소프트웨어 공학

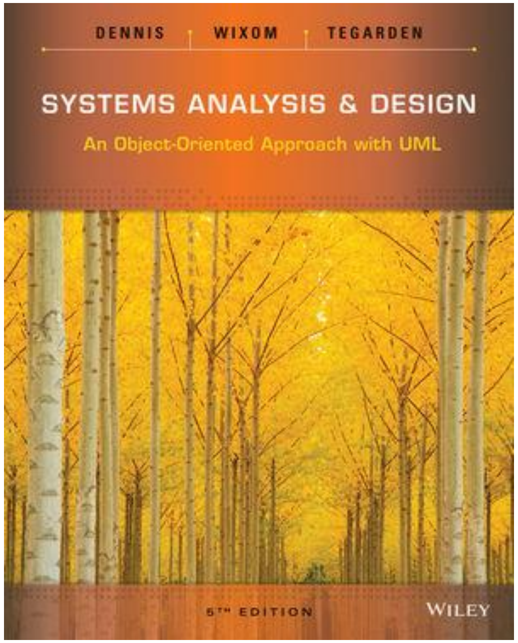
Dr. Young-Woo Kwon

Course Facts

- Meet Thur.
- Office Hour: by appointment IT4 405
- Website:
  - LMS for course materials
- Email: ywkwon@knu.ac.kr
- TA
  - 최유라

Textbook (not required)

- Systems Analysis and Design
- Software Engineering (Sommerville)


DENNIS
WIXOM
TEGARDEN
SYSTEMS ANALYSIS & DESIGN
An Object-Oriented Approach with UML
5TH EDITION
WILEY

Class Format

- Lecture
- Homework
- Exam
  - 1 Midterm and 1 Final
- Project (team project, 4 members)
- Quizzes

Grading

Project-base Learning

Exam (midterm, final)	60% (30, 30)
Project	20%
Homework	10%
Quiz	10%
Participation (attendance)	10%

Project: 20%

- Web-based application or Mobile application
  - Propose any web project that you can complete during the semester
- Step-by-step development
  - Planning
  - Analysis
  - Design
  - Implementation
  - Integration
  - Testing
- 5 documents (50%)
  - Proposal document (10%)
  - 2 progress reports (20%)
  - Final report (20%)
- 2 Presentations (proposal and final) & Demo (40%)
- Code review (10%)

Policy

- Late policy
  - 10% penalty per day only for the project (no more than 3 late days per submission)
  - No late policy for homework assignments
- Attendance
  - 50% penalty of the earned participation points

Topics To Be Covered

- Software processes, process models, and team organization
- Initial needs assessment and project planning
- Requirements capture and analysis
- Implementation and testing
  - Basics: C/C++, Python, Java Programming
  - Essentials: Algorithm/Data Structure
  - Applications: Web/Database/Network
- (Deployment)
- (Maintenance)

프로젝트

- 4가지 요구사항
  - Database를 사용할 것
  - 표준화된 개발 방법론에 따라 개발하고 이에 대한 문서를 남길 것
  - GitHub을 통해서 공동작업 할 것
  - 디자인 패턴을 3개 이상 적용할 것

프로젝트 제출물

- 제안서 (발표)
- 요구사항 명세서 (중간 보고서 1)
- 다이어그램 (중간 보고서2)
  - Use case diagram
  - Sequence diagram
  - Class diagram
- 소프트웨어 개발 문서 (최종 결과물 및 발표)
  - 디자인/아키텍쳐
  - 사용한 알고리즘/자료구조
  - 테스트 케이스

프로젝트 아이디어

- 동아리 관리 프로그램
- 소개팅 프로그램
- 선형대수 프로그램
- 음식 주문 프로그램
- 시간표/강의실 관리 프로그램
- 수강신청 도우미
- 택시 합승 프로그램
- 메뉴 추천
- 술 게임 추천

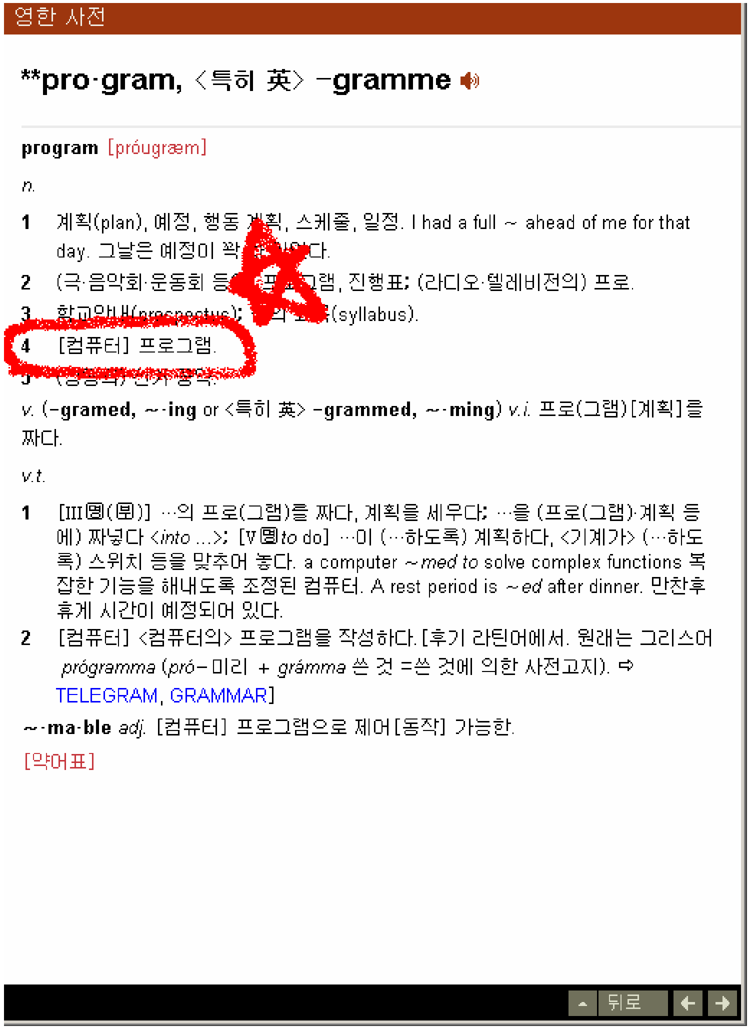
Questions

- Have you developed *“real”* software (not class project)?
- What was the most challenging process in your software development (or coding)?
- Have you used a version control system?

What is a software system?

- Computer System
  - Hardware Systems
  - Software Systems
    - focuses on the major components of software and their interactions and is also related to the field of software architecture
      - Computer Programs
      - Configuration files
      - Documents (e.g., specification, maintenance, test results, etc.)

What is a program?



pro·gram, 〈특히 英〉 -gramme

program [próugræm]

n.

1 계획(plan), 예정, 행동 계획, 스케줄, 일정. I had a full ~ ahead of me for that day. 그날은 예정이 꽉 짜여 있었다.

2 (극·음악회·운동회 등의) 프로그램, 진행표; (라디오·텔레비전의) 프로.

3 학교안내(prospectus); 강의 요목(syllabus).

4 [컴퓨터] 프로그램.

5 (방송의) 연예 종목.

v. (**-gramed, ~-ing** or 〈특히 英〉 **-grammed, ~-ming**) *v.i.* 프로(그램)[계획]을 짜다.

v.t.

1 [III 명(부)] …의 프로(그램)를 짜다, 계획을 세우다; …을 (프로(그램)·계획 등에) 짜넣다 〈*into* ...〉; [V 명 *to* do] …이 (…하도록) 계획하다, 〈기계가〉 (…하도록) 스위치 등을 맞추어 놓다. a computer ~*med to* solve complex functions 복잡한 기능을 해내도록 조정된 컴퓨터. A rest period is ~*ed* after dinner. 만찬후 휴게 시간이 예정되어 있다.

2 [컴퓨터] 〈컴퓨터의〉 프로그램을 작성하다. [후기 라틴어에서. 원래는 그리스어 *prógramma* (*pró*- 미리 + *grámma* 쓴 것 =쓴 것에 의한 사전고지). ⇨ TELEGRAM, GRAMMAR]

~·ma·ble *adj.* [컴퓨터] 프로그램으로 제어[동작] 가능한.

[약어표]

What is *Programming*?

- Design an appropriate algorithm
- Express the algorithm in psuedo-code
- Code the algorithm in a computer language

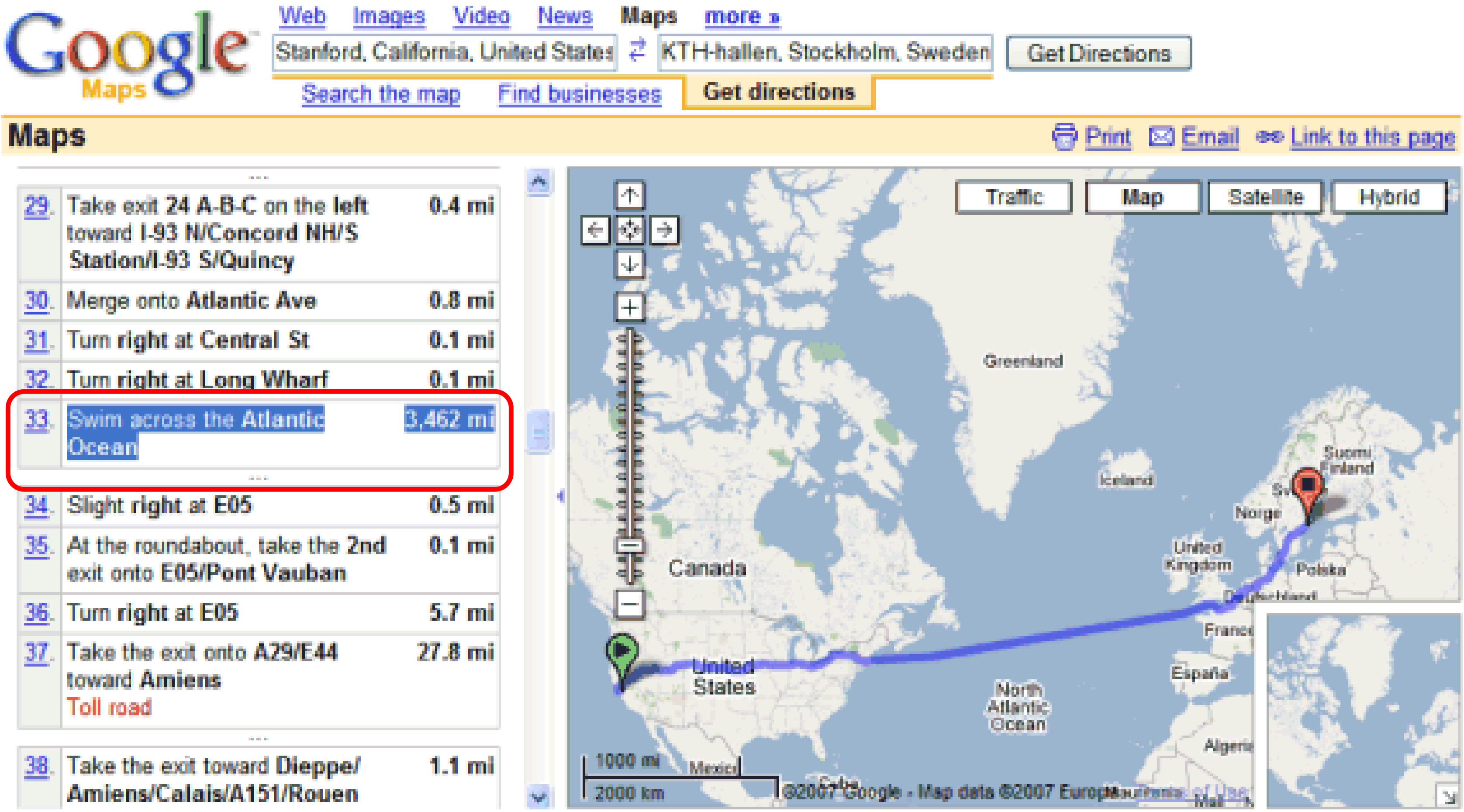
Algorithm

- Basic definition

  An algorithm is a logical sequence of instructions to accomplish a task.

