



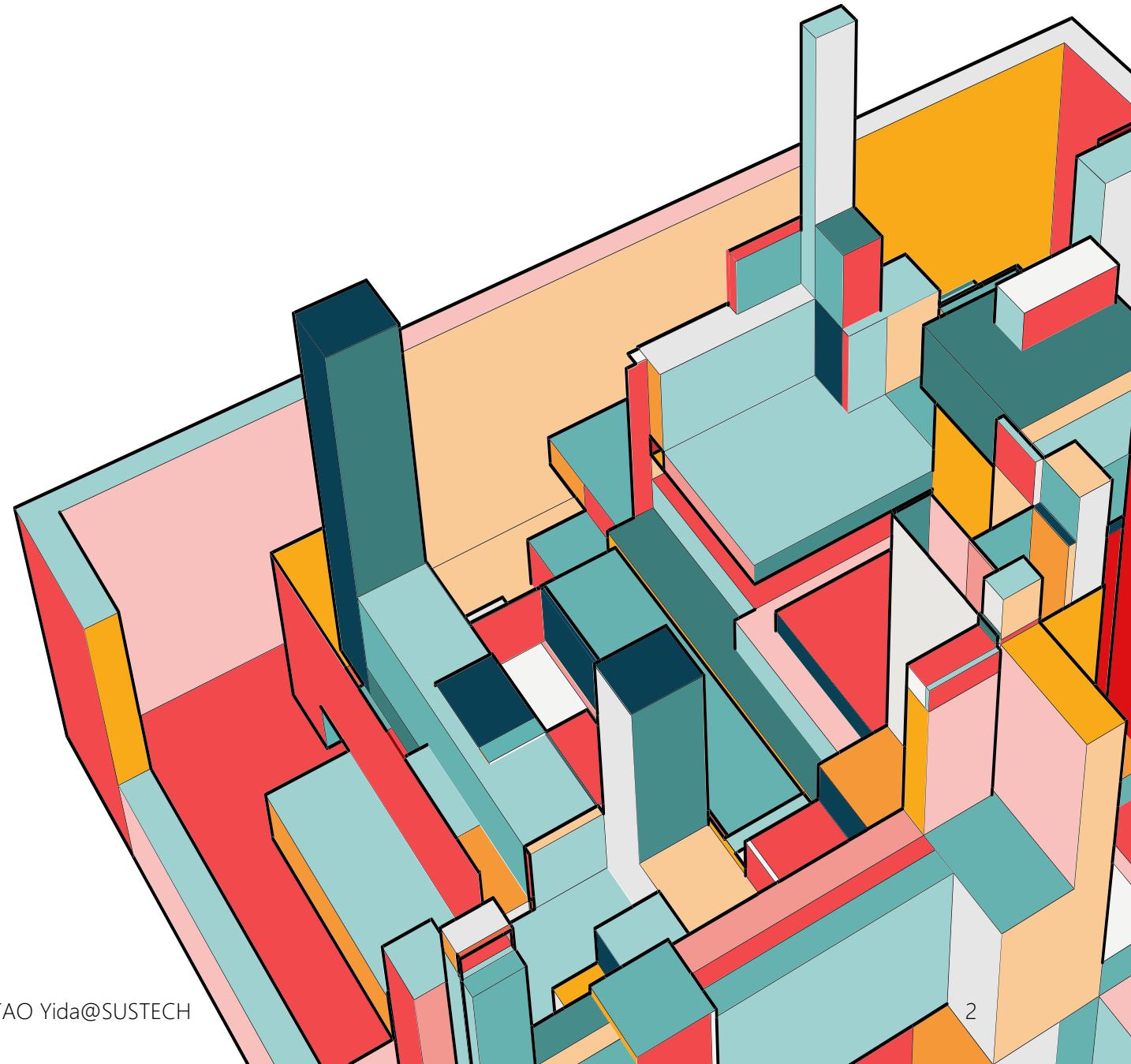
CS304 SOFTWARE ENGINEERING

Yida Tao

taoyd@sustech.edu.cn

LECTURE 1

- SE Overview
- Time & Change
- Scale & Efficiency
- Trade-offs & Costs



SOFTWARE IS EVERYWHERE

<i>Data processing:</i>	telephone billing, pensions
<i>Real time:</i>	air traffic control
<i>Mobile devices:</i>	digital camera, GPS, iPhone
<i>Information systems:</i>	web sites, digital libraries
<i>Sensors:</i>	weather data
<i>System software:</i>	operating systems, compilers
<i>Communications:</i>	routers, telephone switches
<i>Offices:</i>	word processing, video conferences
<i>Scientific:</i>	simulations, weather forecasting
<i>Graphical:</i>	film making, design

<https://www.cs.cornell.edu/courses/cs5150/2010fa/syllabus.html>

SOFTWARE IS EATING THE WORLD

In the future, every company will become a software company.

