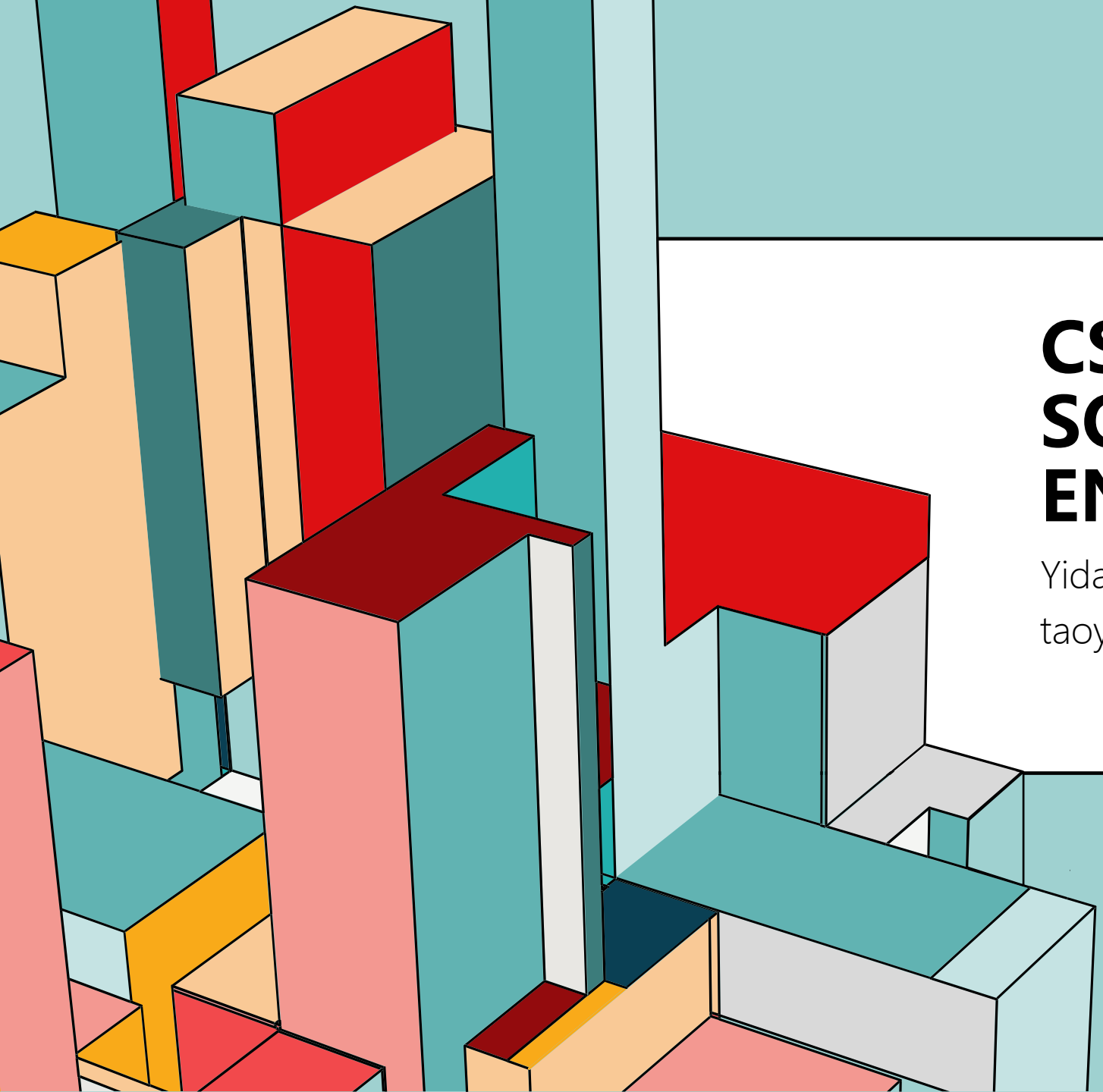


南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CS304 SOFTWARE ENGINEERING

Yida Tao
taoyd@sustech.edu.cn

BEFORE YOU ENROLL

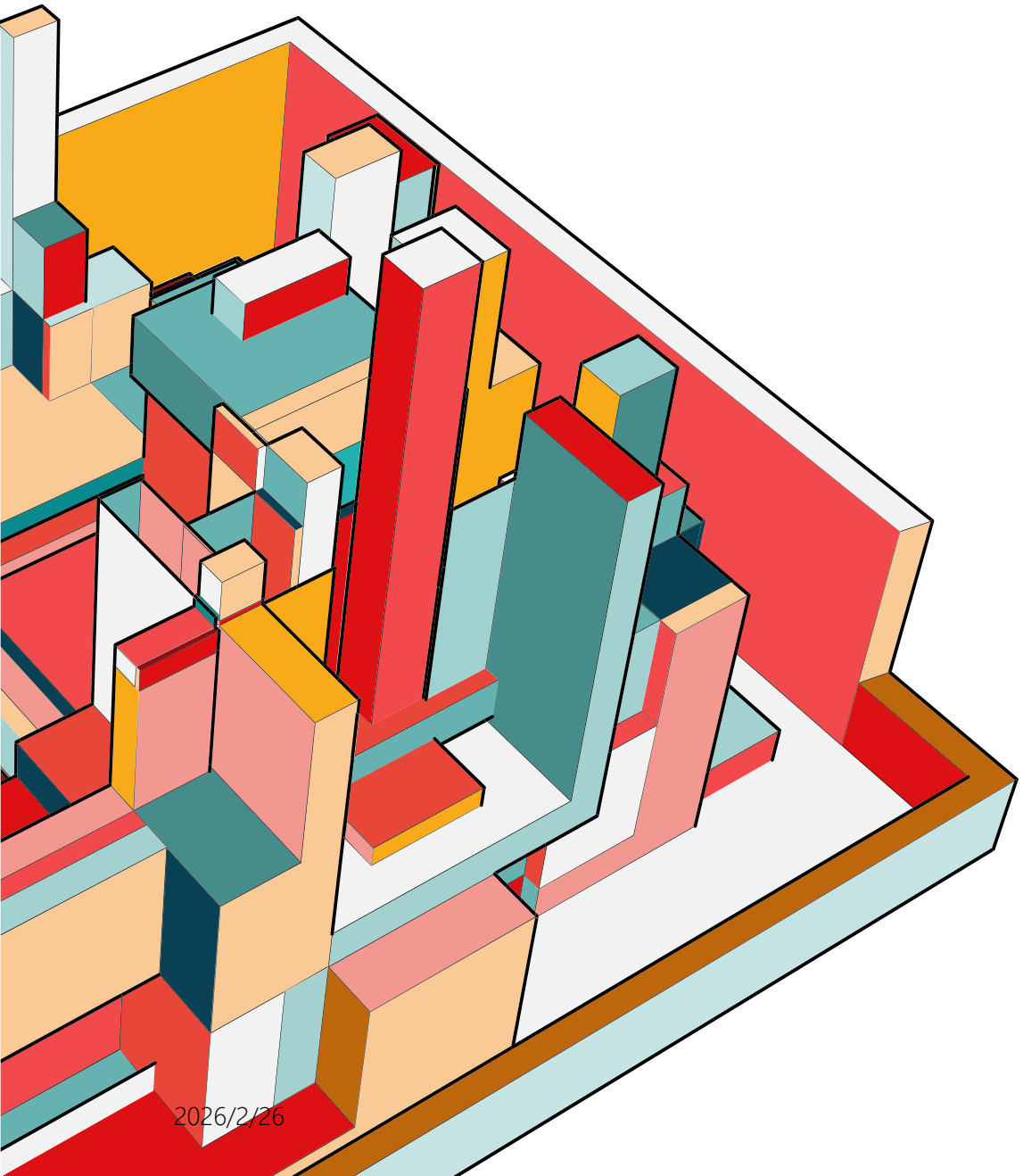
Prerequisite

Introduction to computer programming

Strongly recommended background

Advanced programming (CS209A, OOAD), Data structure & algorithms

- Easier understanding of SE concepts
- Better connection to real-world problems
- More manageable assignments and project



LECTURE 1

- What is software?
- How to build a software?
- What is software engineering?
- Objectives of software engineering?
- What will we learn?

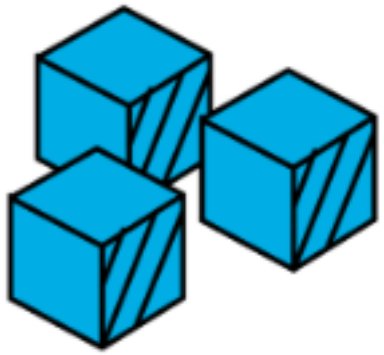
Q1. WHAT IS SOFTWARE?

Is programming assignment a software?