- Refinements
  - *Instructions must be well ordered*
  - *Instructions must be unambiguous*
  - *The process must eventually end*
  - *The actions must be doable*
  - *The algorithm must produce the required results*

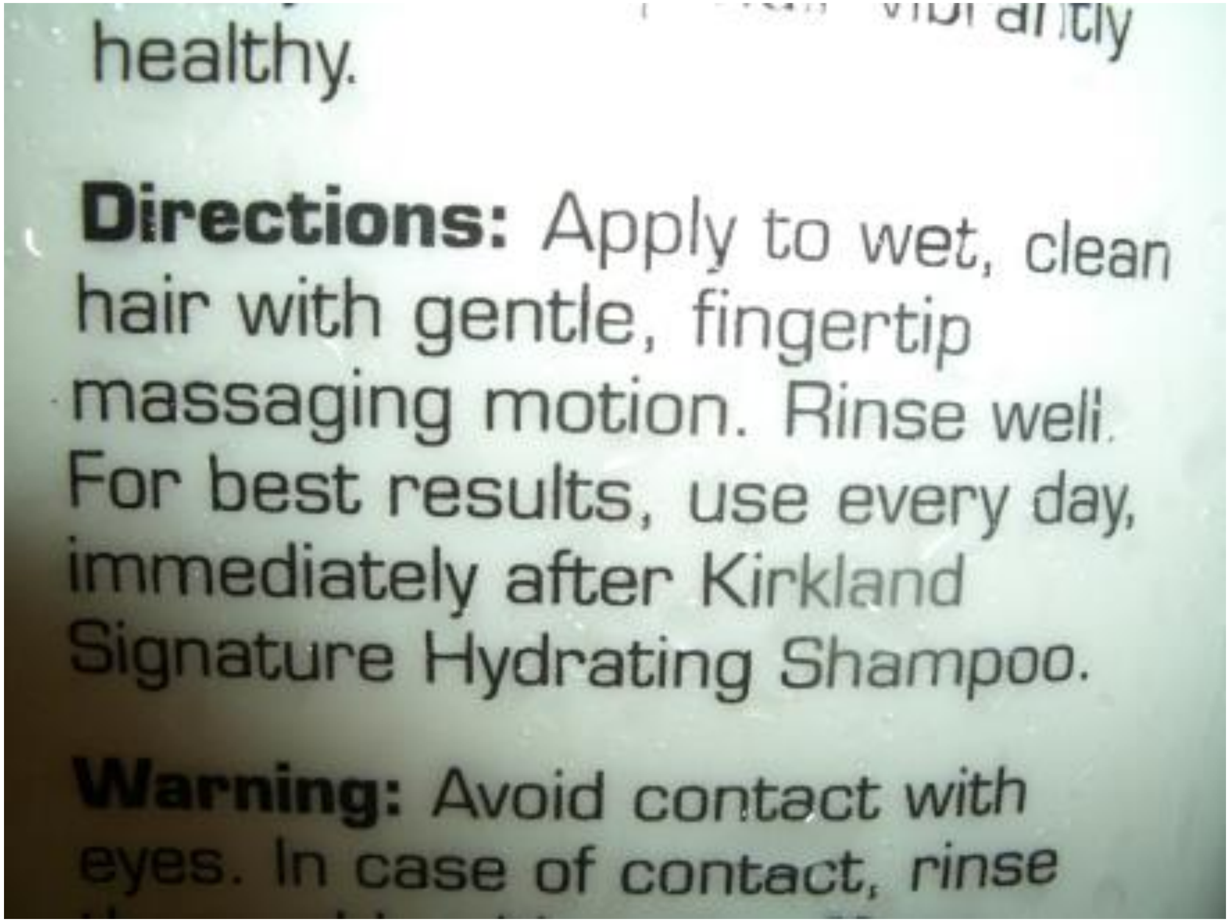
Is this an algorithm?



29.	Take exit 24 A-B-C on the left toward I-93 N/Concord NH/S Station/I-93 S/Quincy	0.4 mi
30.	Merge onto Atlantic Ave	0.8 mi
31.	Turn right at Central St	0.1 mi
32.	Turn right at Long Wharf	0.1 mi
33.	Swim across the Atlantic Ocean	3,462 mi
34.	Slight right at E05	0.5 mi
35.	At the roundabout, take the 2nd exit onto E05/Pont Vauban	0.1 mi
36.	Turn right at E05	5.7 mi
37.	Take the exit onto A29/E44 toward Amiens Toll road	27.8 mi
38.	Take the exit toward Dieppe/ Amiens/Calais/A151/Rouen	1.1 mi

The actions must be doable

Is this an algorithm?



vibrantly healthy.

Directions: Apply to wet, clean hair with gentle, fingertip massaging motion. Rinse well. For best results, use every day, immediately after Kirkland Signature Hydrating Shampoo.

Warning: Avoid contact with eyes. In case of contact, rinse

No! Why not?

Are instructions clear?

What makes Programming so hard to learn?

- A programmer is going to the grocery store and his wife tells him, "Buy a gallon of milk, and if there are eggs, buy a dozen."

```
public int getNumberOfMilkToBuy(bool storeHasEggs) {
 int milkToBuy;

         ……
 return milkToBuy;
}
```

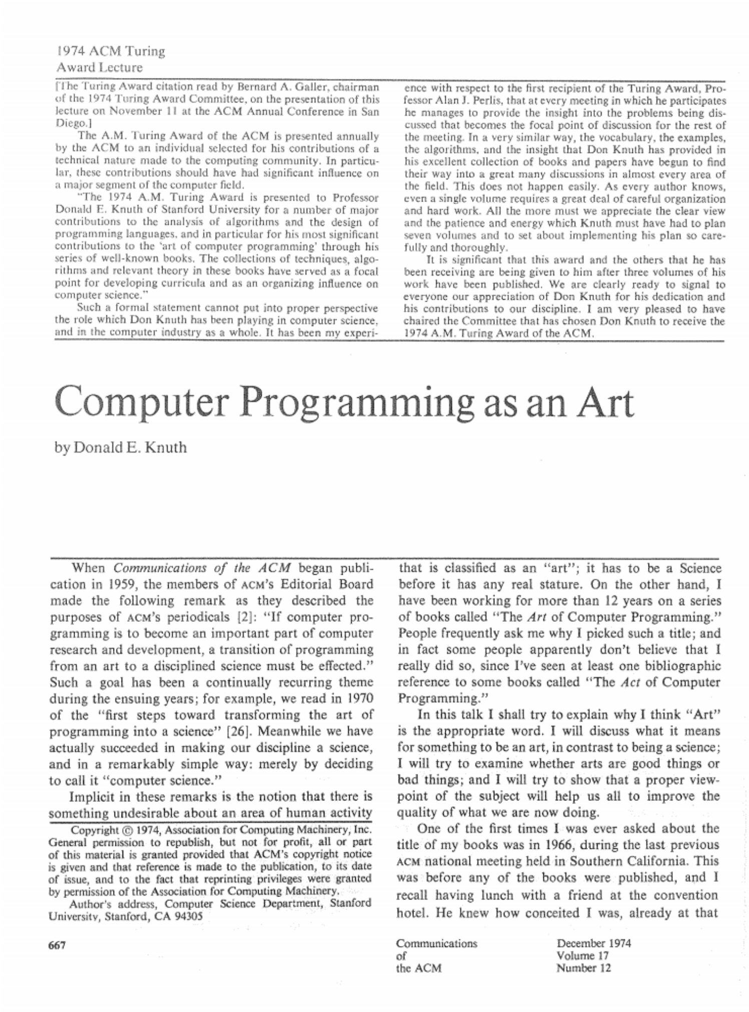
- So the programmer goes, buys everything, and drives back to his house. Upon arrival, his wife angrily asks him, "Why did you get 13 gallons of milk?" The programmer says, "There were eggs!"

```
public int getNumberOfMilkToBuy(bool storeHasEggs) {
 int milkToBuy = 1;
 if(storeHasEggs) milkToBuy += 12;
  return milkToBuy;
}
```

Human vs. Computer

- Human:
  - Interested in modeling the real world
  - More interested in what computer should do than how
- Computer:
  - Only data it can manipulate is sequences of zeros and ones
  - Understands low-level “how” instructions.

Donald Knuth:
The Art of Computer Programming



1974 ACM Turing
Award Lecture

[The Turing Award citation read by Bernard A. Galler, chairman of the 1974 Turing Award Committee, on the presentation of this lecture on November 11 at the ACM Annual Conference in San Diego.]

The A.M. Turing Award of the ACM is presented annually by the ACM to an individual selected for his contributions of a technical nature made to the computing community. In particular, these contributions should have had significant influence on a major segment of the computer field.

"The 1974 A.M. Turing Award is presented to Professor Donald E. Knuth of Stanford University for a number of major contributions to the analysis of algorithms and the design of programming languages, and in particular for his most significant contributions to the 'art of computer programming' through his series of well-known books. The collections of techniques, algorithms and relevant theory in these books have served as a focal point for developing curricula and as an organizing influence on computer science."

Such a formal statement cannot put into proper perspective the role which Don Knuth has been playing in computer science, and in the computer industry as a whole. It has been my experience with respect to the first recipient of the Turing Award, Professor Alan J. Perlis, that at every meeting in which he participates he manages to provide the insight into the problems being discussed that becomes the focal point of discussion for the rest of the meeting. In a very similar way, the vocabulary, the examples, the algorithms, and the insight that Don Knuth has provided in his excellent collection of books and papers have begun to find their way into a great many discussions in almost every area of the field. This does not happen easily. As every author knows, even a single volume requires a great deal of careful organization and hard work. All the more must we appreciate the clear view and the patience and energy which Knuth must have had to plan seven volumes and to set about implementing his plan so carefully and thoroughly.

It is significant that this award and the others that he has been receiving are being given to him after three volumes of his work have been published. We are clearly ready to signal to everyone our appreciation of Don Knuth for his dedication and his contributions to our discipline. I am very pleased to have chaired the Committee that has chosen Don Knuth to receive the 1974 A.M. Turing Award of the ACM.

Computer Programming as an Art

by Donald E. Knuth

When *Communications of the ACM* began publication in 1959, the members of ACM's Editorial Board made the following remark as they described the purposes of ACM's periodicals [2]: "If computer programming is to become an important part of computer research and development, a transition of programming from an art to a disciplined science must be effected." Such a goal has been a continually recurring theme during the ensuing years; for example, we read in 1970 of the "first steps toward transforming the art of programming into a science" [26]. Meanwhile we have actually succeeded in making our discipline a science, and in a remarkably simple way: merely by deciding to call it "computer science."

Implicit in these remarks is the notion that there is something undesirable about an area of human activity that is classified as an "art"; it has to be a Science before it has any real stature. On the other hand, I have been working for more than 12 years on a series of books called "The *Art* of Computer Programming." People frequently ask me why I picked such a title; and in fact some people apparently don't believe that I really did so, since I've seen at least one bibliographic reference to some books called "The *Act* of Computer Programming."

In this talk I shall try to explain why I think "Art" is the appropriate word. I will discuss what it means for something to be an art, in contrast to being a science; I will try to examine whether arts are good things or bad things; and I will try to show that a proper viewpoint of the subject will help us all to improve the quality of what we are now doing.

One of the first times I was ever asked about the title of my books was in 1966, during the last previous ACM national meeting held in Southern California. This was before any of the books were published, and I recall having lunch with a friend at the convention hotel. He knew how conceited I was, already at that

Author's address, Computer Science Department, Stanford University, Stanford, CA 94305

667

Communications
of
the ACM

December 1974
Volume 17
Number 12

We have seen that ***computer programming is an art***, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty. A programmer who subconsciously views himself as an artist will enjoy what he does and will do it better. Therefore we can be glad that people who lecture at computer conferences speak about the state of the Art.

Donald Knuth: The Art of Computer Programming

마이크로 소프트웨어 2005/02

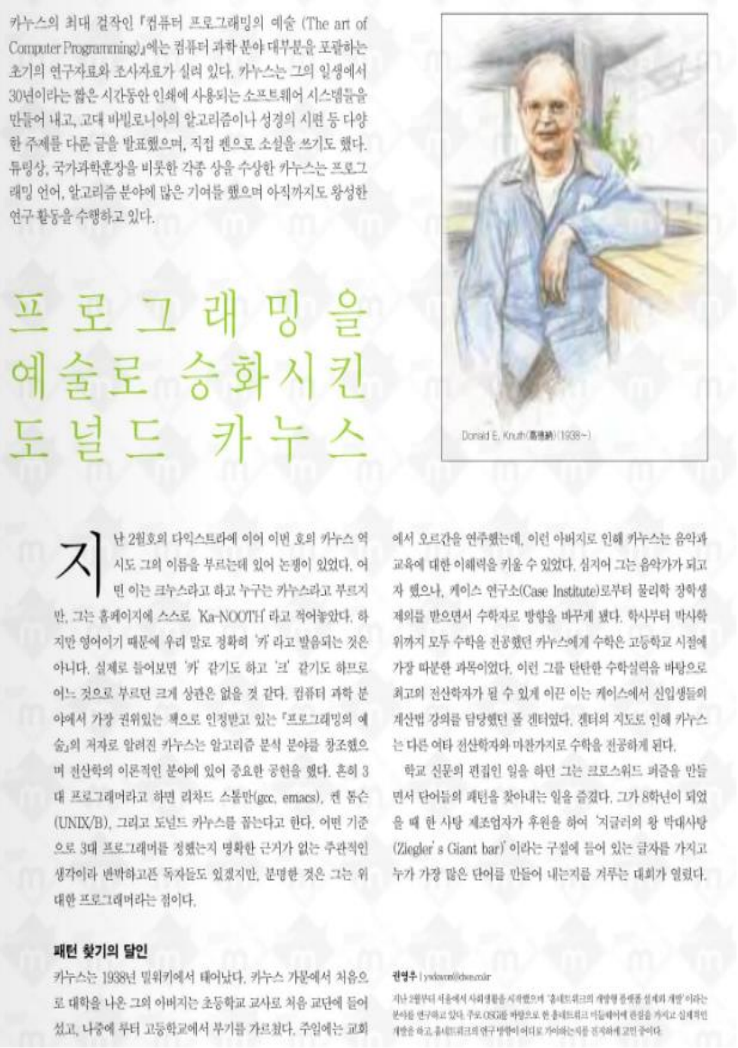
프로그래밍은 예술이 될 수 있을까?