Today, every company is software intensive. That means, if they do not have a software strategy, then they will not be able to compete in the world and eventually kind of fade out.



Marc Andreessen. Co-author of Mosaic, the first widely used web browser; co-founder of Netscape.
Why Software Is Eating the World. Wall Street Journal, 2011.

SOFTWARE IS EATING THE WORLD

Denver	813	Southwest			
Denver	995	Southwest	7:50	18B	Canceled
El Paso	2411	Southwest	2:30	12B	Canceled
Honolulu	3845	Southwest	7:45	17B	Now 7:56 PM
Houston Hobby	2403	Southwest	1:10	18B	On Time
Houston Hobby	2457	Southwest	5:20	17B	Canceled
Kahului Maui	8829	Southwest	2:10	17B	Now 3:59 PM
Las Vegas	2403	Southwest	1:10	18B	On Time
Las Vegas	4558	Southwest	4:20	16	Canceled
Las Vegas	753	Southwest	6:40	16	Canceled
Little Rock	1278	Southwest	5:25	17A	Canceled
Miami	1482	Southwest	11:40	12B	Canceled
Nashville	1527	Southwest	11:20	12B	Canceled
Nashville	1447	Southwest	4:10	18A	Now 6:13 PM
Nashville	1278	Southwest	5:25	17A	Canceled
New Orleans	1972	Southwest	9:15	15	Now 11:04 AM
New Orleans					Now 2:33 PM

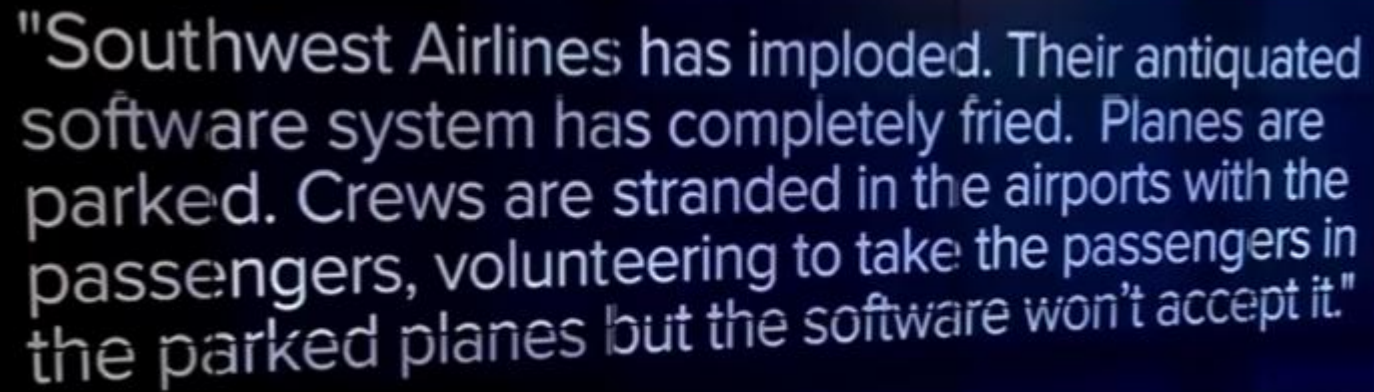


Southwest Meltdown Shows Airlines Need Tighter Software Integration

The airline industry is long overdue for a tech overhaul that takes full advantage of the cloud and data integration, analysts say

SOFTWARE IS EATING THE WORLD

Southwest relies on crew-assignment software called SkySolver, an old application developed decades ago



"Southwest Airlines has imploded. Their antiquated software system has completely fried. Planes are parked. Crews are stranded in the airports with the passengers, volunteering to take the passengers in the parked planes but the software won't accept it."

SOFTWARE IS EATING THE WORLD

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.

Hard questions raised when a software 'glitch' takes down an airliner

Posted by [Taylor Armerding](#) on Thursday, November 29, 2018

The parts and systems on an airplane don't have to fail in a big way to have big consequences. A flaw in airline software could be a matter of life or death.



The original version of this post was published in Forbes.

It doesn't take a failure of anything big to cause big trouble—big as in massive, catastrophic, and lethal damage to a sophisticated transportation system.

SOFTWARE CRISIS

- Software crisis is a term used in the early days of computing science for the difficulty of writing useful and efficient computer programs in the required time
- The causes of the software crisis were linked to the overall complexity of hardware and the software development process.