ARTIFACTS

Programming Assignment

- Source code



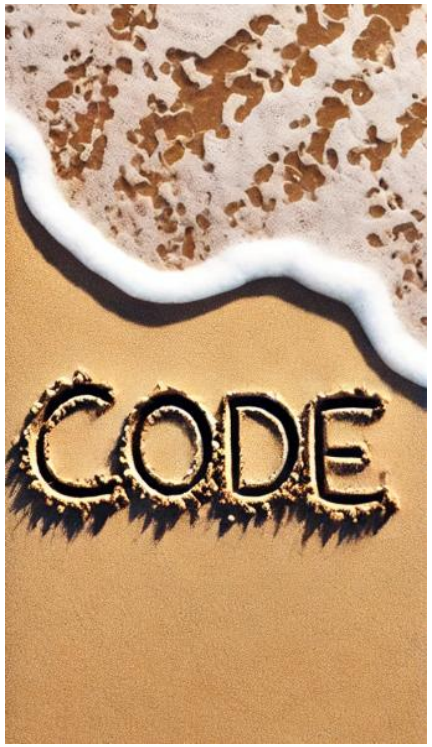
Software

- Source code
- Tests
- Documentation
- Build scripts
- Configurations
- Executables (.exe etc.)
- ……

WHAT'S THE EXPECTED LIFE SPAN OF YOUR CODE?

Programming Assignment

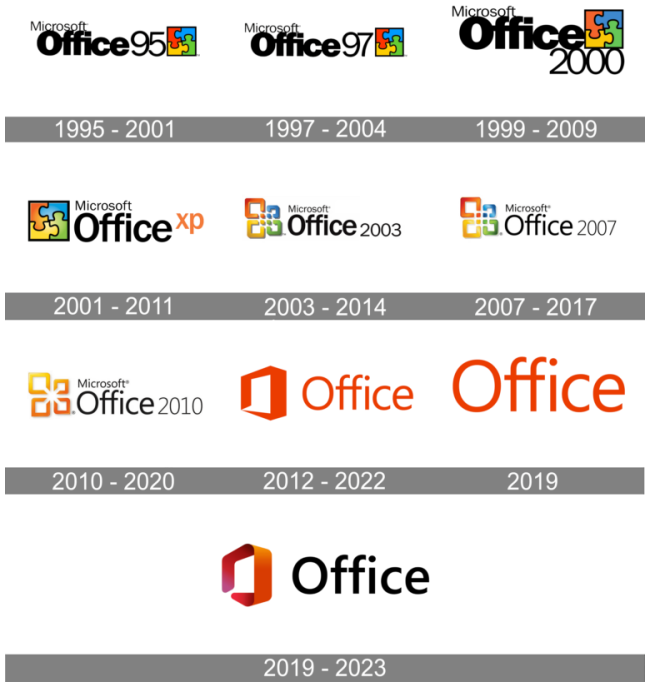
- **Short-term**: your programming code is likely to last for only hours, days, or weeks, not any longer (i.e., decades)
- **No-change**: You probably won't upgrade and maintain your programming code after the assignment deadline ☺



WHAT'S THE EXPECTED LIFE SPAN OF YOUR CODE?

Software

- **Long-term**: large software (e.g., Microsoft Office, Google Search) tend to live for decades
- **Adapt-to-change**: to allow for longer life spans, software needs to adapt to new versions of underlying dependencies, OS, hardware, programming language versions, etc.



HOW MANY RESOURCES ARE INVOLVED?

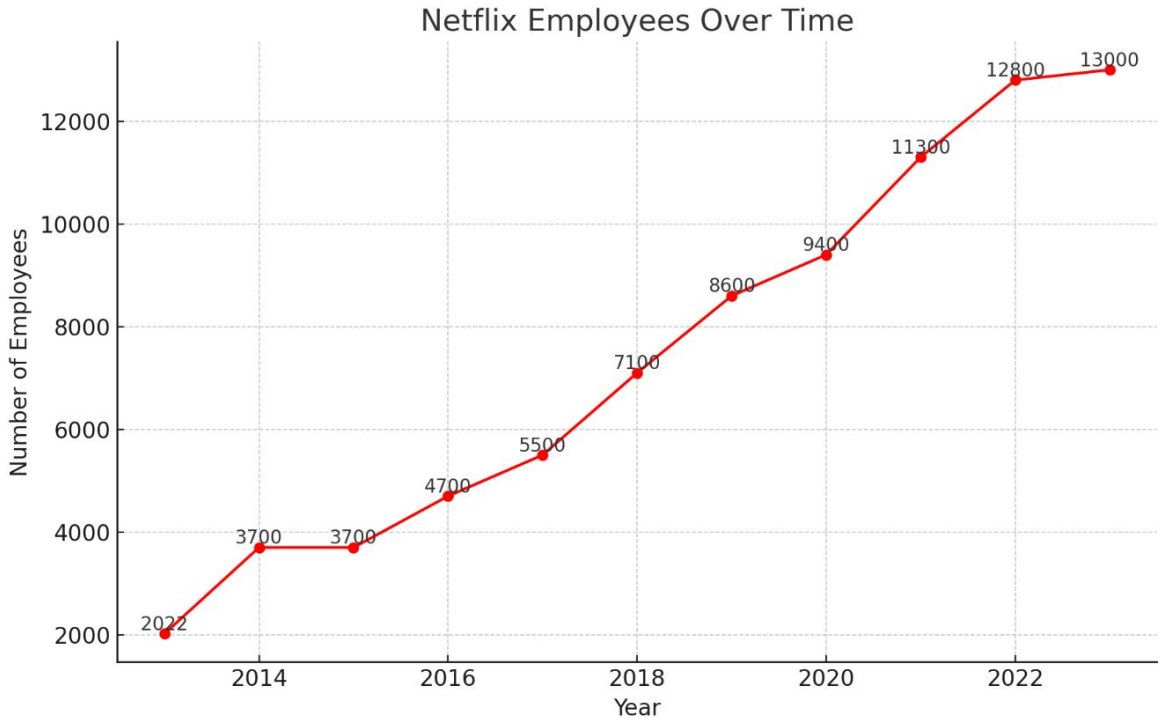
Programming Assignment

- **Human resources**: solo or 2-3 sized small groups
- **Computing resources**: a single laptop is typically sufficient

HOW MANY RESOURCES ARE INVOLVED?

Software

- **Human resources**: large software is developed and maintained by (large) teams
- Computing resources: as organization and users grow, large software needs to scale well with compute, memory, storage, bandwidth resources



HOW MANY RESOURCES ARE INVOLVED?

Software

- **Human resources**: large software is developed and maintained by (large) teams
- **Computing resources**: as organization and users grow, large software needs to **scale** well with compute, memory, storage, bandwidth resources

ChatGPT

ChatGPT: 1-year running cost of up to 475 million dollars

HOW MANY USERS?

Programming Assignment

- Yourself
- Your team members
- Teachers and TAs

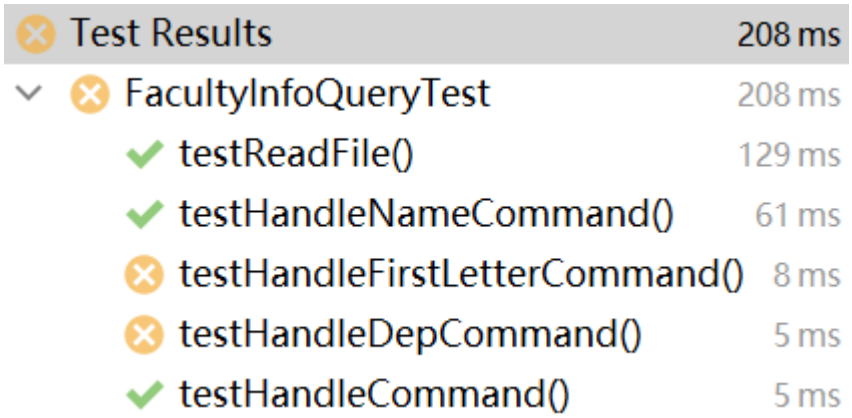
Software

Application	Overall Active Users
ChatGPT	180.5 million
DeepSeek	33.7 million
TikTok	1.56 billion
GitHub	100 million developers
Android	3.9 billion devices
Windows	1.6 billion devices

COMPLEXITY OF DECISIONS

Programming Assignment

- Correctness
- Time (e.g., deadline)



Software

- Software quality
- Engineering efforts
- Financial costs
- Computing resources
- Legal & ethical considerations
- Social impact
- …

Q2. HOW TO BUILD A SOFTWARE AND WHY IS IT HARD?

Building a software == coding?