예술 ← 반대 →과학

과학: 지식

예술: 그 지식의 적용

우리는 컴퓨터 프로그래밍을 하나의 예술로 생각한다. 그것은 그 안에 세상에 대한 지식이 축적되어 있기 때문이고, ***기술(skill)****과* ***독창성(ingenuity)****을 요구하기 때문이고 그리고 아름다움의 대상 (objects of beauty)을 창조하기 때문이다. 어렴풋하게나마 자신을 예술가(artist)라고 의식하는 프로그래머는 스스로 하는 일을 진정으로 즐길 것이며, 또한 남보다 더 훌륭한 작품을 내놓을 것이다.*



카누스의 최대 걸작인 『컴퓨터 프로그래밍의 예술 (The art of Computer Programming)』에는 컴퓨터 과학 분야 대부분을 포괄하는 초기의 연구자료와 조사자료가 실려 있다. 카누스는 그의 일생에서 30년이라는 짧은 시간동안 인쇄에 사용되는 소프트웨어 시스템들을 만들어 내고, 고대 바빌로니아의 알고리즘이나 성경의 시편 등 다양한 주제를 다룬 글을 발표했으며, 직접 펜으로 소설을 쓰기도 했다. 튜링상, 국가과학훈장을 비롯한 각종 상을 수상한 카누스는 프로그래밍 언어, 알고리즘 분야에 많은 기여를 했으며 아직까지도 왕성한 연구 활동을 수행하고 있다.

프로그래밍을 예술로 승화시킨 도널드 카누스

Donald E. Knuth(高德納)(1938~)

지난 2월호의 다익스트라에 이어 이번 호의 카누스 역시도 그의 이름을 부르는데 있어 논쟁이 있었다. 어떤 이는 크누스라고 하고 누구는 카누스라고 부르지만, 그는 홈페이지에 스스로 'Ka-NOOTH' 라고 적어놓았다. 하지만 영어이기 때문에 우리 말로 정확히 '카' 라고 발음되는 것은 아니다. 실제로 들어보면 '카' 같기도 하고 '크' 같기도 하므로 어느 것으로 부르던 크게 상관은 없을 것 같다. 컴퓨터 과학 분야에서 가장 권위있는 책으로 인정받고 있는 『프로그래밍의 예술』의 저자로 알려진 카누스는 알고리즘 분석 분야를 창조했으며 전산학의 이론적인 분야에 있어 중요한 공헌을 했다. 흔히 3대 프로그래머라고 하면 리차드 스톨만(gcc, emacs), 켄 톰슨(UNIX/B), 그리고 도널드 카누스를 꼽는다고 한다. 어떤 기준으로 3대 프로그래머를 정했는지 명확한 근거가 없는 주관적인 생각이라 반박하고픈 독자들도 있겠지만, 분명한 것은 그는 위대한 프로그래머라는 점이다.

패턴 찾기의 달인

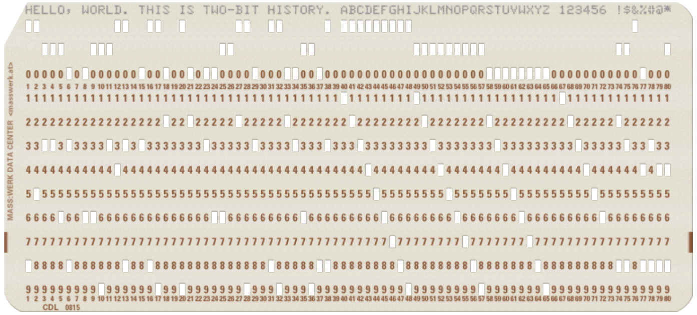
카누스는 1938년 밀워키에서 태어났다. 카누스 가문에서 처음으로 대학을 나온 그의 아버지는 초등학교 교사로 처음 교단에 들어섰고, 나중에 부터 고등학교에서 부기를 가르쳤다. 주일에는 교회에서 오르간을 연주했는데, 이런 아버지로 인해 카누스는 음악과 교육에 대한 이해력을 키울 수 있었다. 심지어 그는 음악가가 되고자 했으나, 케이스 연구소(Case Institute)로부터 물리학 장학생 제의를 받으면서 수학자로 방향을 바꾸게 됐다. 학사부터 박사학위까지 모두 수학을 전공했던 카누스에게 수학은 고등학교 시절에 가장 따분한 과목이었다. 이런 그를 탄탄한 수학실력을 바탕으로 최고의 전산학자가 될 수 있게 이끈 이는 케이스에서 신입생들의 계산법 강의를 담당했던 폴 겐터였다. 겐터의 지도로 인해 카누스는 다른 여타 전산학자와 마찬가지로 수학을 전공하게 된다.

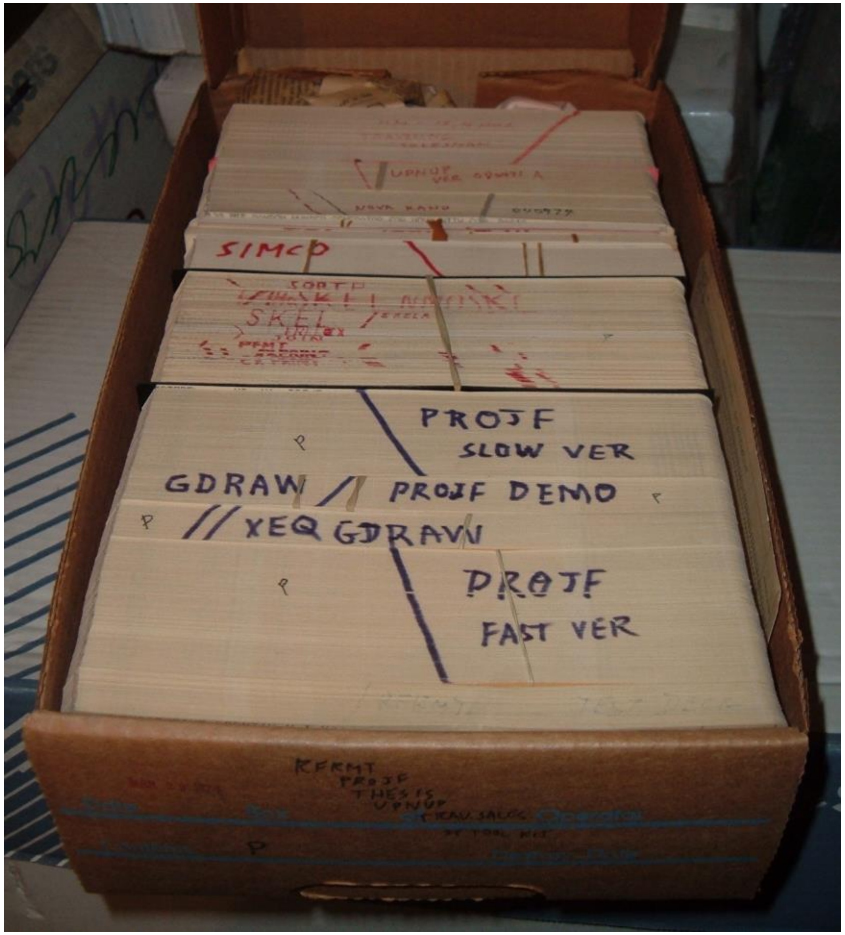
학교 신문의 편집인 일을 하던 그는 크로스워드 퍼즐을 만들면서 단어들의 패턴을 찾아내는 일을 즐겼다. 그가 8학년이 되었을 때 한 사탕 제조업자가 후원을 하여 '지글러의 왕 박대사탕(Ziegler's Giant bar)' 이라는 구절에 들어 있는 글자를 가지고 누가 가장 많은 단어를 만들어 내는지를 겨루는 대회가 열렸다.

권영우 | ywkwon@dwe.co.kr

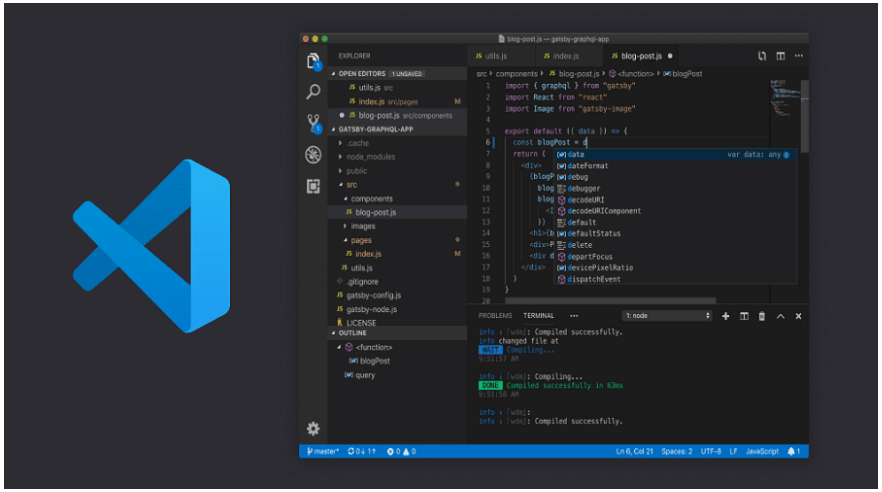
지난 2월부터 서울에서 사회생활을 시작했으며 '홈네트워크의 개방형 플랫폼 설계와 개발' 이라는 분야를 연구하고 있다. 주로 OSGi를 바탕으로 한 홈네트워크 미들웨어에 관심을 가지고 실제적인 개발을 하고, 홈네트워크의 연구 방향이 어디로 가야하는지를 진지하게 고민 중이다.

프로그래밍에 대한 인식의 변화


HELLO, WORLD. THIS IS TWO-BIT HISTORY. ABCDEFGHIJKLMNOPQRSTUVWXYZ 123456 !$&%#@*
MASS:WERK DATA CENTER <masswerk.at>
CDL 0815



프로그래밍에 대한 인식의 변화



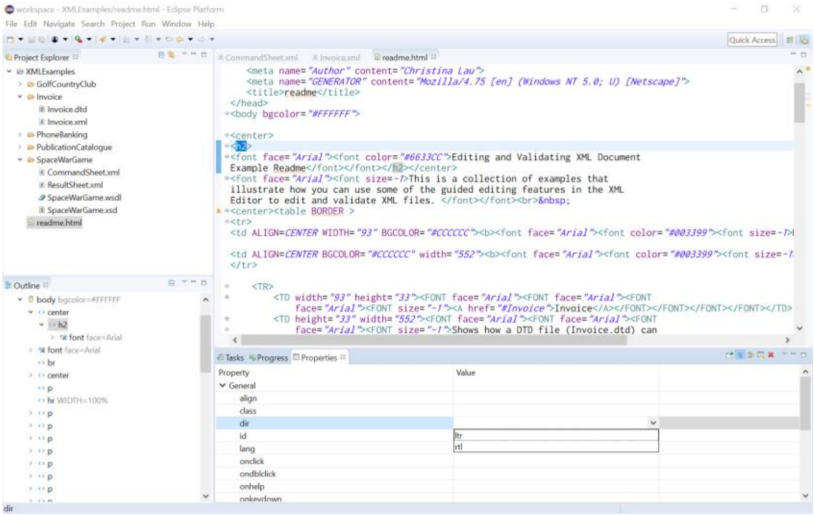
```
import { graphql } from "gatsby"
import React from "react"
import Image from "gatsby-image"

export default ({ data }) => {
  const blogPost = d
  return (
    <div>
      {blogP
        blog
        blog
          <I
      })
      <h1>{b
      <div>P
      <div d
    </div>
  )
}
```

```
info ｉ ｢wdm｣: Compiled successfully.
info changed file at
 WAIT  Compiling...
9:51:57 AM

info ｉ ｢wdm｣: Compiling...
 DONE  Compiled successfully in 63ms
9:51:58 AM

info ｉ ｢wdm｣:
info ｉ ｢wdm｣: Compiled successfully.
```



