SWCON201
Opensource & Software
Development Methods and Tools

Project Planning & Management

Department of
Software Convergence

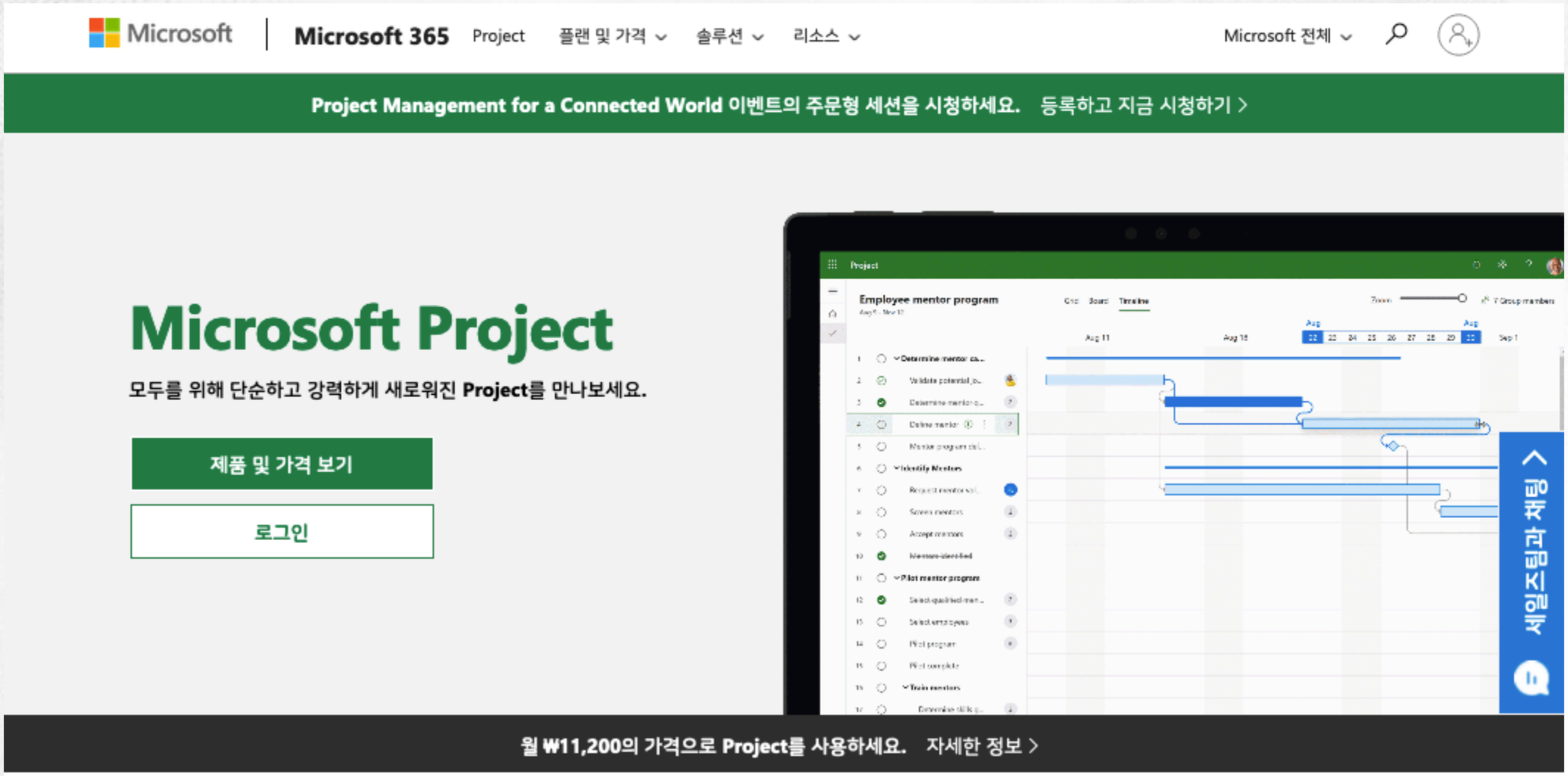
경희대학교
KYUNG HEE UNIVERSITY

Agenda

- Intro to project planning & management
  - https://www.youtube.com/watch?v=MRaNpy3KTZs
- How do you feel?
- When & where is it required?