THE BUILDING OF IBM OS/360

- Time: 1963-1966
- Human involved: 5000 man-month (one person’s working time for a month)
- Codebase: 1M lines of code
- Cost: hundreds of millions $

图灵奖得主、IBM 360系统之父
Frederick Brooks

THE BUILDING OF IBM OS/360

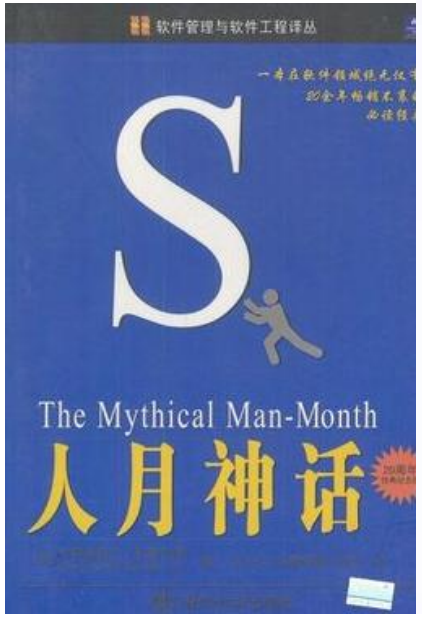
- Deferred releases
- Underestimated cost & memory resources
- Low-quality in first public release
- Thousands of bug fixes even after several releases

图灵奖得主、IBM 360系统之父
Frederick Brooks

THE BUILDING OF IBM OS/360

Software like a tar pit (焦油坑): The more you fight it, the deeper you sink!

……正像一只逃亡的野兽落到泥潭中做垂死的挣扎，越是挣扎，陷得越深， 最后无法逃脱灭顶的灾难。……程序设计工作正像这样一个泥潭，……一批批程序员被迫在泥潭中拼命挣扎，……谁也没有料到问题竟会陷入这样的困境……

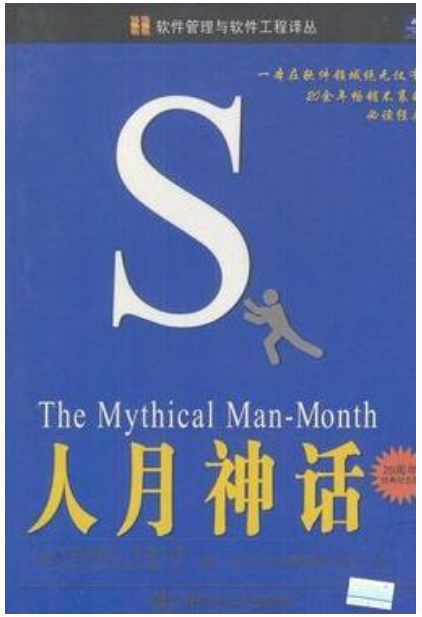


图灵奖得主、IBM 360系统之父
Frederick Brooks

TAKEAWAYS

“Adding manpower to a late software project makes it later.”

向进度落后的项目中增加人力，只会让项目更加落后。

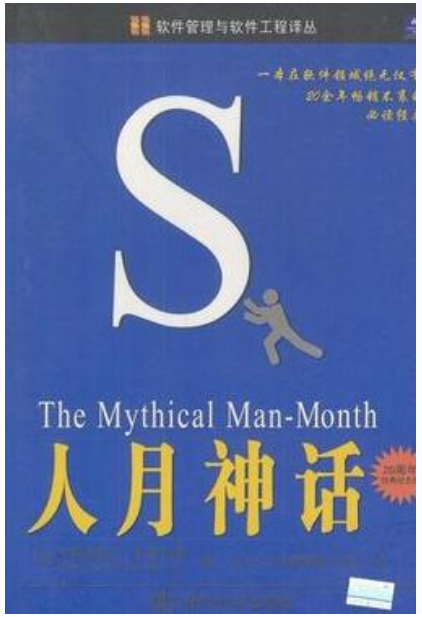


图灵奖得主、IBM 360系统之父
Frederick Brooks

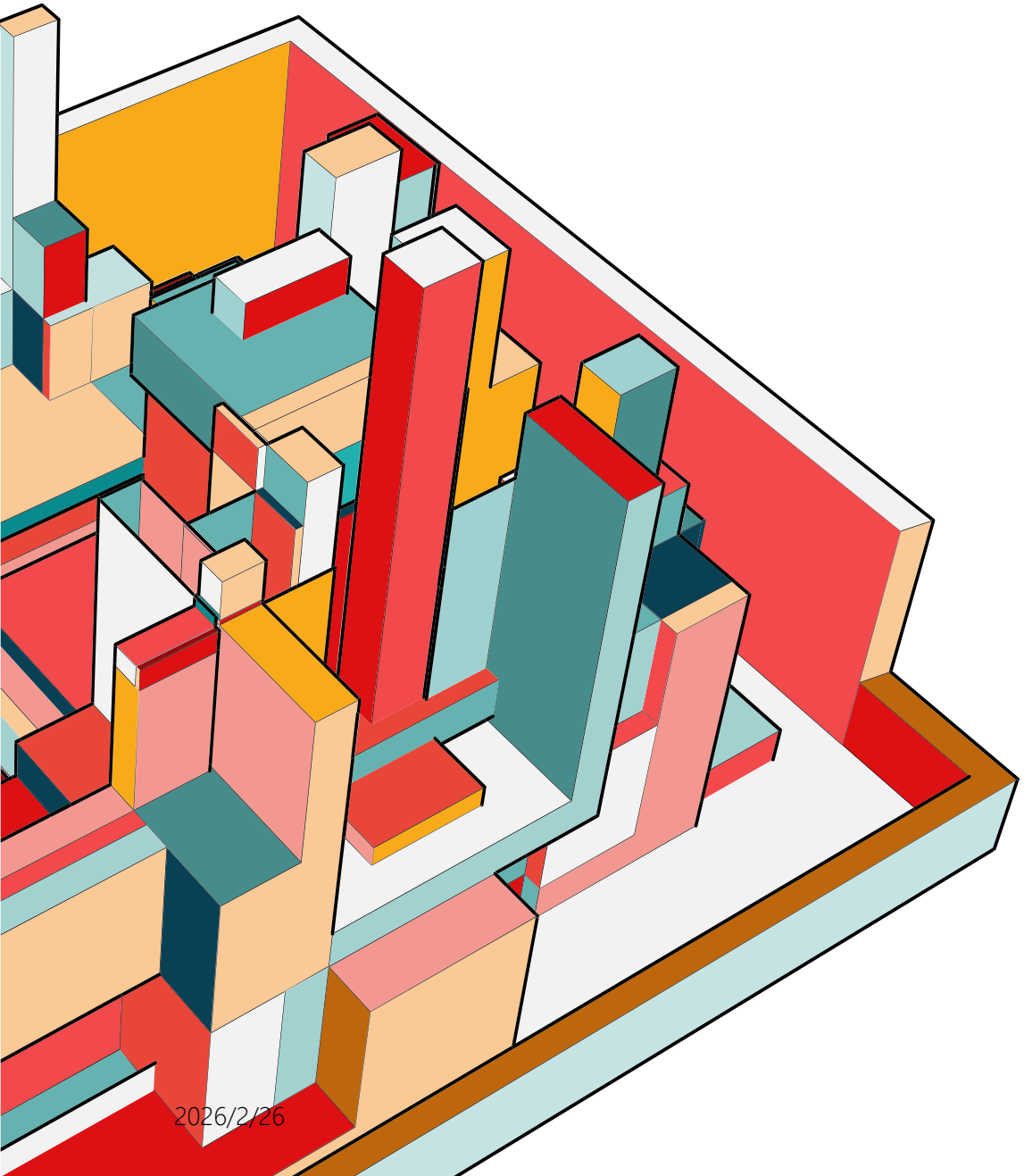
TAKEAWAYS

“The complexity of software is an essential property, not an accidental one. This **inherent complexity** is what makes software difficult to build and harder to scale.”