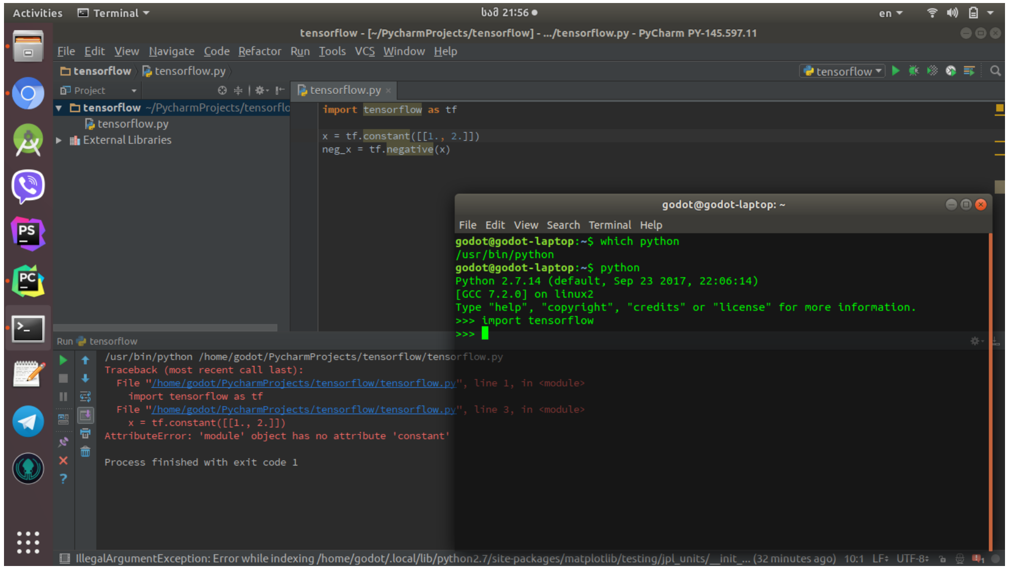
```
    <meta name="Author" content="Christina Lau">
    <meta name="GENERATOR" content="Mozilla/4.75 [en] (Windows NT 5.0; U) [Netscape]">
    <title>readme</title>
 </head>
<body bgcolor="#FFFFFF">

<center>
<h2>
<font face="Arial"><font color="#6633CC">Editing and Validating XML Document
Example Readme</font></font></h2></center>
<font face="Arial"><font size=-1>This is a collection of examples that
illustrate how you can use some of the guided editing features in the XML
Editor to edit and validate XML files. </font></font><br>&nbsp;
<center><table BORDER >
<tr>
<td ALIGN=CENTER WIDTH="93" BGCOLOR="#CCCCCC"><b><font face="Arial"><font color="#003399"><font size=-1>

<td ALIGN=CENTER BGCOLOR="#CCCCCC" width="552"><b><font face="Arial"><font color="#003399"><font size=-1
</tr>

   <TR>
       <TD width="93" height="33"><FONT face="Arial"><FONT face="Arial"><FONT
           face="Arial"><FONT size="-1"><A href="#Invoice">Invoice</A></FONT></FONT></FONT></FONT></TD>
       <TD height="33" width="552"><FONT face="Arial"><FONT face="Arial"><FONT
           face="Arial"><FONT size="-1">Shows how a DTD file (Invoice.dtd) can
```

Property	Value
General	
align	
class	
dir	
id	ltr
lang	rtl
onclick	
ondblclick	
onhelp	
onkeydown	


Activities
Terminal
일 21:56
en
tensorflow - [~/PycharmProjects/tensorflow] - .../tensorflow.py - PyCharm PY-145.597.11
File Edit View Navigate Code Refactor Run Tools VCS Window Help
tensorflow
tensorflow.py
Project
tensorflow ~/PycharmProjects/tensorflo
tensorflow.py
External Libraries
Run tensorflow
godot@godot-laptop: ~
File Edit View Search Terminal Help

```
import tensorflow as tf

x = tf.constant([[1., 2.]])
neg_x = tf.negative(x)
```

```
/usr/bin/python /home/godot/PycharmProjects/tensorflow/tensorflow.py
Traceback (most recent call last):
  File "/home/godot/PycharmProjects/tensorflow/tensorflow.py", line 1, in <module>
    import tensorflow as tf
  File "/home/godot/PycharmProjects/tensorflow/tensorflow.py", line 3, in <module>
    x = tf.constant([[1., 2.]])
AttributeError: 'module' object has no attribute 'constant'

Process finished with exit code 1
```

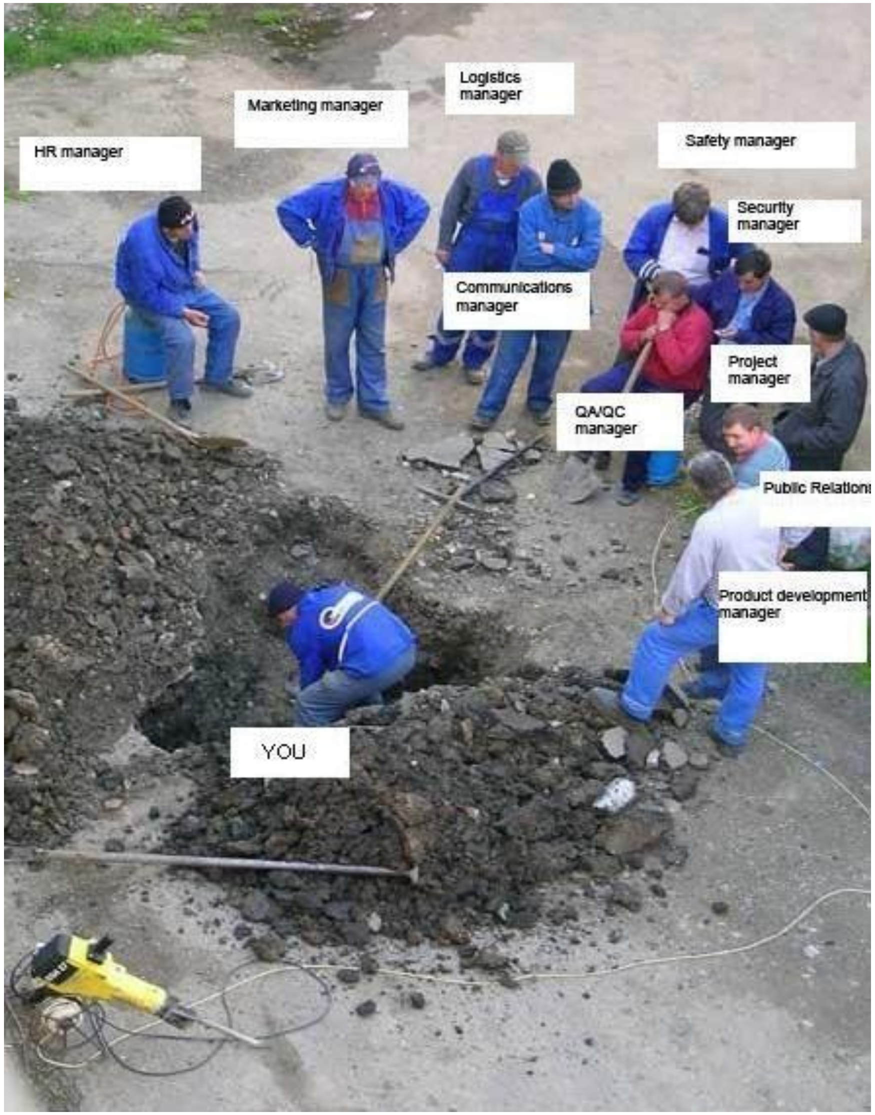
```
godot@godot-laptop:~$ which python
/usr/bin/python
godot@godot-laptop:~$ python
Python 2.7.14 (default, Sep 23 2017, 22:06:14)
[GCC 7.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow
>>> 
```

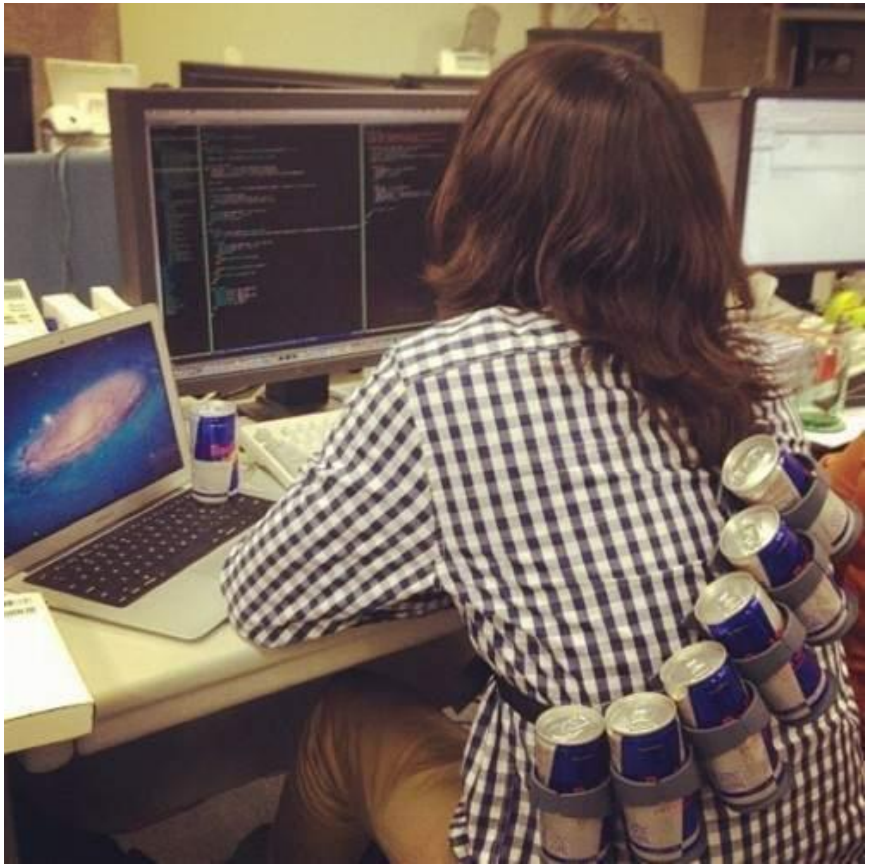
IllegalArgumentException: Error while indexing /home/godot/.local/lib/python2.7/site-packages/matplotlib/testing/jpl_units/__init__... (32 minutes ago) 10:1 LF: UTF-8:

How is SW in industry different from programming assignments?

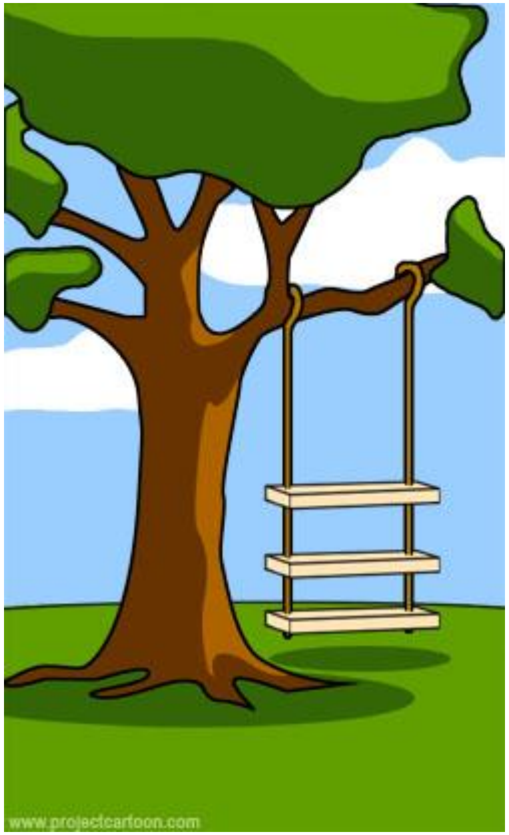
- Requirements are ambiguous
- Requirements change during development
- Scale is larger
  - Requires different design skills
  - Requires teamwork
- Software must be changed after development is complete
- Failure is more expensive
  - Business-critical
  - Safety-critical

What Is Software Development?