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

SOFTWARE CRISIS

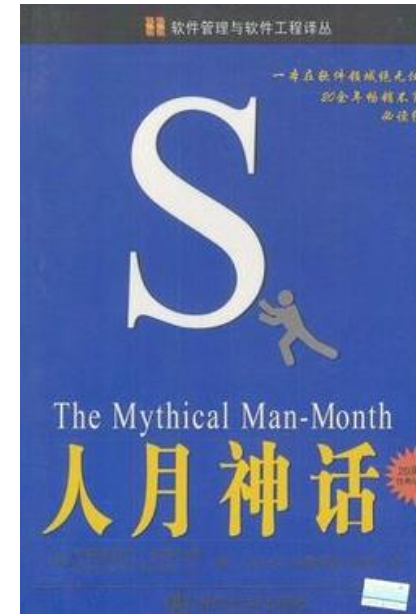
The crisis manifested itself in several ways:

- 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 DEVELOPMENT OF IBM OS/360

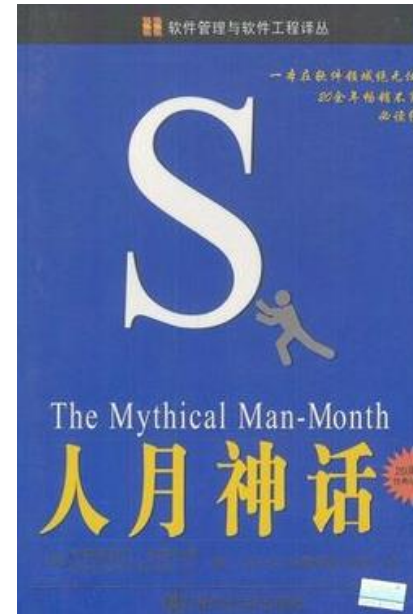
- Time: 1963-1966
- Human involved: 5000 man-month (one person's working time for a month)
- Codebase: 1M lines of code
- Cost: hundreds of millions \$



图灵奖得主、IBM 360系统之父
Frederick Brooks

THE DEVELOPMENT OF IBM OS/360

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

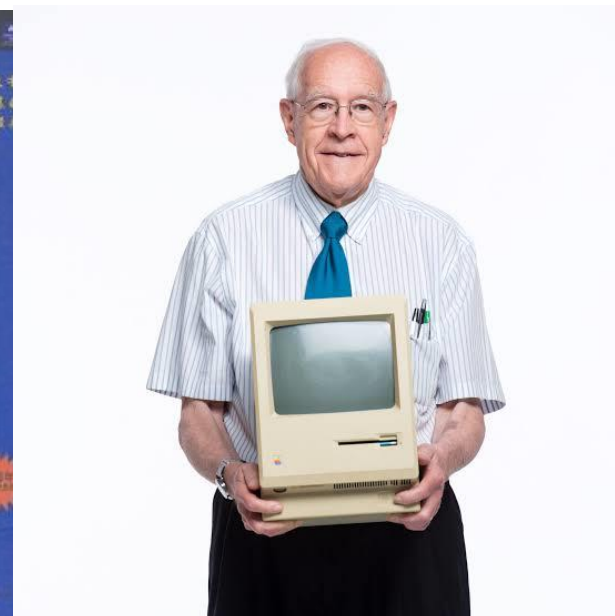
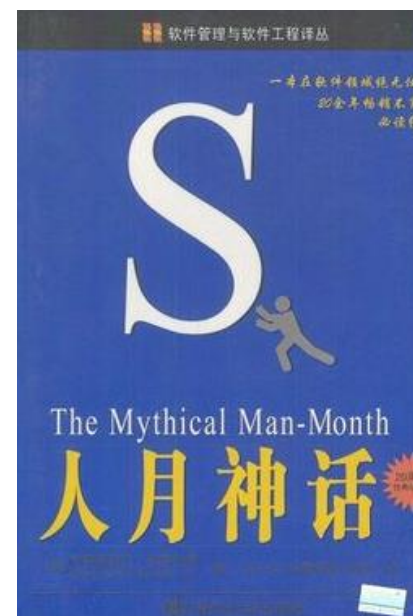