软件开发本质上是复杂的。这种本质复杂性使得软件开发困难且难以扩展



图灵奖得主、IBM 360系统之父
Frederick Brooks



COMPLEXITY FACTORS

- Time
- Scale
- Tradeoff

TIME FACTORS

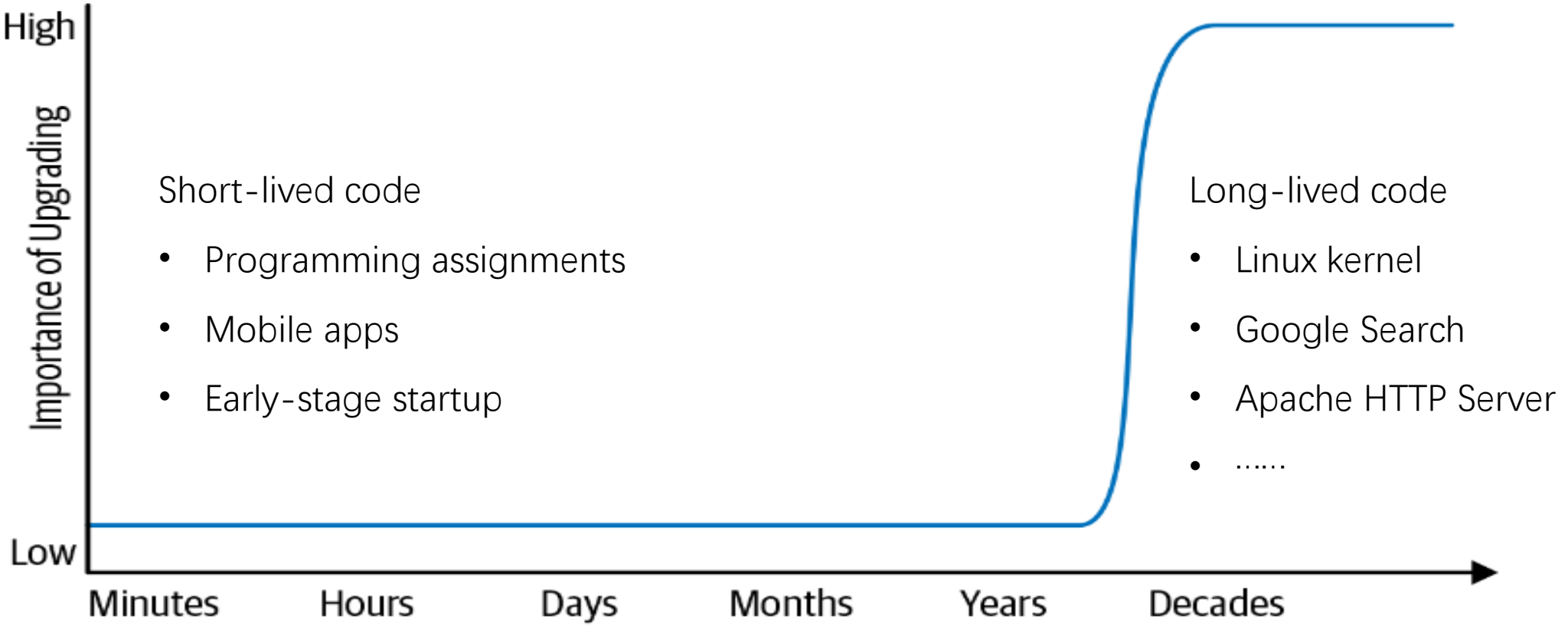
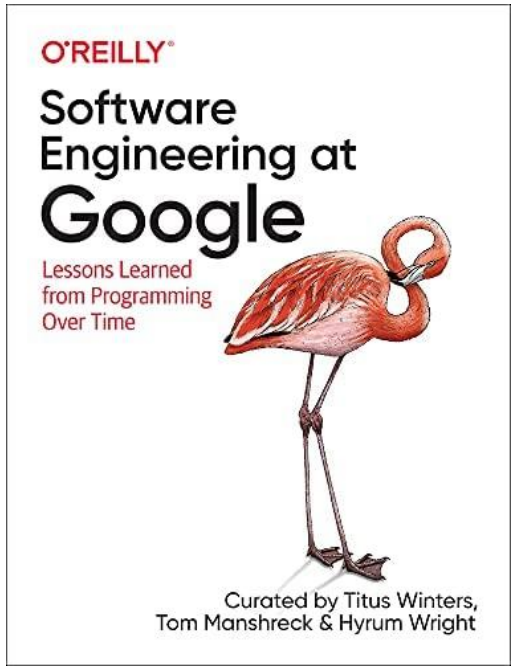


Figure 1-1. Life span and the importance of upgrades

TIME FACTORS



Hyrum's Law: Given enough time and enough users on a software system, someone will start depending on every little detail of how it behaves — even the parts you never planned to guarantee.

Because of these unexpected dependencies, even the smallest change will eventually break something, making it harder and harder to safely update the system.

EXAMPLE

Suppose your API outputs JSON:

```
{
  "id": 1,
  "name": "Alice",
  "email": "alice@example.com"
}
```

You, as the API designer, never promised any specific order of the keys.

User 1: Assumes the JSON key order is fixed

- User 1 observes that the keys always appear in the order: id, name, email.
- User 1 writes code that directly depends on this order.
- If the JSON library changes the order, User 1’s system immediately breaks.

EXAMPLE

Suppose your API outputs JSON:

```
{
  "id": 1,
  "name": "Alice",
  "email": "alice@example.com"
}
```

You, as the API designer, never promised any specific order of the keys.

User 2: Assumes the JSON key order is random and uses it

- User 2 assumes the order is random and misuses it as a random-number generator.
- If the order later becomes stable or predictable, User 2's system no longer works as expected.

EXAMPLE

Suppose your API outputs JSON:

```
{
  "id": 1,
  "name": "Alice",
  "email": "alice@example.com"
}
```

You, as the API designer, never promised any specific order of the keys.

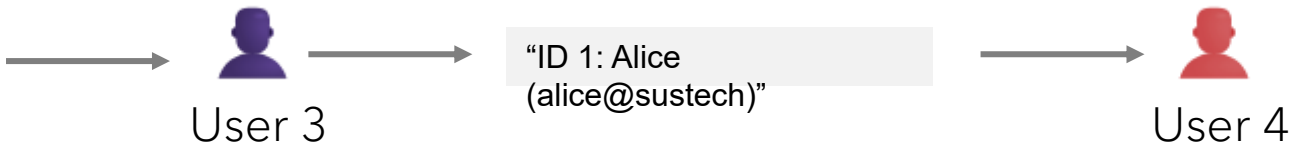
User 3: Does the right thing

- User 3 correctly ignores the key order and writes proper, order-independent JSON parsing.
- User 3's system remains stable even if the key order changes.
- At this point, User 3 is safe… or so it seems.

EXAMPLE

Suppose your API outputs JSON:

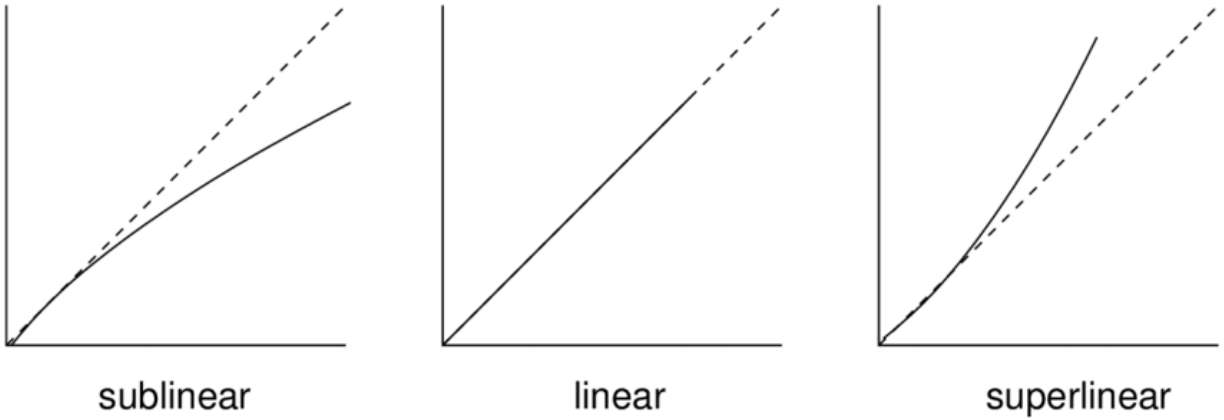
```
{
  "id": 1,
  "name": "Alice",
  "email": "alice@sustech"
}
```



You, as the API designer, never promised any specific order of the keys.

User 3's system converts the parsed JSON to a string, which is consumed by User 4.