Agenda

- MS Project ? Let's visit site !
  - https://products.office.com/ko-kr/project/project-and-portfolio-management-software?tab=tabs-1



간편하게 정리하고, 업무에 집중하며, 책임지고 관리할 수 있습니다. 소규모 프로젝트 부터 대규모 이니셔티브까지 모든 작업을 다룰 수 있습니다. 실제로 맡은 역할에 관계 없이 강력하고 간단한 앱을 활용하여 모든 프로젝트의 관리자가 될 수 있습니다.

Man-Month

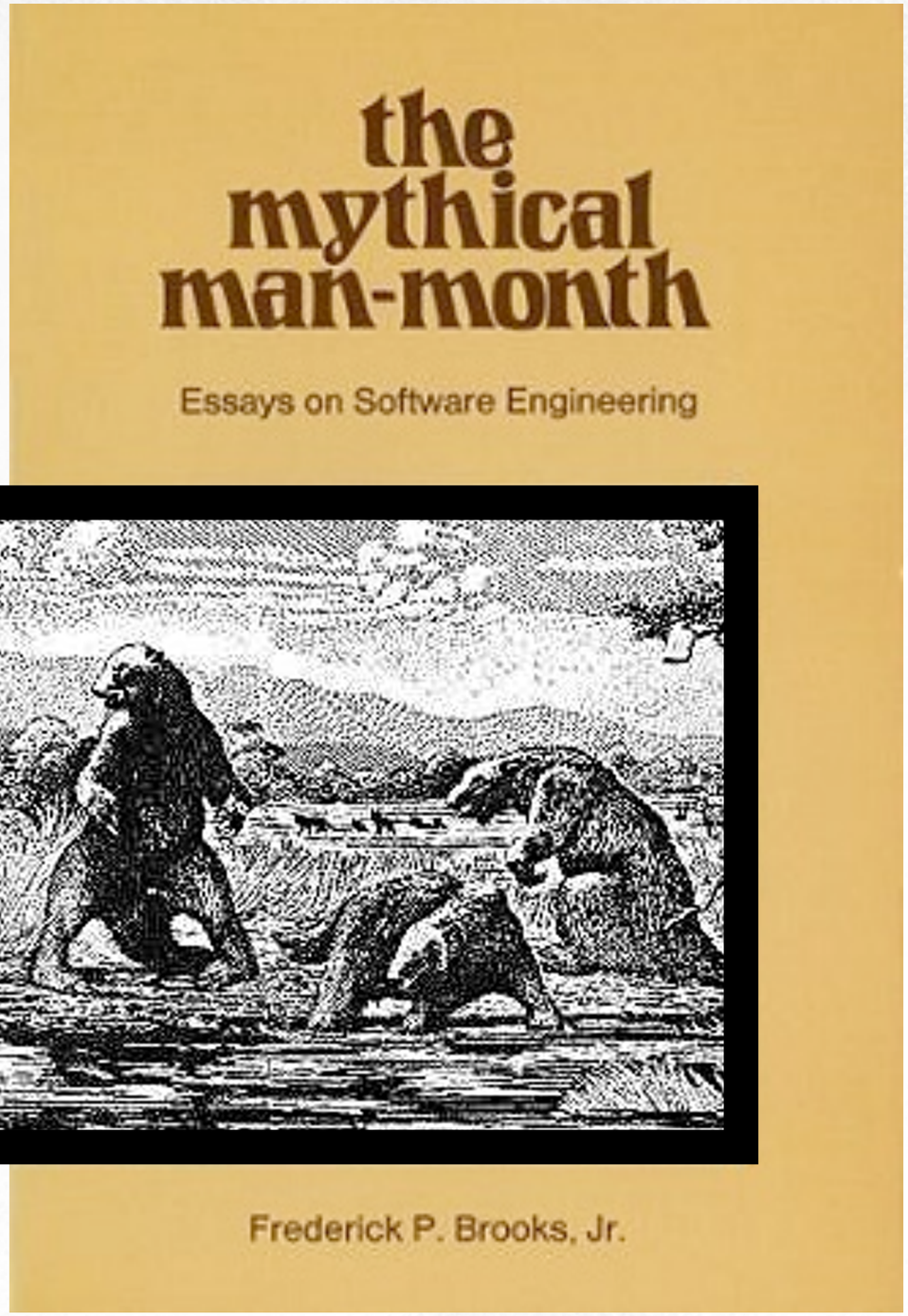
- Noun,
  - man-month (plural man-months),
  - One person's working time for a month, or the equivalent, used as a measure of how much work or labor is required or consumed to perform some task.