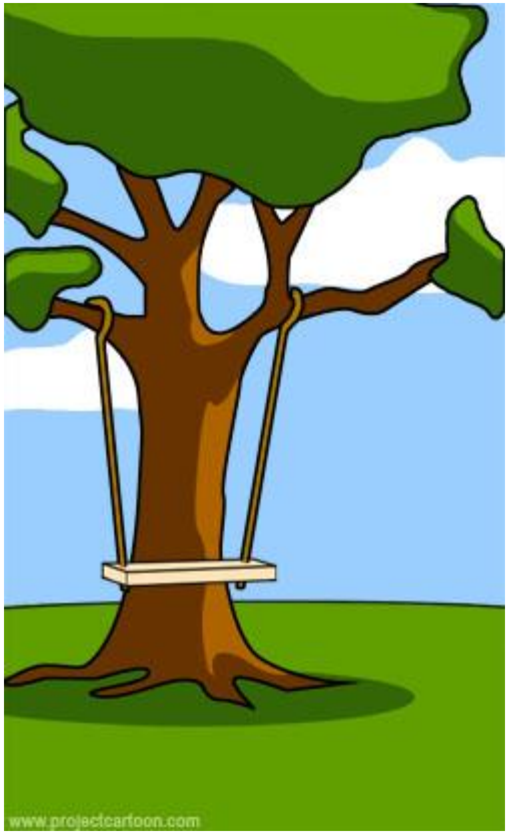




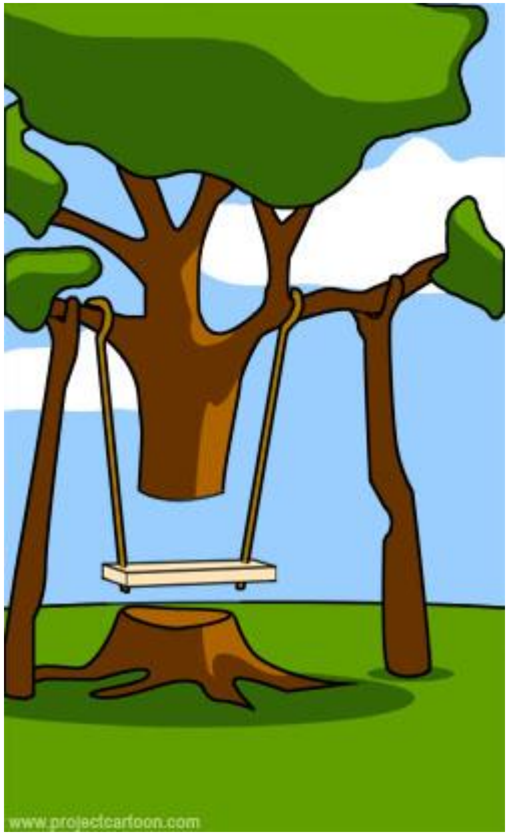
What is Really Needed?



How the customer explains it

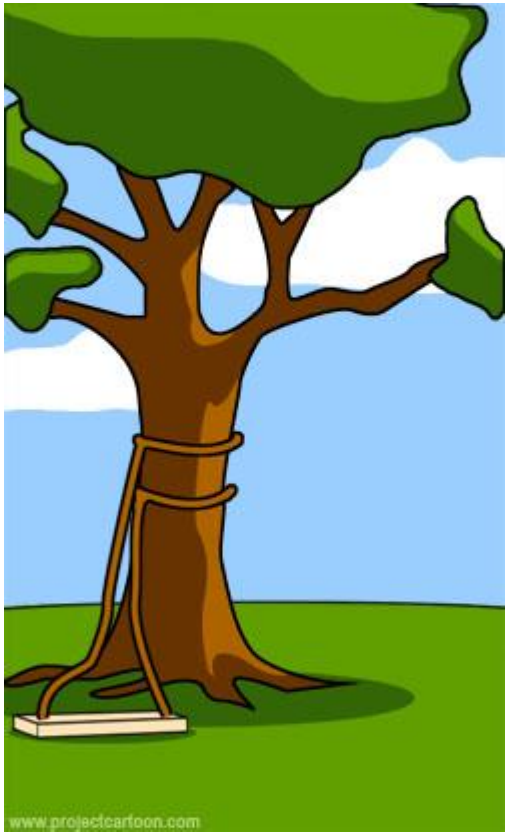


How marketing proposed it

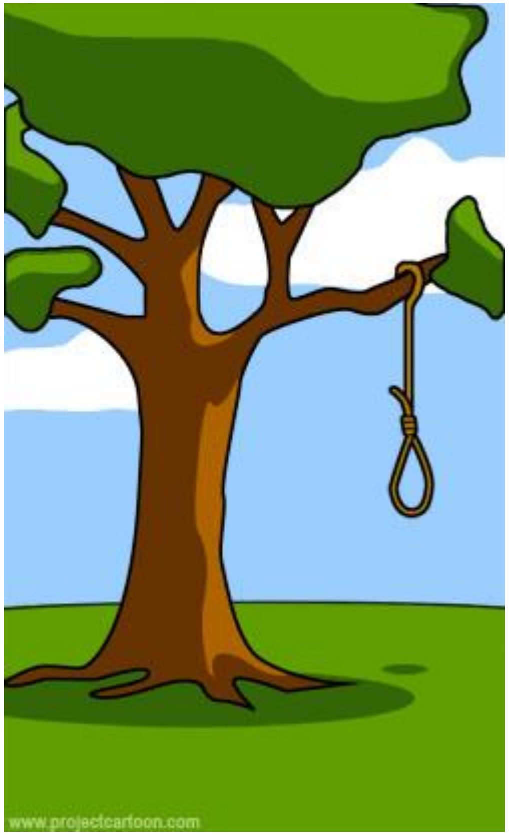


How engineers designed it

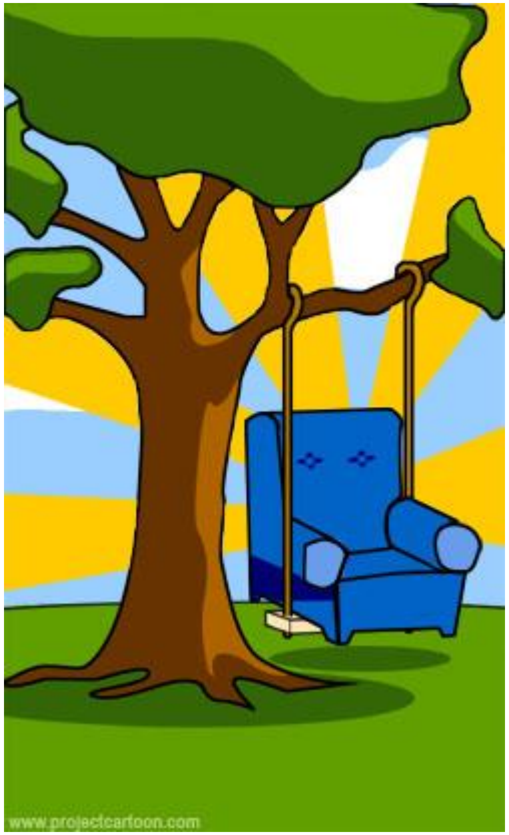
What is Really Needed?



How programmers
wrote it



How beta tester
received it

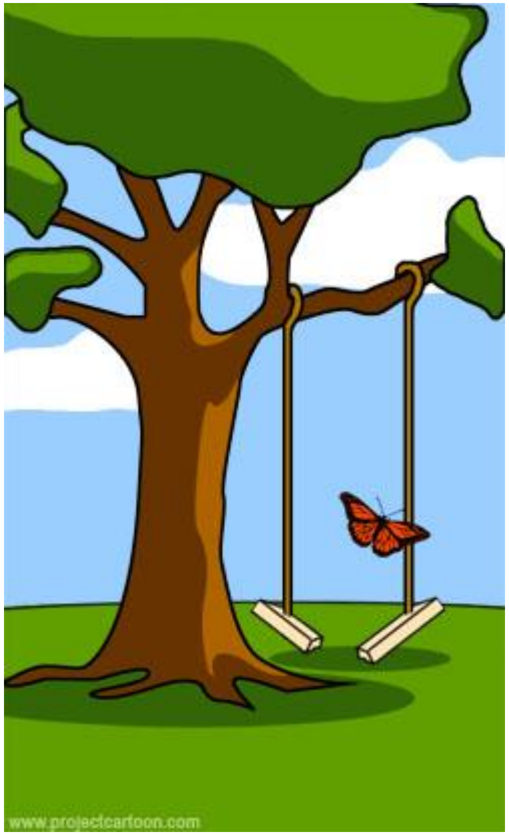


How business
consultants
described it

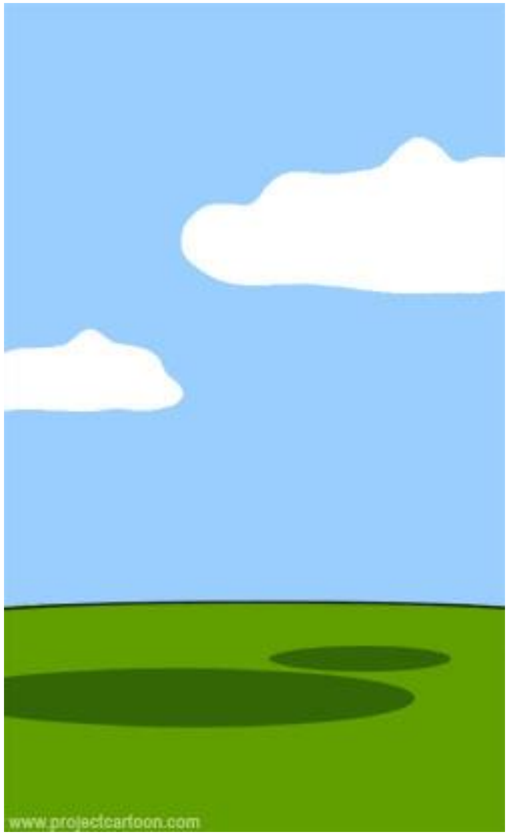
What is Really Needed?

iSwing
www.projectcartoon.com

How marketing advertised it



How it performed under load

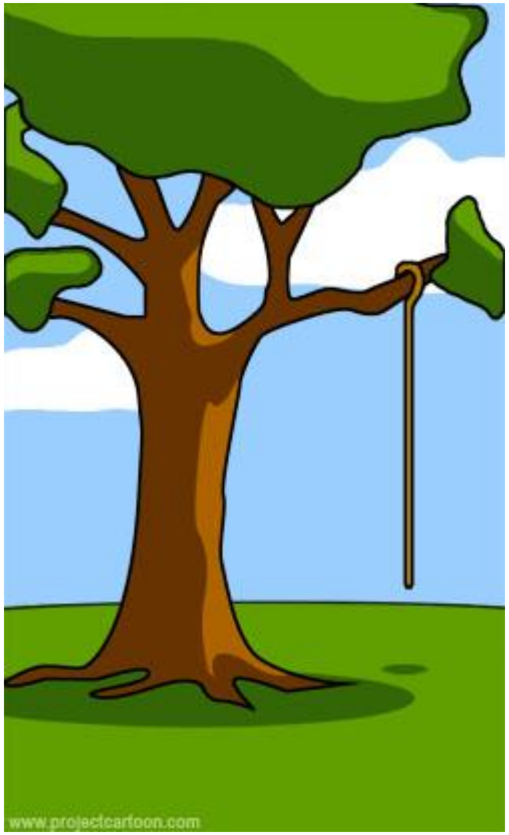


How it was documented

What is Really Needed?

www.projectcartoon.com

When it was delivered

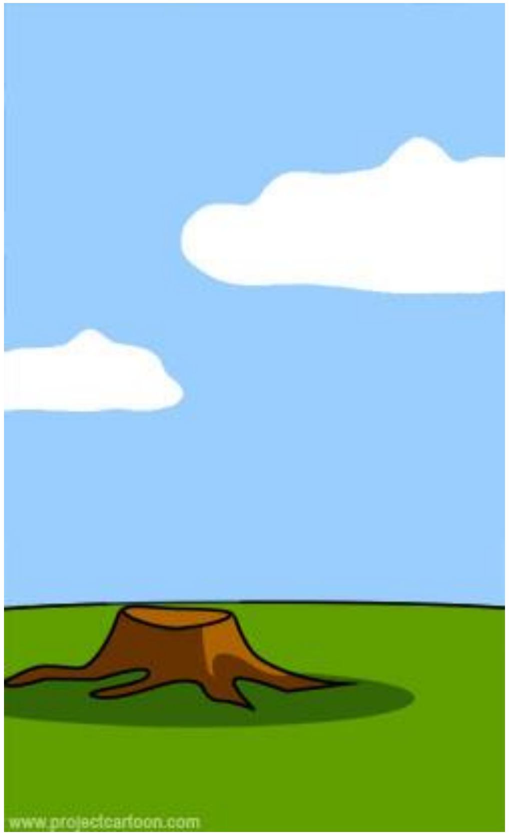


How operations installed it

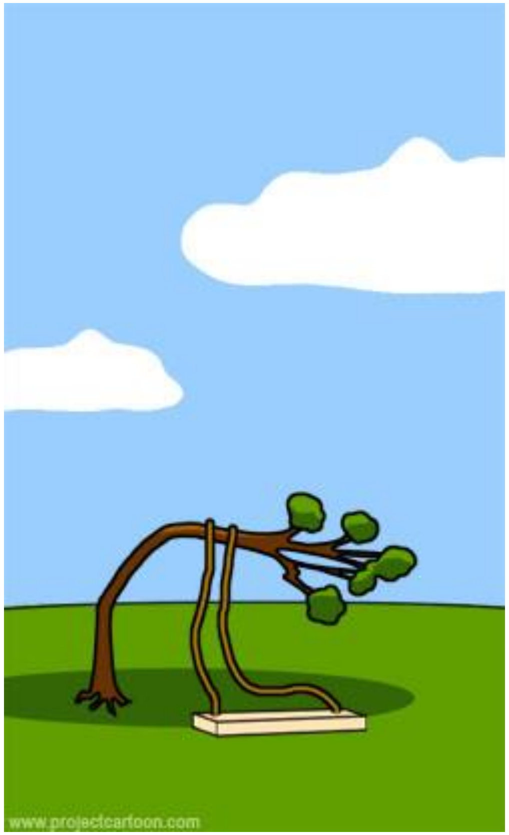
www.projectcartoon.com

How the customer was billed

What is Really Needed?