SCALE



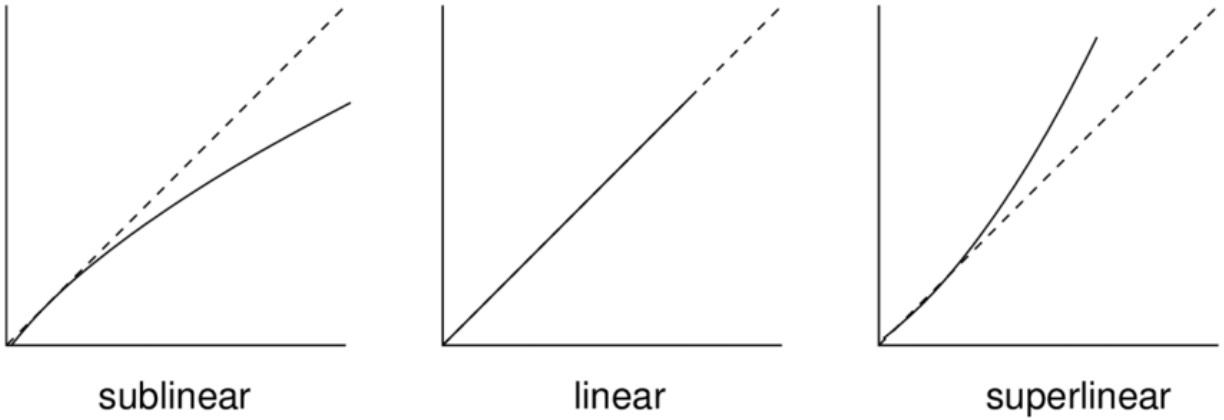
X-axis: workload (e.g., code size, data size, # users)

Y-axis: costs or resource consumption (e.g., execution time, memory usage, time to build, time for bug fixes)

Linear

- Predictable
- Ex. If you double the code size, you double the time needed to build or fix

SCALE



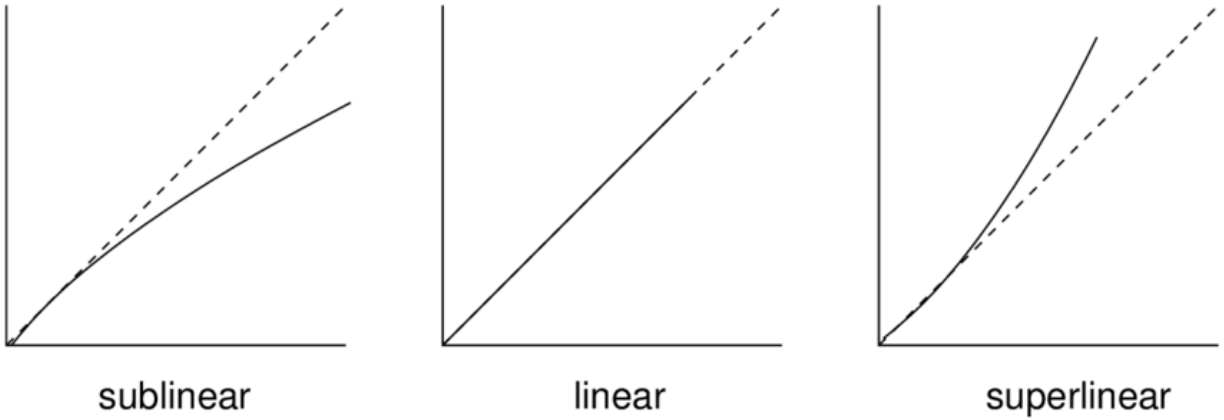
X-axis: workload (e.g., code size, data size, # users)

Y-axis: costs or resource consumption (e.g., execution time, memory usage, time to build, time for bug fixes)

Sublinear

- Ideal, **good scalability**
- Ex. User load increases 1000x while app response time increases only 2x

SCALE



X-axis: workload (e.g., code size, data size, # users)

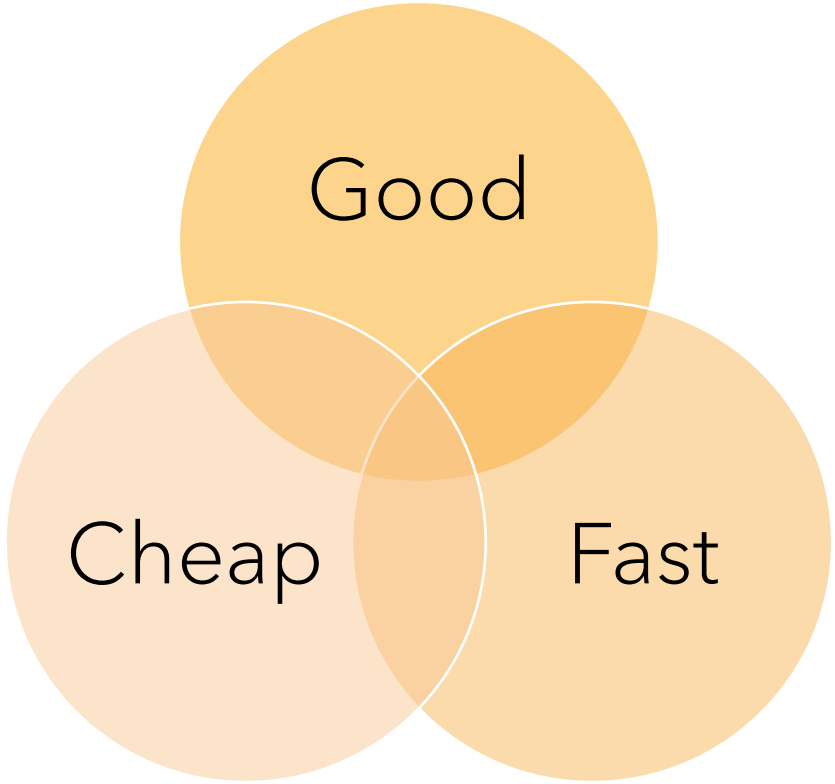
Y-axis: costs or resource consumption (e.g., execution time, memory usage, time to build, time for bug fixes)

Superlinear

- Dangerous, poor scalability
- Ex. adding 10% more code but needing 50% more time to fix or upgrade

TRADEOFF

- Speed vs. Quality
- Features vs. Simplicity
- Individual vs. Team schedule
- Flexibility vs. Reliability
- ……



BUILDING A SOFTWARE VS. CODING

- When you say, “I can easily write xx lines of code a day”, you are talking about **coding something, not building a product**.

- **Building a software** is coding or programming integrated over
  - **Time**
  - **Scale**
  - **Trade-offs**

Q3. WHAT IS SOFTWARE ENGINEERING?

SOFTWARE CRISIS

The crisis manifested itself in several ways:

- Projects running over-budget
- Projects running over-time
- Software was very inefficient
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code difficult to maintain
- Software was never delivered

https://en.wikipedia.org/wiki/Software_crisis

THE ORIGIN OF SOFTWARE ENGINEERING

Software Engineering was first formally used by Professor Friedrich L. Bauer, at NATO conference, the first conference on software engineering, in 1968:

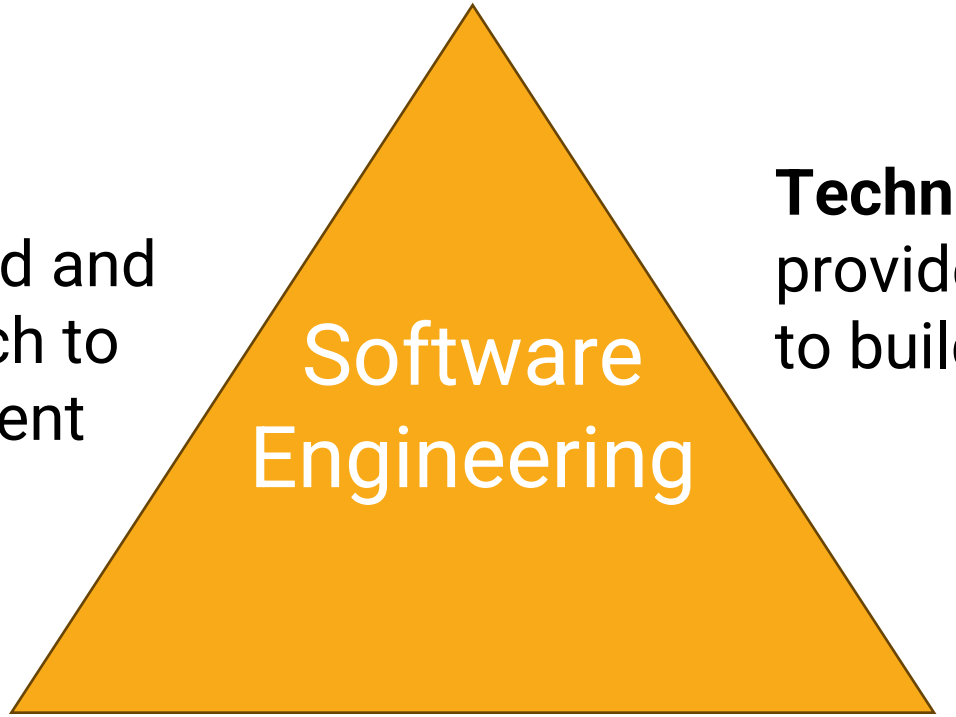
"The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines."

Software engineering is now considered one of major computing disciplines.

NATO conference, 1968

WHAT IS SOFTWARE ENGINEERING?

Engineering:
ensures a structured and systematic approach to software development



Technology:
provides the tools and platforms to build and implement solutions

Management: organizes teams and processes for efficient execution

SOFTWARE ENGINEERING: DEFINITION

Software engineering: the branch of engineering that deals with the design, development, testing, and maintenance of software applications, while ensuring that the software to be built is:

- Correct
- Consistent
- On budget
- On time
- Align with the requirements.

