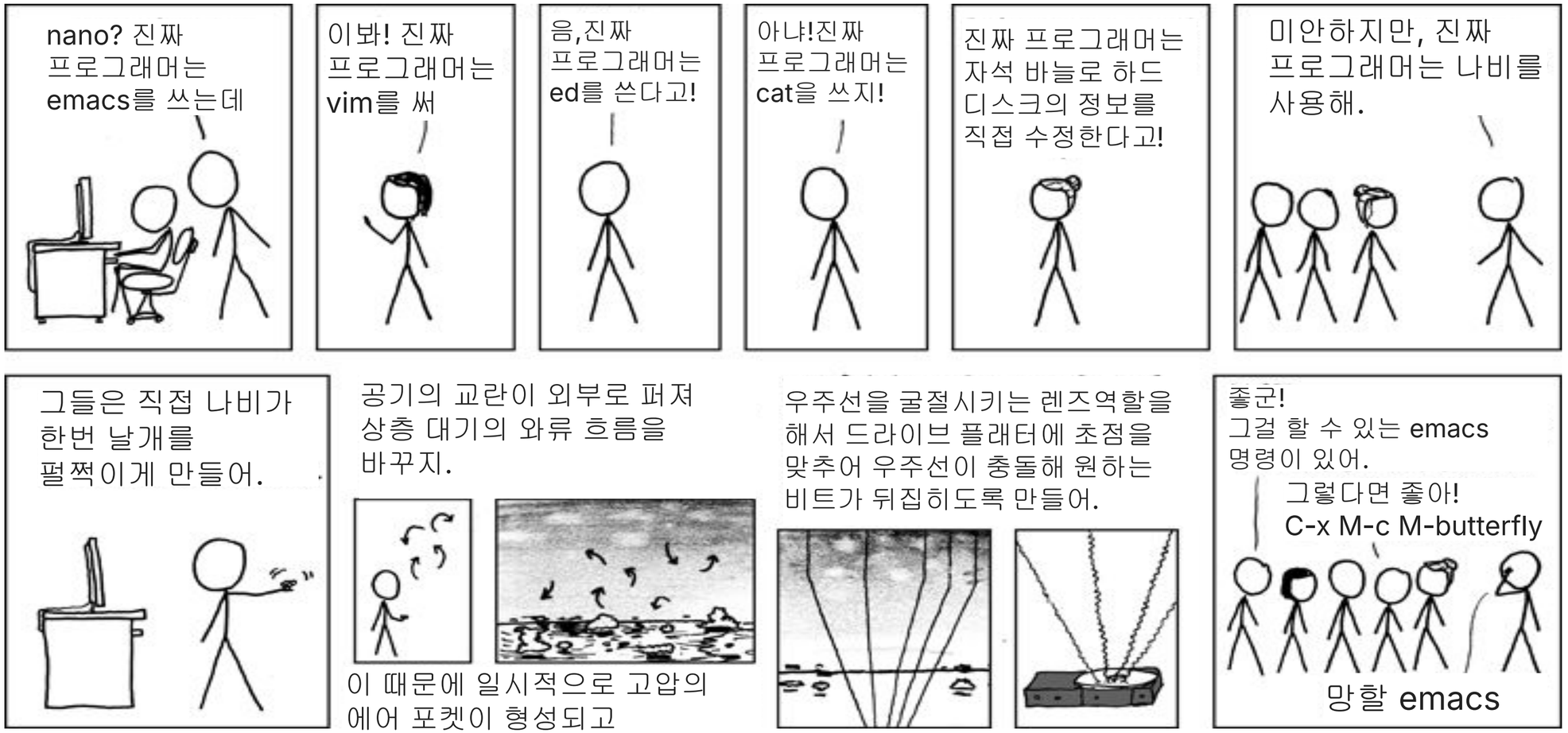
기술 생애 주기의 단계

"기술혁명과 금융자본", 카를로타 페레스



결론

진짜 프로그래머



https://xkcd.com/378/

락스타


ROCK STAR DEVELOPER
WE'RE LOOKING FOR A ROCK STAR DEVELOPER TO JOIN OUR TEAM
HE'S GOOD!
@SKELETON_CLAW
SKELETONCLAW.COM

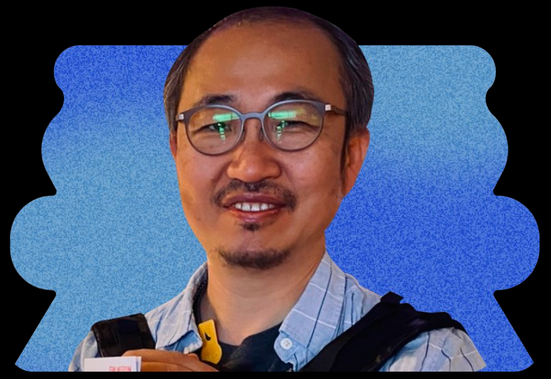

All right, folks, we need a catchy job title
Java Developer
IT Rockstar Java Pro Evangelist

'

이미 우리는 몇 가지 교훈을 얻었고, 그 중에서 이 강연에서 강조하고자 하는 것은 다음과 같다.
이 작업의 엄청난 어려움을 충분히 인식하고, 수수하고 품격 있는 프로그래밍 언어를 고수하며, 인간 정신의 태생적 한계를 존중하고, 매우 겸손한 프로그래머로서 작업에 접근한다면 우리는 훨씬 더 나은 프로그래밍 작업을 할 수 있을 것이다.

'

Thank You



Speaker Name

Affiliation / Position

Optional Contact Infrmation