How it was supported

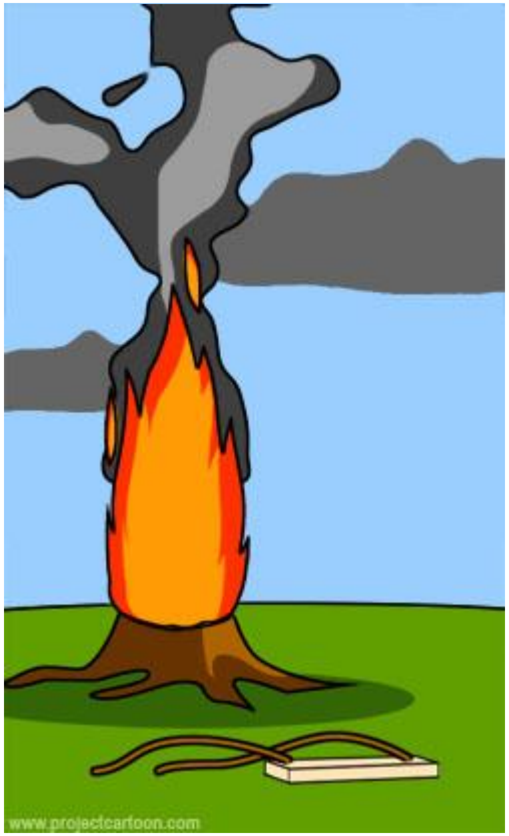


The disaster recovery plan

SECURITY SECURITY SECURITY SECURITY SECURITY SECURITY SECURITY SECURITY
www.projectcartoon.com

Updated disaster recovery plan

What is Really Needed?



What the digg effect did to it

$0.00
www.projectcartoon.com

The open source version

www.projectcartoon.com

What the customer wanted

Definitions

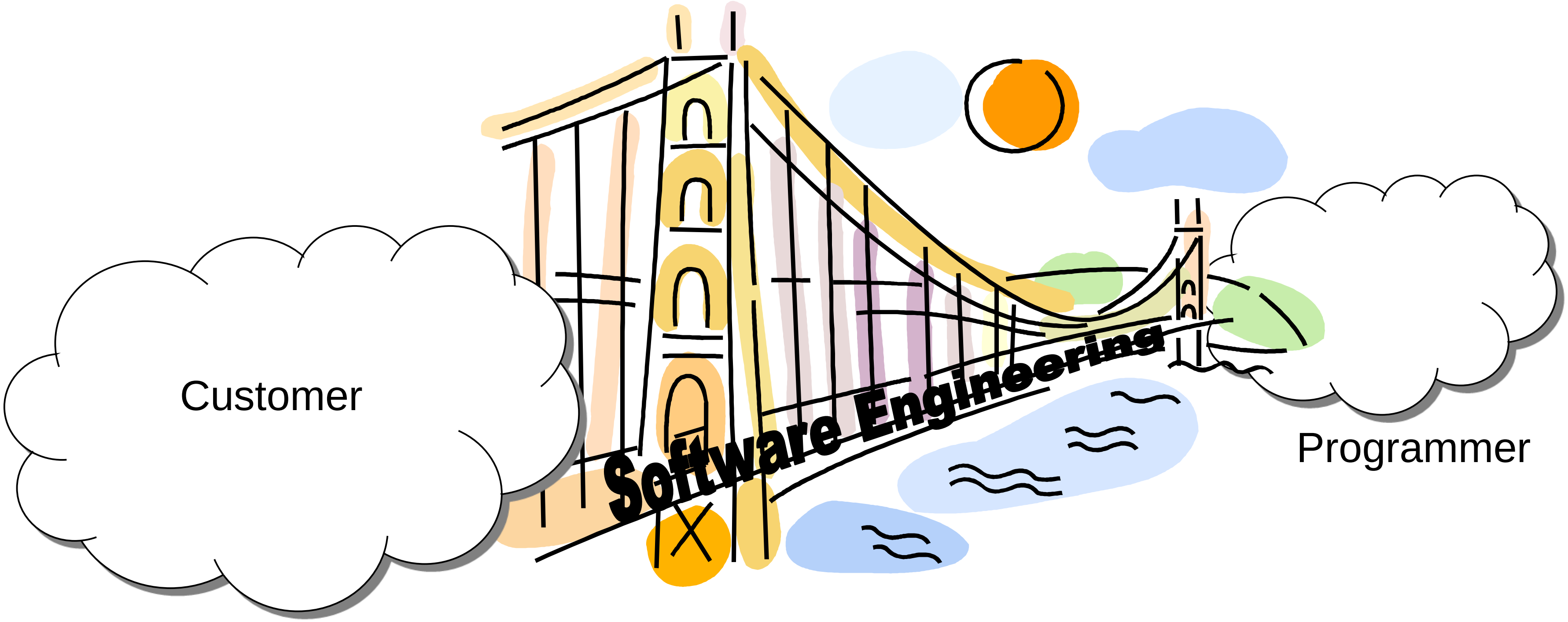
- IEEE
  - “The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software”
- Ian Sommerville
  - “An engineering discipline that is concerned with all aspects of software production”

소프트웨어 공학 vs. 타 공학

- 비즈니스를 고려
- 고객을 최우선시 함
- 설계 변경이 용이함
- 결과물이 무형(비가시적)임
- 요구사항 변경이 잦음
- 요구사항이 모호함

The Role of Software Eng.

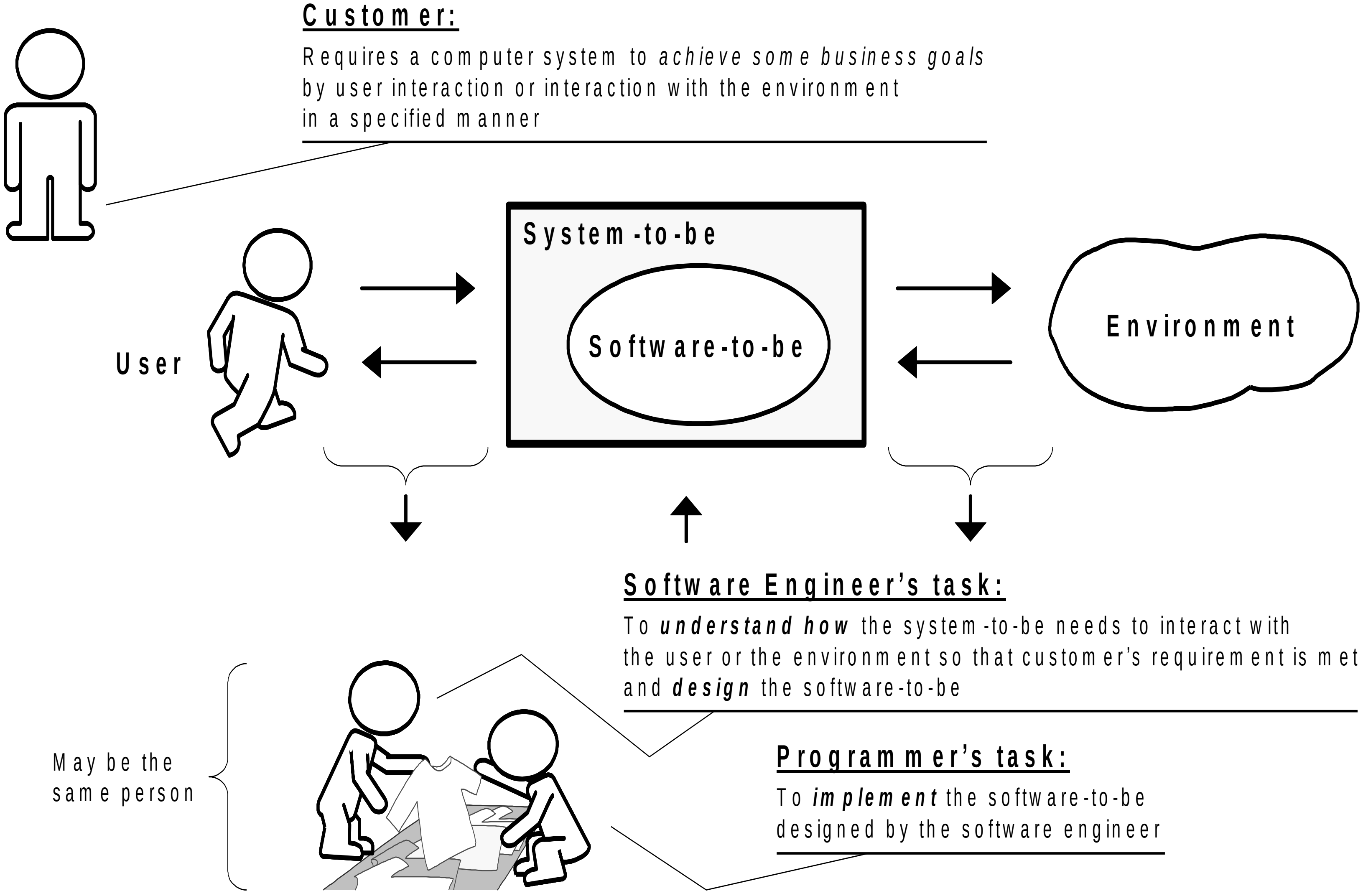
A bridge from customer needs to programming implementation



<u>First law of software engineering</u>

Software engineer is willing to learn the problem domain
(problem cannot be solved without understanding it first)

The Role of Software Eng



Customer:

Requires a computer system to *achieve some business goals* by user interaction or interaction with the environment in a specified manner

Software Engineer's task:

To ***understand how*** the system-to-be needs to interact with the user or the environment so that customer's requirement is met and ***design*** the software-to-be

Programmer's task:

To ***implement*** the software-to-be designed by the software engineer

May be the same person

What is software engineering?

Development process

Development methods

Architecture

Development tools

Project management

Financial management

Engineering Discipline + **All aspects of S/W**

What should we consider when starting S/W development?

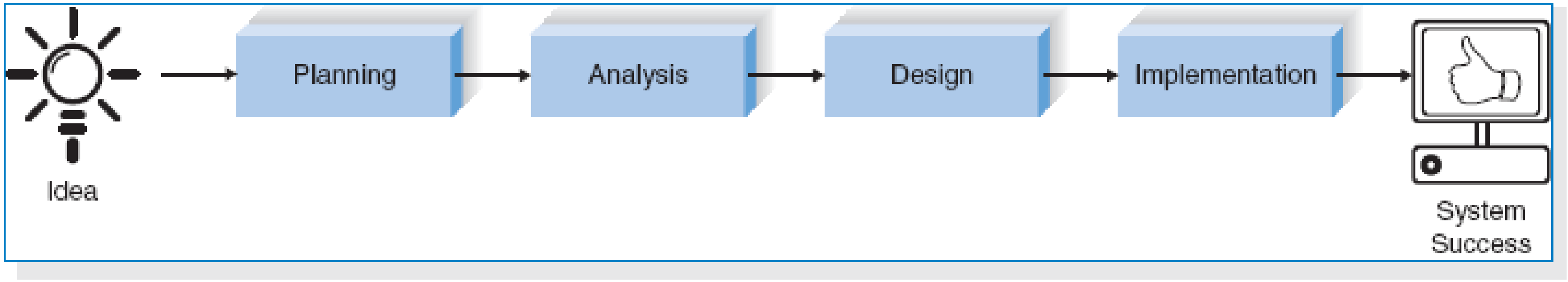


FIGURE 1-2
The Systems Development Life Cycle

What is a software process?

- Software specification
- Software development
- Software validation
- Software evolution

What is good software?

- Maintainability
- Dependability
- Efficiency
- Usability

What are the key challenges facing SE?