Q4. OBJECTIVE OF SOFTWARE ENGINEERING?

SOFTWARE BUGS ARE EXPENSIVE

- Ariane 5 Rocket Explosion (1996) – A software bug in the rocket's guidance system caused an integer overflow, leading to a catastrophic failure just 37 seconds after launch, resulting in a loss of $370 million.

- Knight Capital Trading Glitch (2012) – A faulty trading algorithm caused Knight Capital to lose $440 million in just 45 minutes due to unintended stock trades.

SOFTWARE BUGS ARE DEADLY

Toyota Case: Single Bit Flip That Killed

By Junko Yoshida 10.25.2013 0

Share Post Share on Facebook Share on Twitter in

MADISON, Wis. — Could bad code kill a person? It could, and it apparently did.

The Bookout v Toyota Motor Corp. case, which blamed sudden acceleration in a Toyota Camry for wrongful death, touches the issue directly.

This case — one of several hundred contending that Toyota's vehicles inadvertently accelerated — was the first in which a jury heard the plaintiffs' attorneys supporting their argument with extensive testimony from embedded systems experts. That testimony focused on Toyota's electronic throttle control system — specifically, its source code.

The plaintiffs' attorneys closed their argument by saying that the electronic throttle control system caused the sudden acceleration of a 2005 Camry in a September 2007 accident that killed one woman and seriously injured another on an Oklahoma highway off-ramp. It wasn't loose floor mats, a sticky pedal, or driver error.

Killer Bug. Therac-25: Quick-and-Dirty

Program code started using machines to kill people as early as in 1985.

The murderer

The Therac-25 is a radiation therapy machine, a medical linear accelerator produced by Atomic Energy of Canada Limited (AECL).

IMPROVE THE EFFICIENCY

DevOps, a software engineering practice, **improves efficiency dramatically**:

- 8000x faster deployment time
- 21% less time spent on unplanned work and rework
- 44% more time on new work
- 50% lower change-failure rates
- 50% less time spent fixing security issues
- 50% higher market cap growth over 3 years

https://services.google.com/fh/files/misc/state-of-devops-2014.pdf

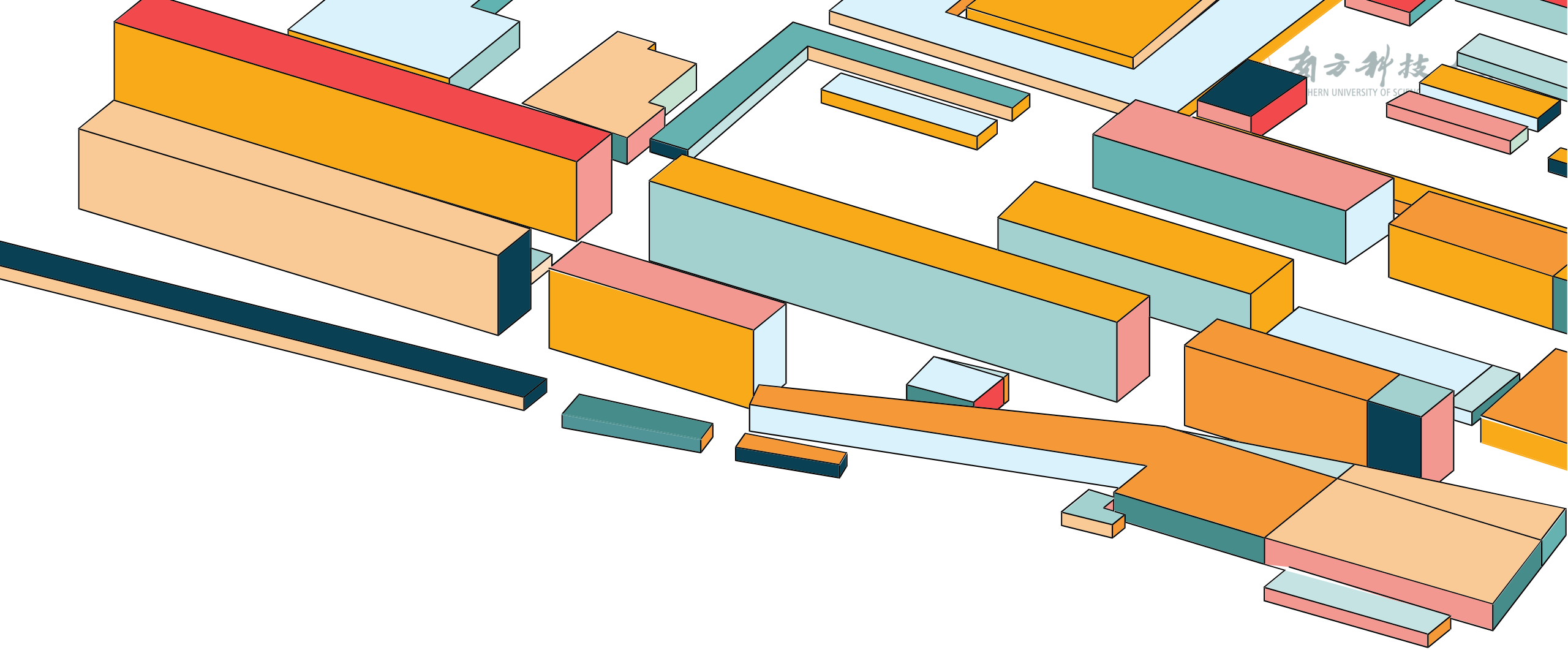
Q5. WHAT WILL WE LEARN IN THE COURSE?

COURSE THEMES

COURSE THEMES

The themes may change as the course progresses.

1. Intro to Software Engineering
2. Software Process & DevOps
3. Version Control Systems
4. Software Requirements
5. Software Design
6. Software Architecture
7. Build & Dependency Management
8. Software Documentation
9. Software Testing
10. Continuous Integration
11. Continuous Deployment
12. Software Quality & Metrics
13. Software Evolution & Maintenance
14. AI in Software Engineering



COURSE LOGISTICS

SCHEDULE

- Lectures and labs are paired: theory + implementation
- Labs reinforce and apply concepts from the lectures
- Schedule may be affected due to public holidays