- Example:
  - 1 full person per month is 1 M/M
  - 1 full person per year is 12 M/Ms

Reference: https://en.wiktionary.org/wiki/man-month

Man-Month

- "Adding manpower to a late software project makes it later"



Reference: https://en.wikipedia.org/wiki/The_Mythical_Man-Month

Man-Month

- "Adding manpower to a late software project makes it later"

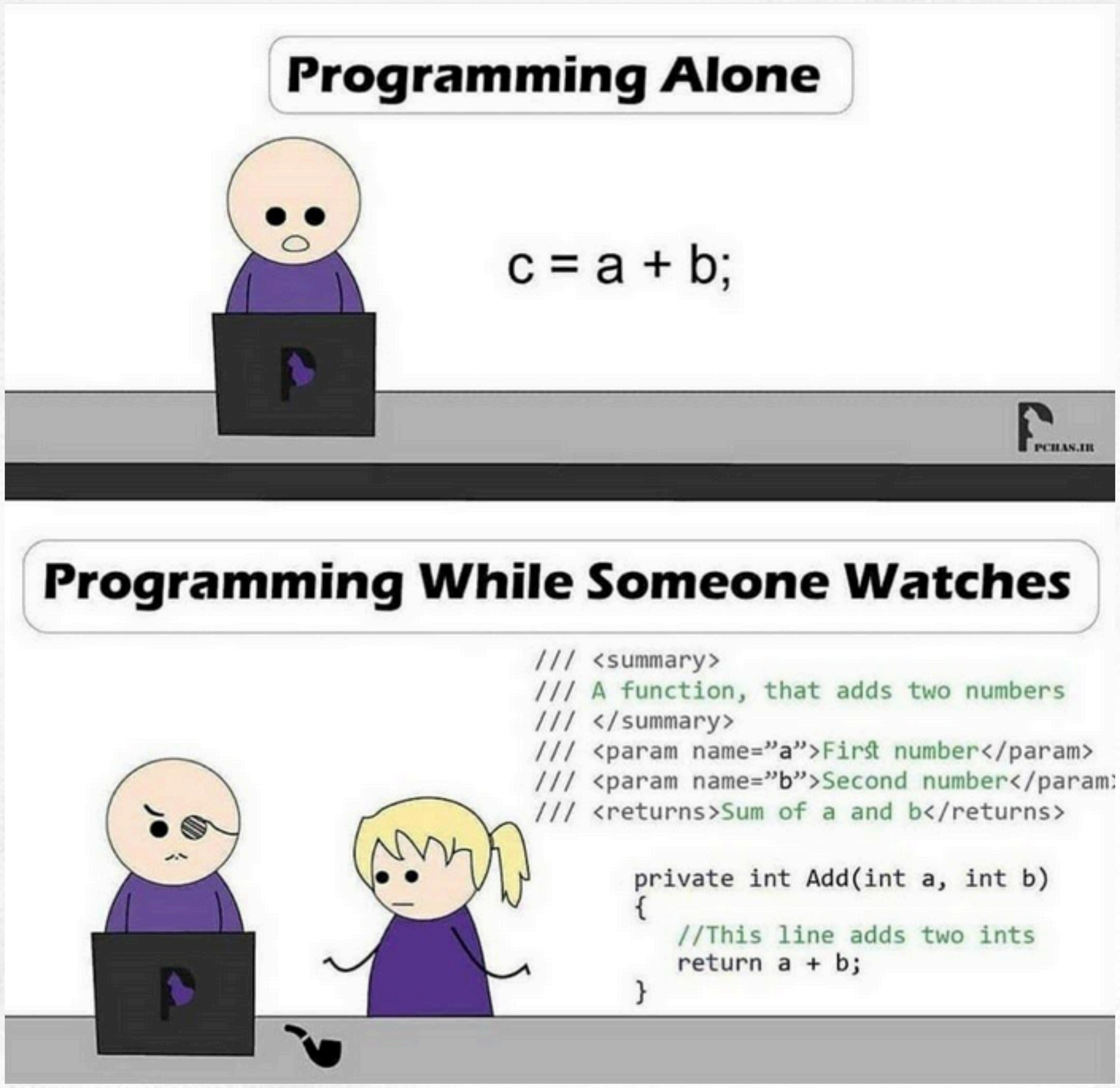
Reference: https://www.sachinrekhi.com/solving-for-the-mythical-man-month