- The legacy challenge
- The heterogeneity challenge
- The delivery challenge

SOFTWARE PROCESS

Software Development Activities

- Gathering Requirements
- (Team management)
- Software Design
- Development (Coding)
- Testing
- (Documentation)
- Deployment
- Maintenance

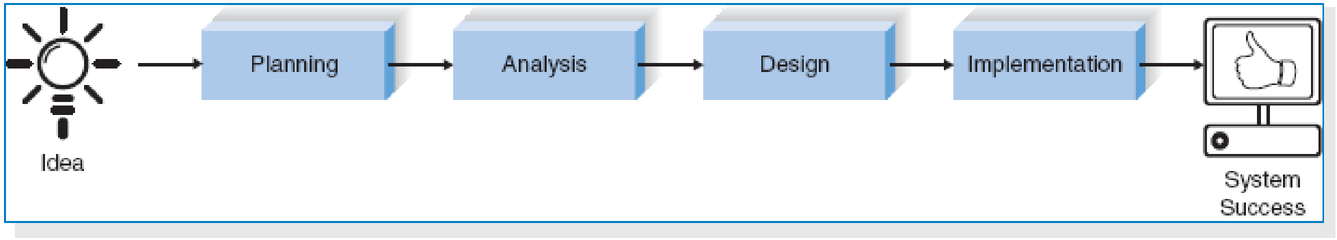


FIGURE 1-2
The Systems Development Life Cycle

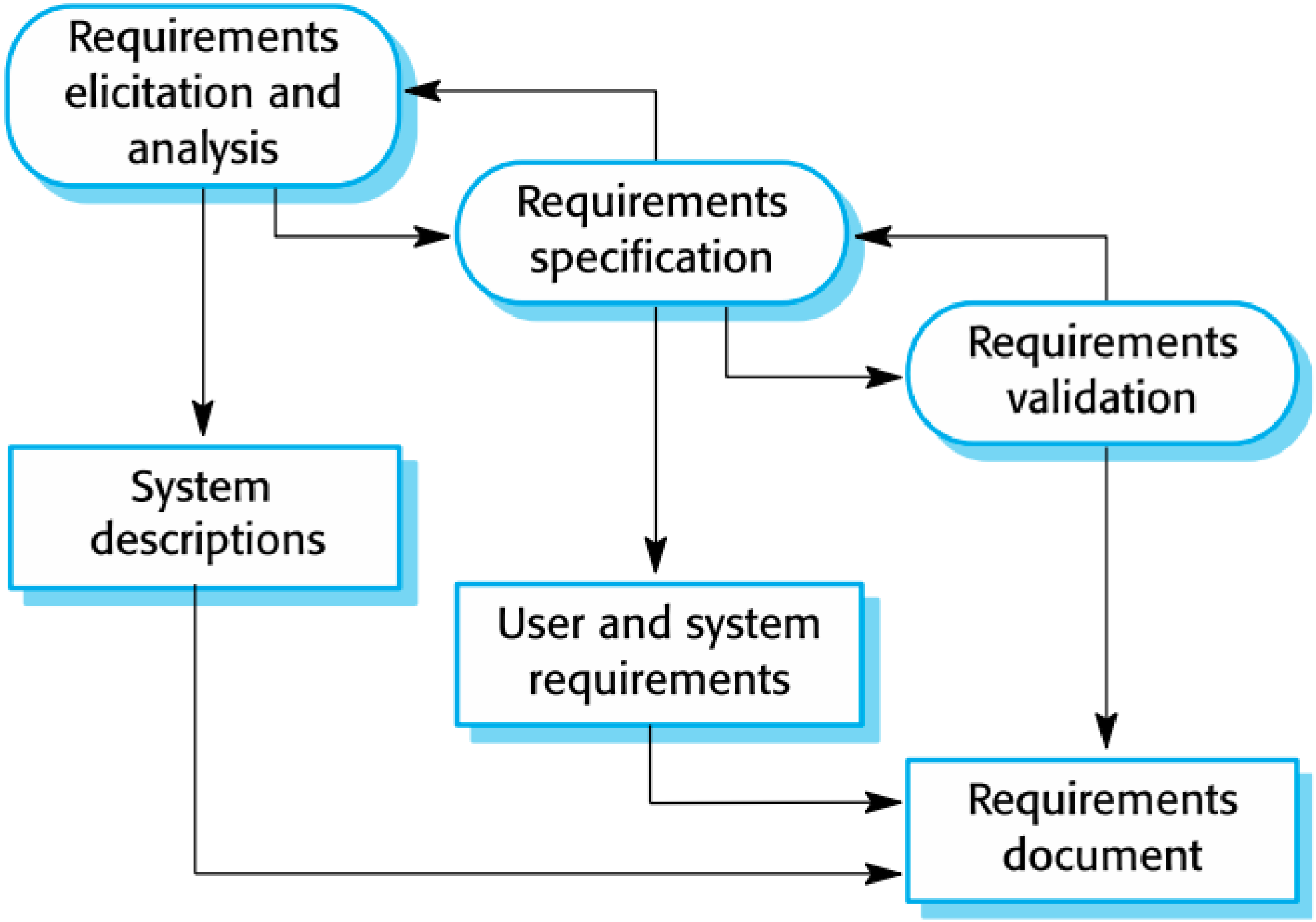
Stakeholders in SE

- Customers (Project Owners)
  - Those who pay for the software
- Users
  - Those who use the software
- Software Developers
  - Those who create and maintain the software
- Project Managers
  - Those who supervise the software development process

Requirements Gathering

- The process of establishing what services are required and the constraints on the system’s operation and development
- Determining what customers want and need from software
  - Problem space, not solution space
  - May include quality attributes
    - Performance, security, maintainability
- Challenges
  - Customers don’t know what they want
  - Customers can’t express what they want
  - Customers often change what they want

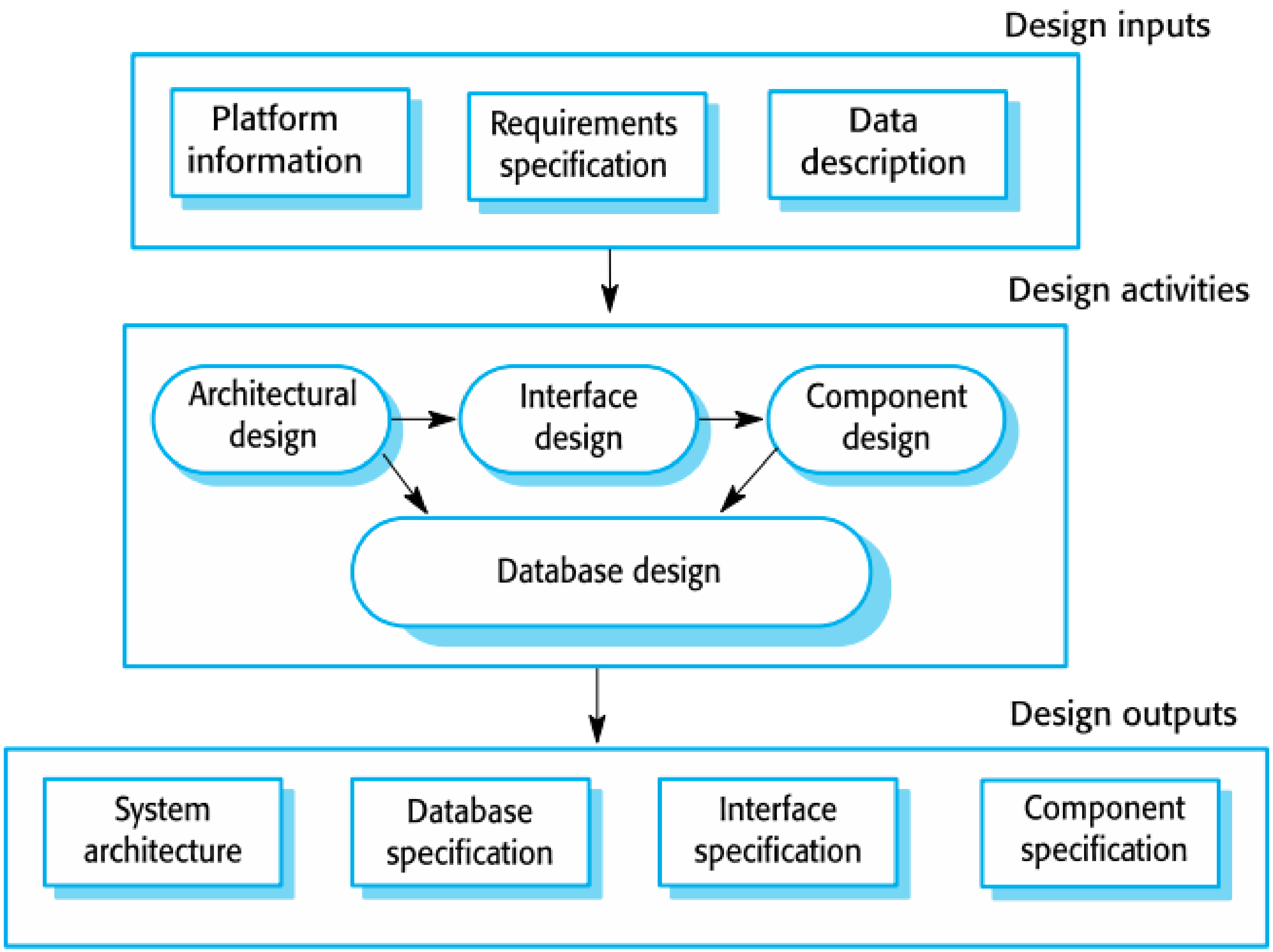
Requirement Engineering


Requirements elicitation and analysis
Requirements specification
Requirements validation
System descriptions
User and system requirements
Requirements document

Software Design

- Engineering solution that addresses requirements
- Designs include
  - Architecture
  - Code interfaces
  - User interfaces
  - Components
  - Data structures
  - Algorithms

Design Process



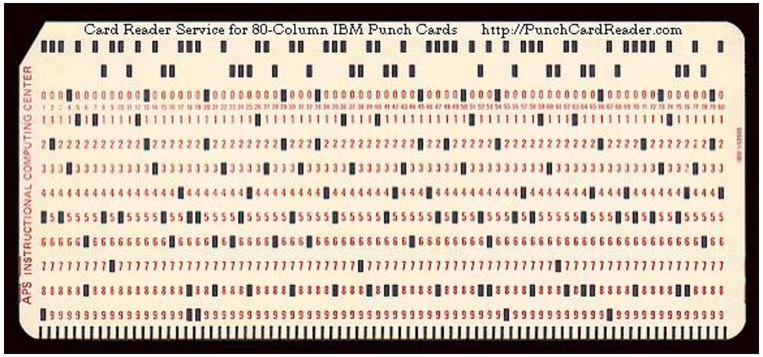
To Design or Not To Design

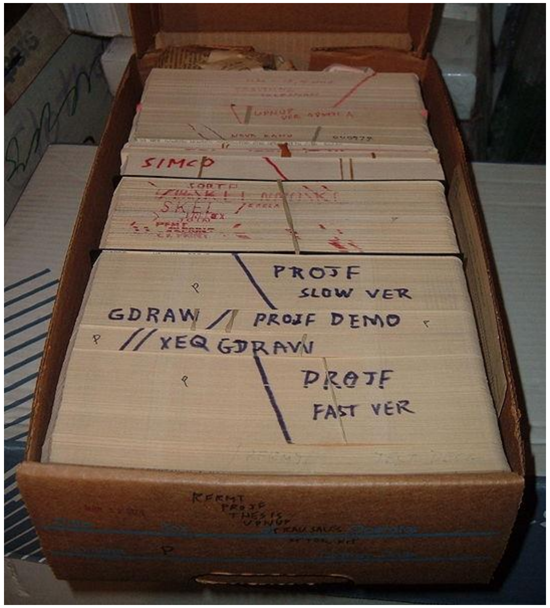
- *"When I learned to program, you were lucky if you got five minutes with the machine a day. If you wanted to get the program going, it just had to be written right. So people just learned to program like it was carving stone. You sort of have to sidle up to it. That's how I learned to program.“ --- Donald Knuth*
  - *http://www-cs-faculty.stanford.edu/~uno/*

- The Art of Computer Programming
  - *http://www-cs-faculty.stanford.edu/~uno/taocp.html*

To Design or Not To Design

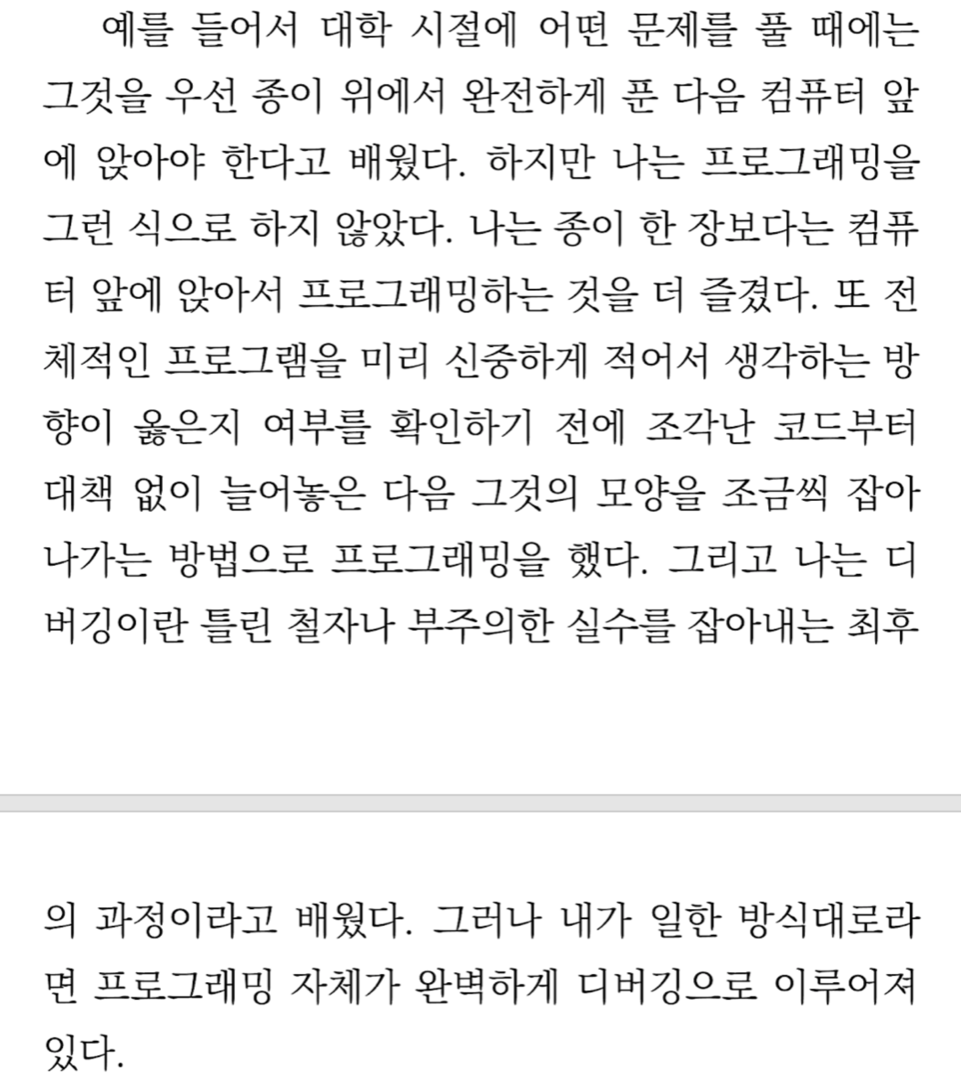
- Software != Write Once Read Many


Card Reader Service for 80-Column IBM Punch Cards http://PunchCardReader.com
APS INSTRUCTIONAL COMPUTING CENTER