Lecture

01班: Tuesday 3-4 智华楼207
02班: Monday 3-4 一教527

Lab

01班
1组 Tuesday 5-6 智华楼508 赵耀
2组 Wednesday 3-4 智华楼508 赵耀

02班
1组 Monday 5-6 智华楼509 陶伊达
2组 Tuesday 3-4 智华楼510 赵耀

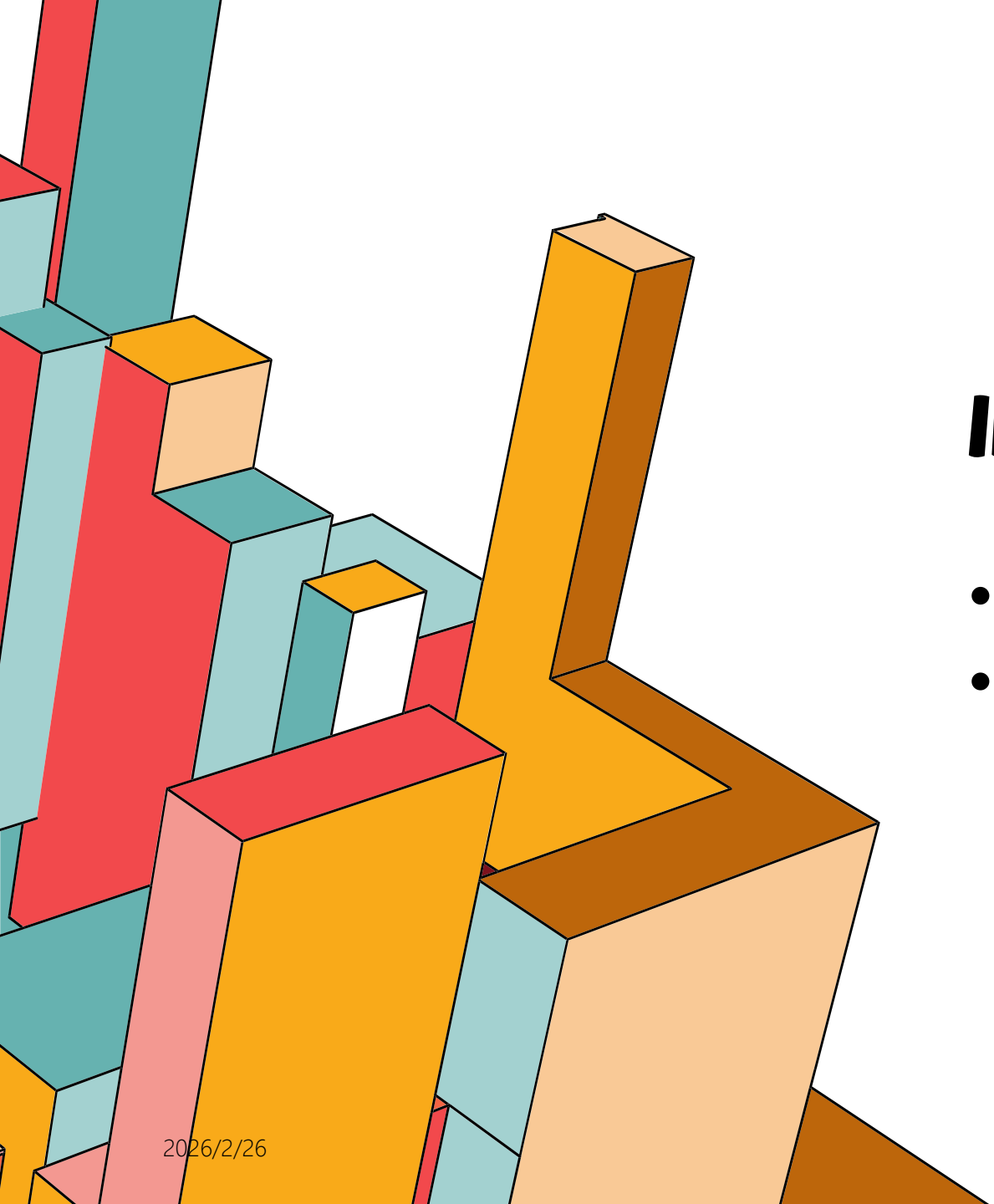
COURSE WEBSITE

Blackboard

You'll be automatically added to the enrolled class.
All course resources (slides, notifications, etc.) will be uploaded here.

GitHub Classroom:

Team projects, milestones, and assignments will be submitted here.
Details later.



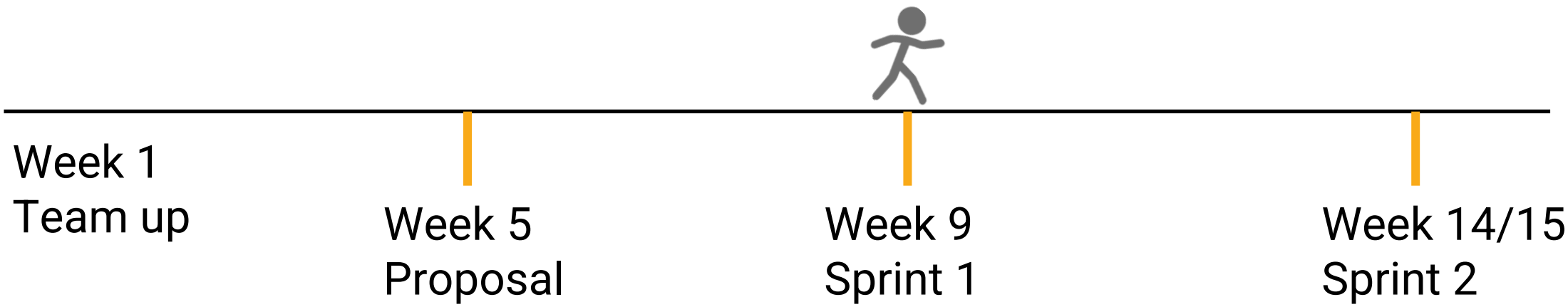
南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

INDIVIDUAL COURSEWORK

- Weekly lab practices (x10)
- Assignments (x2)

TEAM PROJECT

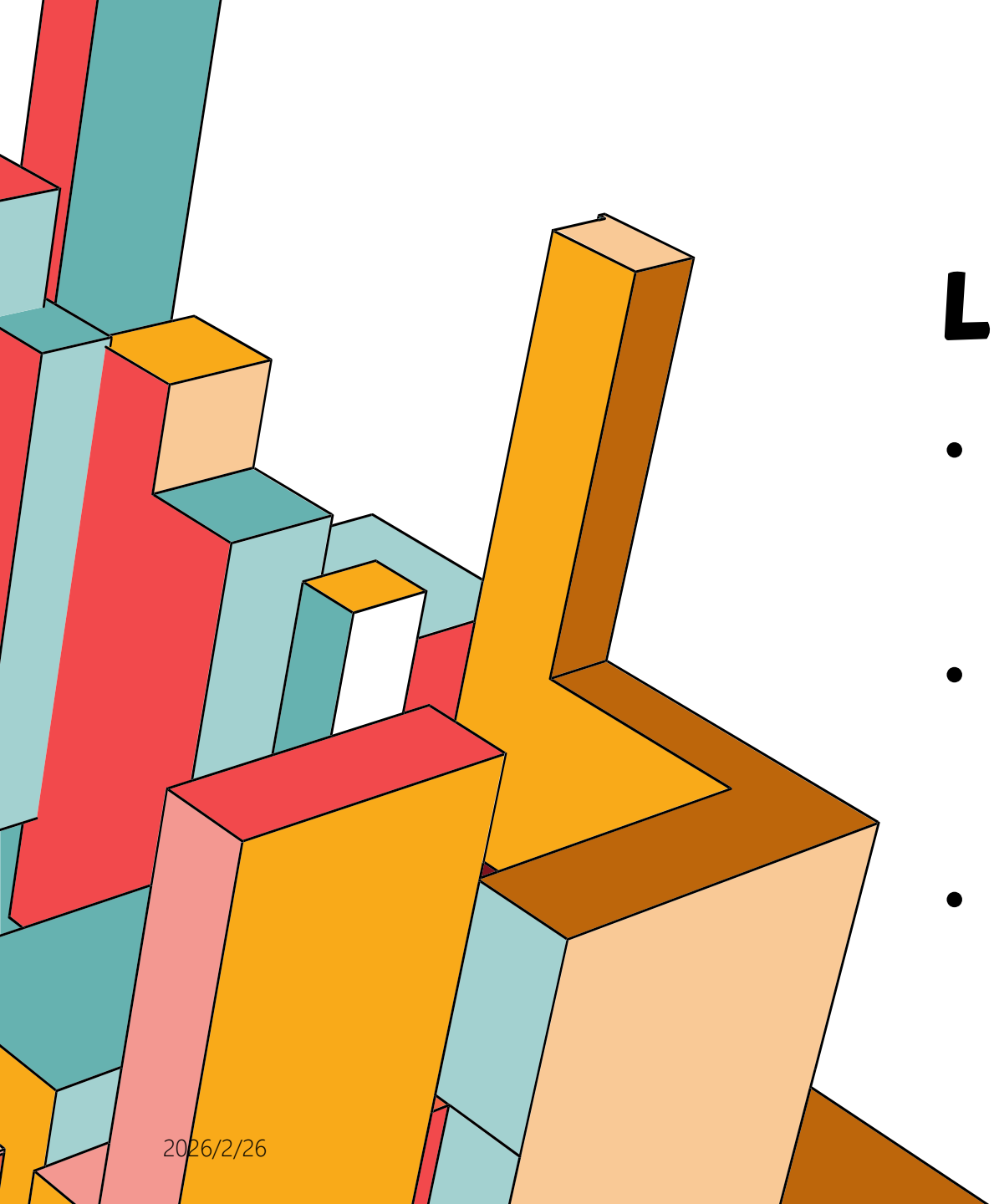
- Medium-sized projects that mirror realistic settings
- Work in a fixed team of 4-5 people throughout the semester
- 3 milestones, each with reports, deliverables, and presentations



More project details on Lab 1

GRADING POLICY

Lectures (attendance + quiz)	10%
Individual Assignments	10%
Lab (attendance + practices)	15%
Team Project - Proposal: 5% - Sprint 1: 10% - Sprint 2: 20%	35%
Final Exam (close-book)	30%



LATE DAY POLICY

- Individual assignments
  - No late day
- Lab practices
  - 20% penalty after the deadline
- Project milestones
  - No late day

TEXTBOOK

- Available in university libraries
- Optional purchase on major platforms

MOOC

学堂在线

Software Engineering

南方科技大学 0 人已报名

由 南方科技大学 开设，授课教师为 陶伊达 老师

开课班级 Round 1 2026-02-12~2026-07-23

陶伊达

计算机科学与工程系，讲师

陶伊达，博士，南方科技大学计算机系讲师。
在南科大开设《计算机程序设计基础》、《...