Man-Month

- "Adding manpower to a late software project makes it later"

Man-Month

- "Adding manpower to a late software project makes it later"



Reference: https://programmerhumour.tumblr.com/image/617272639161024512

Man-Month

- "Adding manpower to a late software project makes it later”
  - From ‘The Mythical Man-Month’: Essays on Software Engineering is a book on software engineering and project management by Fred Brooks first published in 1975, with subsequent editions in 1982 and 1995
  - Its central theme is that "adding manpower to a late software project makes it later"
  - This idea is known as Brooks's law, and is presented along with the second-system effect and advocacy of prototyping

The Mythical Man-Month

- Brooks' observations are based on his experiences at IBM while managing the development of OS/360
- He had added more programmers to a project falling behind schedule, a decision that he would later conclude had, counter-intuitively, delayed the project even further
- He also made the mistake of asserting that one project—involved in writing an ALGOL compiler—would require six months, regardless of the number of workers involved (it required longer)
- The tendency for managers to repeat such errors in project development led Brooks to quip that his book is called "The Bible of Software Engineering", because "everybody quotes it, some people read it, and a few people go by it". The book is widely regarded as a classic on the human elements of software engineering.

The Mythical Man-Month

- Brooks discusses several causes of scheduling failures
- The most enduring is his discussion of Brooks's law: Adding manpower to a late software project makes it later
- Man-month is a hypothetical unit of work representing the work done by one person in one month; **Brooks' law says that the possibility of measuring useful work in man-months is a myth**, and is hence the centerpiece of the book
- Complex programming projects cannot be perfectly partitioned into discrete tasks that can be worked on without communication between the workers and without establishing a set of complex interrelationships between tasks and the workers performing them

The Mythical Man-Month

- Therefore, assigning more programmers to a project running behind schedule will make it even later
- This is because the time required for the new programmers to learn about the project and the increased communication overhead will consume an ever increasing quantity of the calendar time available
- When n people have to communicate among themselves, as n increases, their output decreases and when it becomes negative the project is delayed further with every person added
- Group intercommunication formula: n(n – 1) / 2
  - Example: 50 developers give 50 · (50 − 1) / 2 = 1225 channels of communication.

No silver bullet

- Brooks added "No Silver Bullet — Essence and Accidents of Software Engineering"—and further reflections on it, "'No Silver Bullet' Refired"—to the anniversary edition of The Mythical Man-Month
- Brooks insists that there is no one silver bullet -- "there is no single development, in either technology or management technique, which by itself promises even one order of magnitude [tenfold] improvement within a decade in productivity, in reliability, in simplicity."
- The argument relies on the distinction between accidental complexity and essential complexity, similar to the way Amdahl's law relies on the distinction between "strictly serial" and "parallelizable"

The second-system effect

- The second-system effect proposes that, when an architect designs a second system, it is the most dangerous system they will ever design, because they will tend to incorporate all of the additions they originally did not add to the first system due to inherent time constraints
- Thus, when embarking on a second system, an engineer should be mindful that they are susceptible to over-engineering it

Reference: https://en.wikipedia.org/wiki/Software_design_pattern

Irreducible number of errors

- The tendency towards irreducible number of errors

  99 little bugs in the code.
  99 little bugs.
  Take one down, patch it around.
  127 little bugs in the code...

- The author makes the observation that in a suitably complex system there is a certain irreducible number of errors
- Any attempt to fix observed errors tends to result in the introduction of other errors

Progress tracking

- Brooks wrote

  Question: How does a large software project get to be one year late?
  Answer: One day at a time!