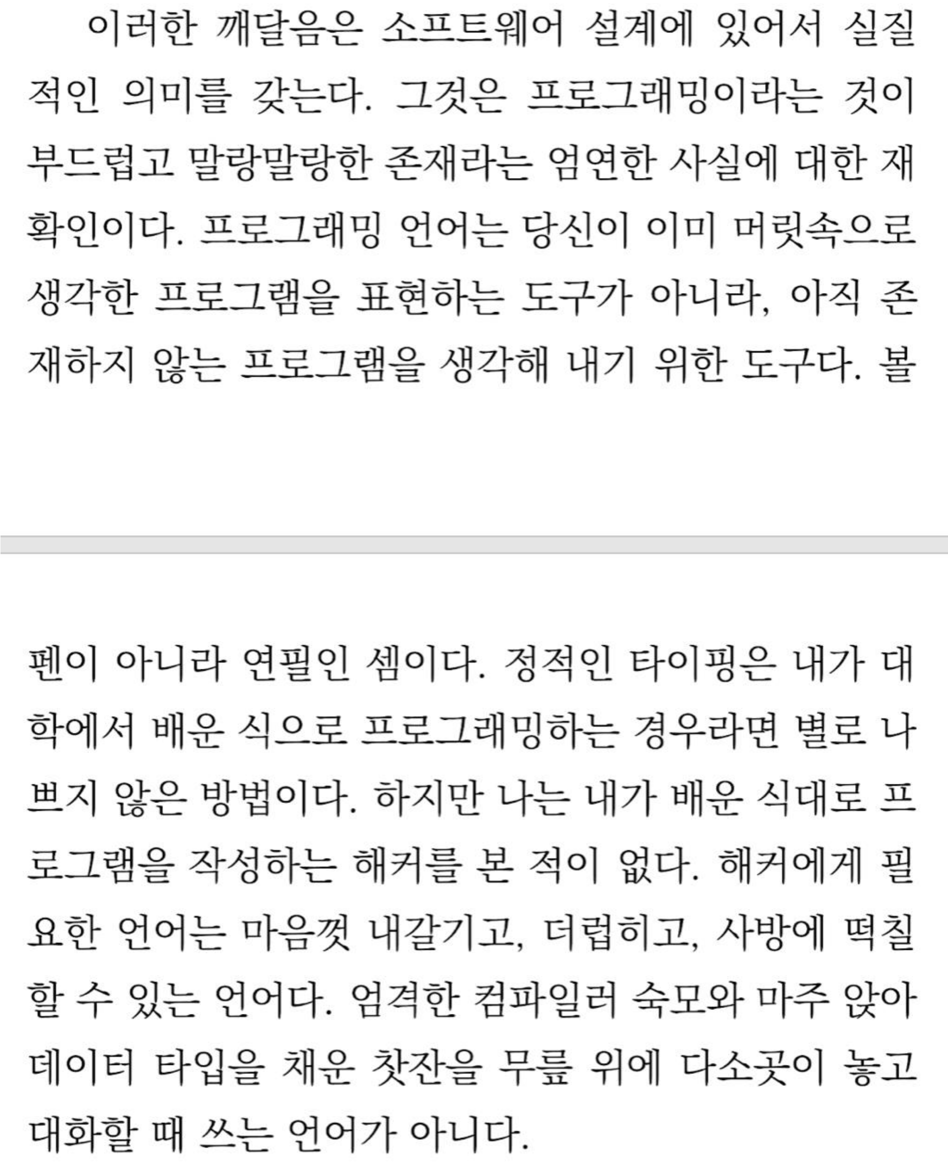


- Premature design--- deciding too early what a program should do.

Reading Material



예를 들어서 대학 시절에 어떤 문제를 풀 때에는 그것을 우선 종이 위에서 완전하게 푼 다음 컴퓨터 앞에 앉아야 한다고 배웠다. 하지만 나는 프로그래밍을 그런 식으로 하지 않았다. 나는 종이 한 장보다는 컴퓨터 앞에 앉아서 프로그래밍하는 것을 더 즐겼다. 또 전체적인 프로그램을 미리 신중하게 적어서 생각하는 방향이 옳은지 여부를 확인하기 전에 조각난 코드부터 대책 없이 늘어놓은 다음 그것의 모양을 조금씩 잡아 나가는 방법으로 프로그래밍을 했다. 그리고 나는 디버깅이란 틀린 철자나 부주의한 실수를 잡아내는 최후의 과정이라고 배웠다. 그러나 내가 일한 방식대로라면 프로그래밍 자체가 완벽하게 디버깅으로 이루어져 있다.



이러한 깨달음은 소프트웨어 설계에 있어서 실질적인 의미를 갖는다. 그것은 프로그래밍이라는 것이 부드럽고 말랑말랑한 존재라는 엄연한 사실에 대한 재확인이다. 프로그래밍 언어는 당신이 이미 머릿속으로 생각한 프로그램을 표현하는 도구가 아니라, 아직 존재하지 않는 프로그램을 생각해 내기 위한 도구다. 볼펜이 아니라 연필인 셈이다. 정적인 타이핑은 내가 대학에서 배운 식으로 프로그래밍하는 경우라면 별로 나쁘지 않은 방법이다. 하지만 나는 내가 배운 식대로 프로그램을 작성하는 해커를 본 적이 없다. 해커에게 필요한 언어는 마음껏 내갈기고, 더럽히고, 사방에 떡칠할 수 있는 언어다. 엄격한 컴파일러 숙모와 마주 앉아 데이터 타입을 채운 찻잔을 무릎 위에 다소곳이 놓고 대화할 때 쓰는 언어가 아니다.

Development (Coding)

- Realize a design in code
- Continue refining the designs
- Test the code to make sure it doesn’t contain any bugs

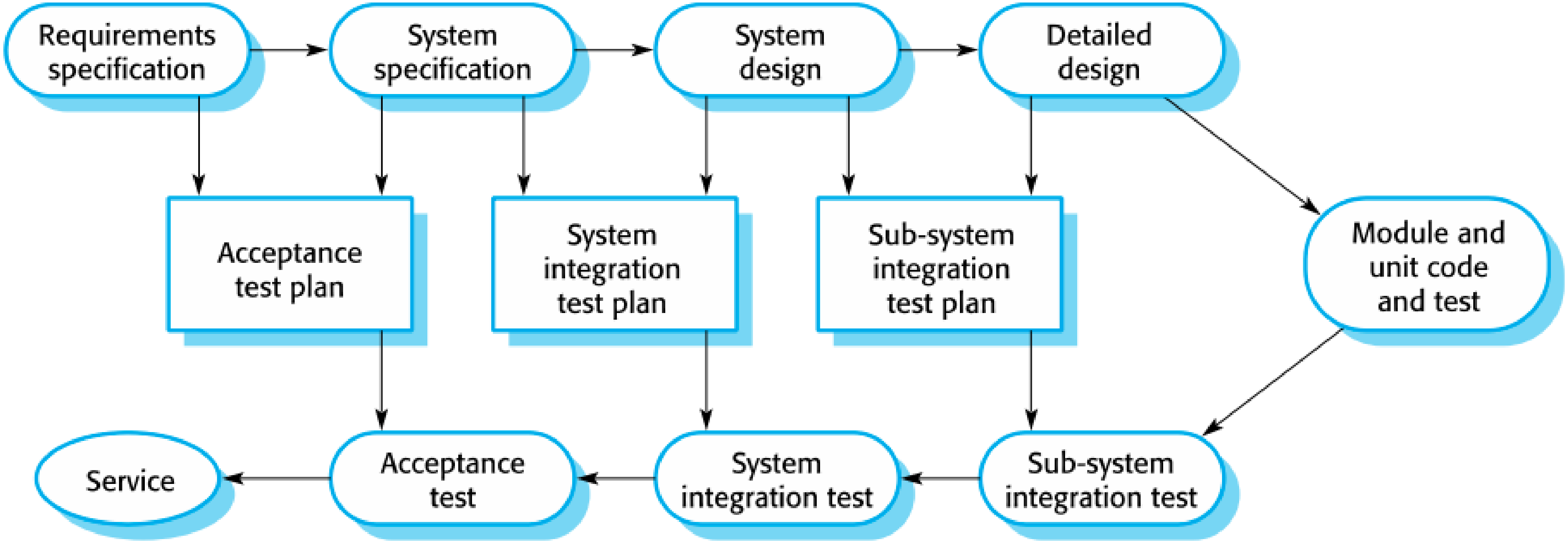
Testing (Quality Assurance)

- Finding bugs
  - *If your software passes all tests and you don’t find anything wrong, that doesn’t mean there are no bugs*
- QA: ensuring the implementation meets quality standards
  - Availability, modifiability, performance, scalability, security, testability, usability, efficiency, reliability, maintainability, reusability, etc.

Stages of Testing

- Component testing
  - – Individual components are tested independently;
  - – Components may be functions or objects or coherent groupings of these entities.
- System testing
  - – Testing of the system as a whole. Testing of emergent properties is particularly important.
- Customer testing
  - – Testing with customer data to check that the system meets the customer’s needs.

Testing Phases in Plan-driven Development

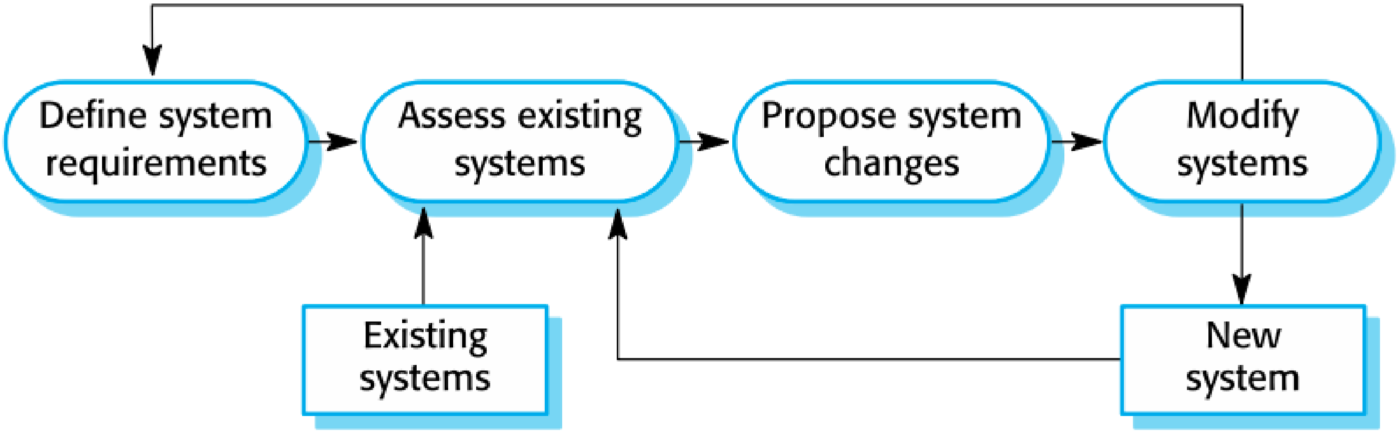

Requirements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and test
Service
Acceptance test
System integration test
Sub-system integration test

Deployment

- Rolling out software!

Maintenance

- Fixing bugs
- Enhancements
- Improvements


Define system requirements
Assess existing systems
Propose system changes
Modify systems
Existing systems
New system