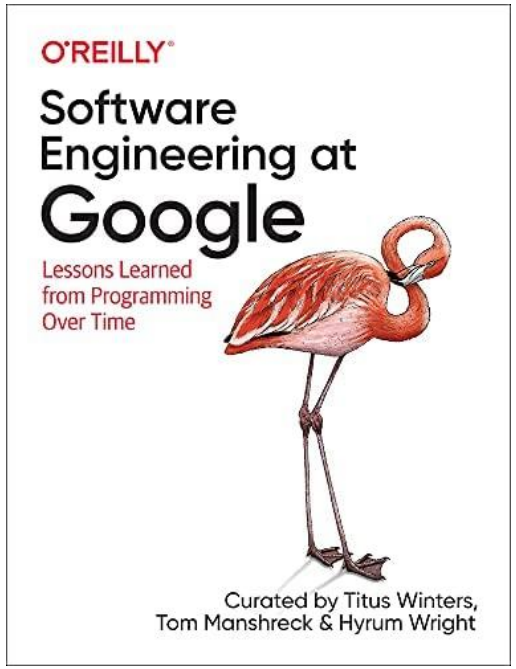
1. Software Enigeering Overview
2. Software Process
3. Software Requirements
4. Version Control
5. Software Architecture
6. Software Build Systems
7. Code Quality
8. Software Testing
9. CI/CD
10. Cloud-native
11. Software Maintenance and Evolution
12. Exam

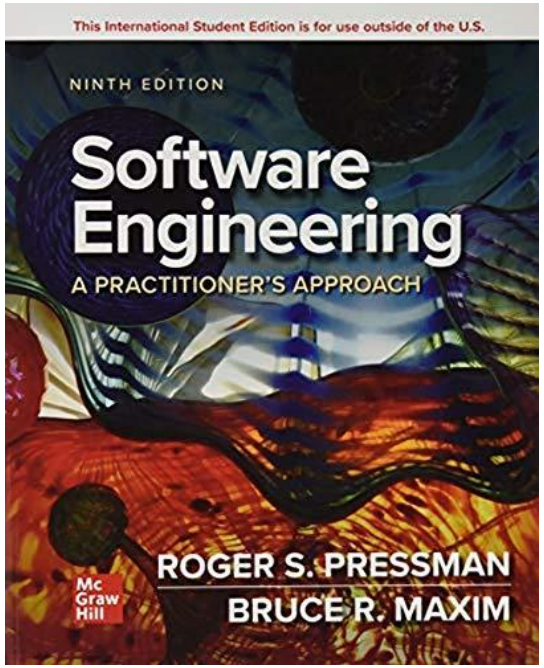
- Consistent with our syllabus
- 5-10 min video
- English and Chinese subtitles
- Quiz and exam-style questions
  (**as part of the assignments, details later**)

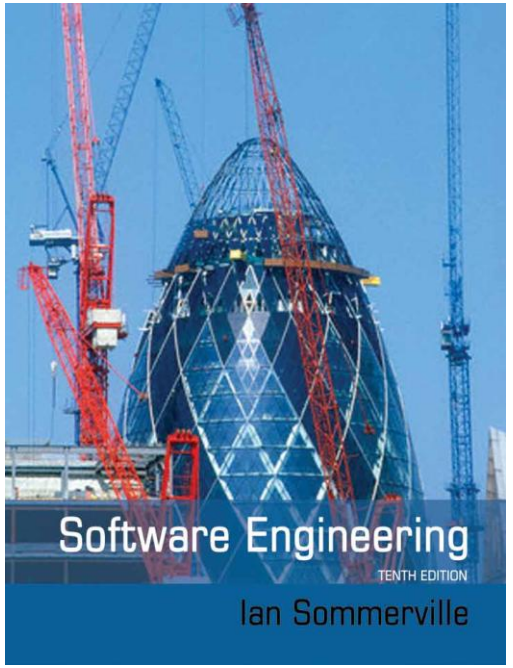
免费加入学习

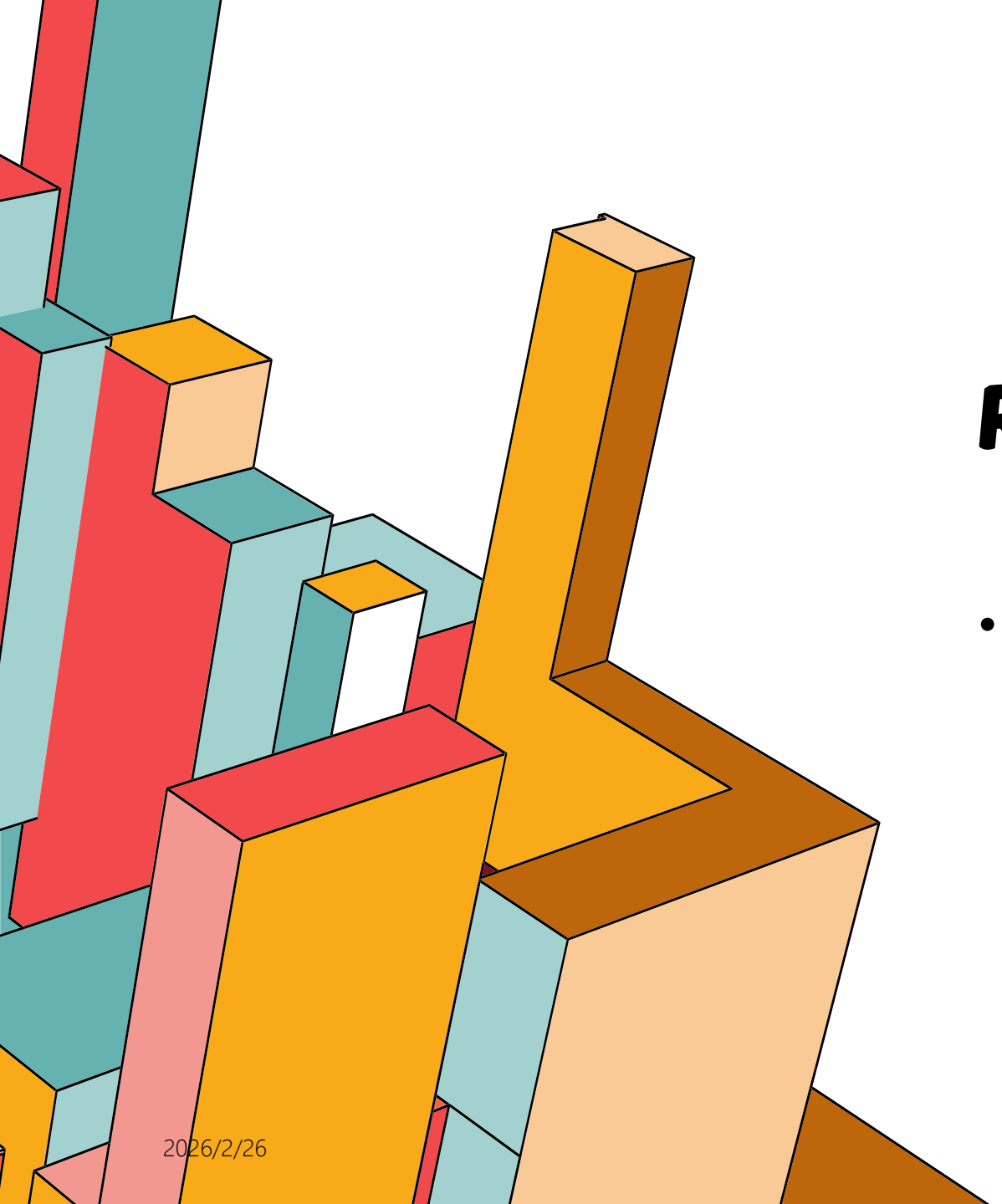
https://www.xuetangx.com/course/sustc08091016972intl/29822401?channel=i.area.manual_search
Or just search for “Software engineering”

REFERENCE TEXTBOOKS









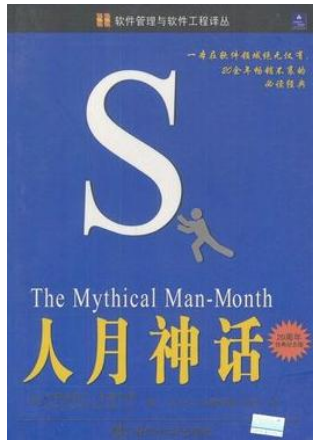
READINGS

- Chapter 4. What is Software Engineering? Software Engineering at Google by Titus Winters, et al.

DISCUSSION

No Silver Bullet (1986): There is no single development method or technology that can dramatically improve software productivity by an order of magnitude within a decade, because software challenges are rooted in inherent complexity, not just easily solvable technical or process flaws.

没有银弹： 没有任何一种单一的技术或方法，能够在十年内让软件的生产效率获得数量级的提升。因为软件开发的核心挑战源于其固有的复杂性，而不是那些靠某种技术或流程就能轻易解决的表面问题。



Do you think that AI is the silver bullet?
Fill in the survey on BB!

NEXT

- Software process
- Agile
- DevOps