- Incremental slippages on many fronts eventually accumulate to produce a large overall delay
- Continued attention to meeting small individual milestones is required at each level of management

Conceptual integrity

- To make a user-friendly system, the system must have conceptual integrity, which can only be achieved by separating architecture from implementation
- A single chief architect (or a small number of architects), acting on the user's behalf, decides what goes in the system and what stays out
- The architect or team of architects should develop an idea of what the system should do and make sure that this vision is understood by the rest of the team
- A novel idea by someone may not be included if it does not fit seamlessly with the overall system design
- In fact, to ensure a user-friendly system, a system may deliberately provide fewer features than it is capable of
- The point being, if a system is too complicated to use, many features will go unused because no one has time to learn them.

The manual

- The chief architect produces a manual of system specifications
- It should describe the external specifications of the system in detail, i.e., everything that the user sees
- The manual should be altered as feedback comes in from the implementation teams and the users

The pilot system

- When designing a new kind of system, a team will design a throw-away system (whether it intends to or not)
- This system acts as a "pilot plant" that reveals techniques that will subsequently cause a complete redesign of the system
- This second, smarter system should be the one delivered to the customer, since delivery of the pilot system would cause nothing but agony to the customer, and possibly ruin the system's reputation and maybe even the company

Formal documents

- Every project manager should create a small core set of formal documents defining the project objectives, how they are to be achieved, who is going to achieve them, when they are going to be achieved, and how much they are going to cost
- These documents may also reveal inconsistencies that are otherwise hard to see

Reference: https://en.wikipedia.org/wiki/Software_design_pattern

Project estimation

- When estimating project times, it should be remembered that programming products (which can be sold to paying customers) and programming systems are both three times as hard to write as simple independent in-house programs
- It should be kept in mind how much of the work week will actually be spent on technical issues, as opposed to administrative or other non-technical tasks, such as meetings, and especially "stand-up" or "all-hands" meetings

Communication

- To avoid disaster, all the teams working on a project should remain in contact with each other in as many ways as possible— email, phone, meetings, memos etc
- Instead of assuming something, implementers should ask the architect(s) to clarify their intent on a feature they are implementing, before proceeding with an assumption that might very well be completely incorrect
- The architect(s) are responsible for formulating a group picture of the project and communicating it to others

The surgical team

- Much as a surgical team during surgery is led by one surgeon performing the most critical work, while directing the team to assist with less critical parts, it seems reasonable to have a "good" programmer develop critical system components while the rest of a team provides what is needed at the right time
- Additionally, Brooks muses that "good" programmers are generally five to ten times as productive as mediocre ones

Code freeze and system versioning

- Software is invisible
- Therefore, many things only become apparent once a certain amount of work has been done on a new system, allowing a user to experience it
- This experience will yield insights, which will change a user's needs or the perception of the user's needs
- The system should, therefore, be changed to fulfill the changed requirements of the user
- This can only occur up to a certain point, otherwise the system may never be completed
- At a certain date, no more changes should be allowed to the system and the code should be frozen
- All requests for changes should be delayed until the next version of the system

Specialized tools

- Instead of every programmer having his own special set of tools, each team should have a designated tool-maker who may create tools that are highly customized for the job that team is doing, e.g., a code generator tool that creates code based on a specification
- In addition, system-wide tools should be built by a common tools team, overseen by the project manager

Reference: https://en.wikipedia.org/wiki/Software_design_pattern

Lowering software development costs

- IThere are two techniques for lowering software development costs that Brooks writes about:
  - Implementers may be hired only after the architecture of the system has been completed (a step that may take several months, during which time prematurely hired implementers may have nothing to do)
  - Another technique Brooks mentions is not to develop software at all, but simply to buy it "off the shelf" when possible

Tools

- ProjectLibre
  - Homepage: http://www.projectlibre.com/
  - Tutorial: https://www.youtube.com/watch?v=bp0YdQ7gfwE

- Other Tools
  - https://opensource.com/business/13/5/open-project-interview
  - https://thedigitalprojectmanager.com/creative-project-management-software/
  - https://opensource.com/business/16/3/top-project-management-tools-2016

Summary

- We already learned Agile.

- But we need Structured approach some times.
  - Man-Month
  - Ghant-Chart
  - WBS (Work Breakdown Structure)

Summary

- Direct and indirect experience enables correct project planning and management.

  Read,

  Experience,

  Don’t repeat the same thing.

Reference

- The Mythical Man-Month : Korean Version

  http://egloos.zum.com/gyumee/v/1021765

Thank you

경희대학교
KYUNG HEE UNIVERSITY