图灵奖得主、IBM 360系统之父
Frederick Brooks

THE DEVELOPMENT OF IBM OS/360

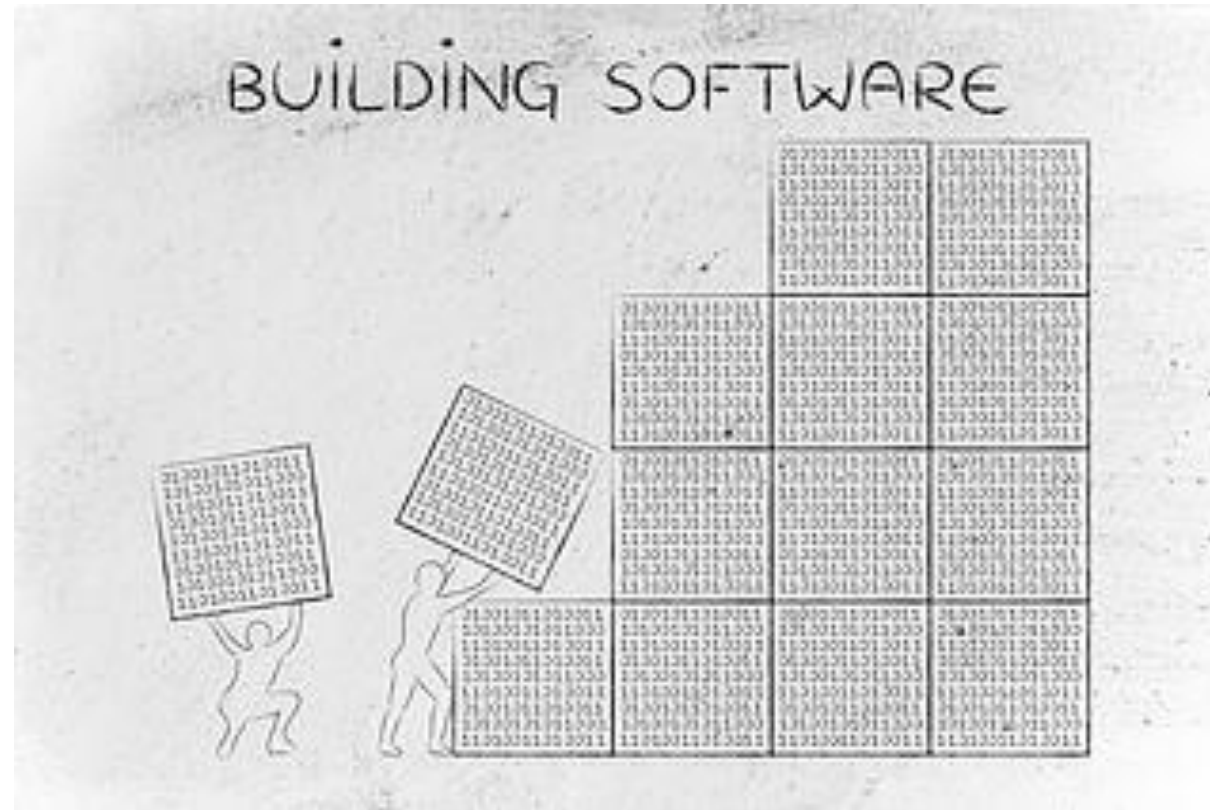
Software like a tar pit (焦油坑): The more you fight it, the deeper you sink!

.....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

BUILDING SOFTWARE IS COMPLEX



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?

- **Software:** a program or set of programs containing instructions that provide desired functionality.
- **Engineering:** the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems.

WHAT IS SOFTWARE ENGINEERING?

- **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
 - Within the required requirements.

SOFTWARE ENGINEERING VS. PROGRAMMING

- **Programming** is a significant part of software engineering
 - Programming is how you generate a new software in the first place
- **Software engineering** is programming integrated over **time, scale, and trade-offs**



WHAT'S THE EXPECTED LIFE SPAN OF YOUR CODE?

Programming

- **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 😊

Software Engineering

- **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

- **Human resources:** programming assignments are done by individuals or 2-3 sized small groups
- **Computing resources:** a single laptop is generally sufficient for programming assignments

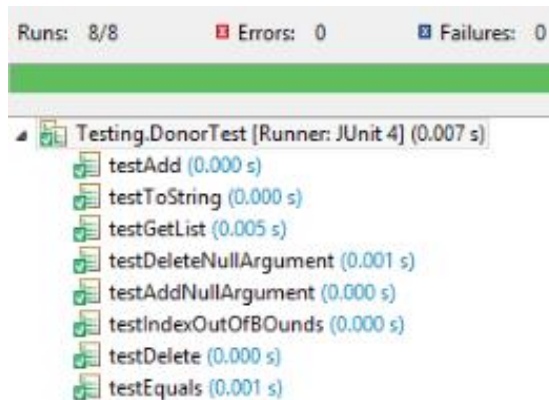
Software Engineering

- **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

COMPLEXITY OF DECISIONS

Programming

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

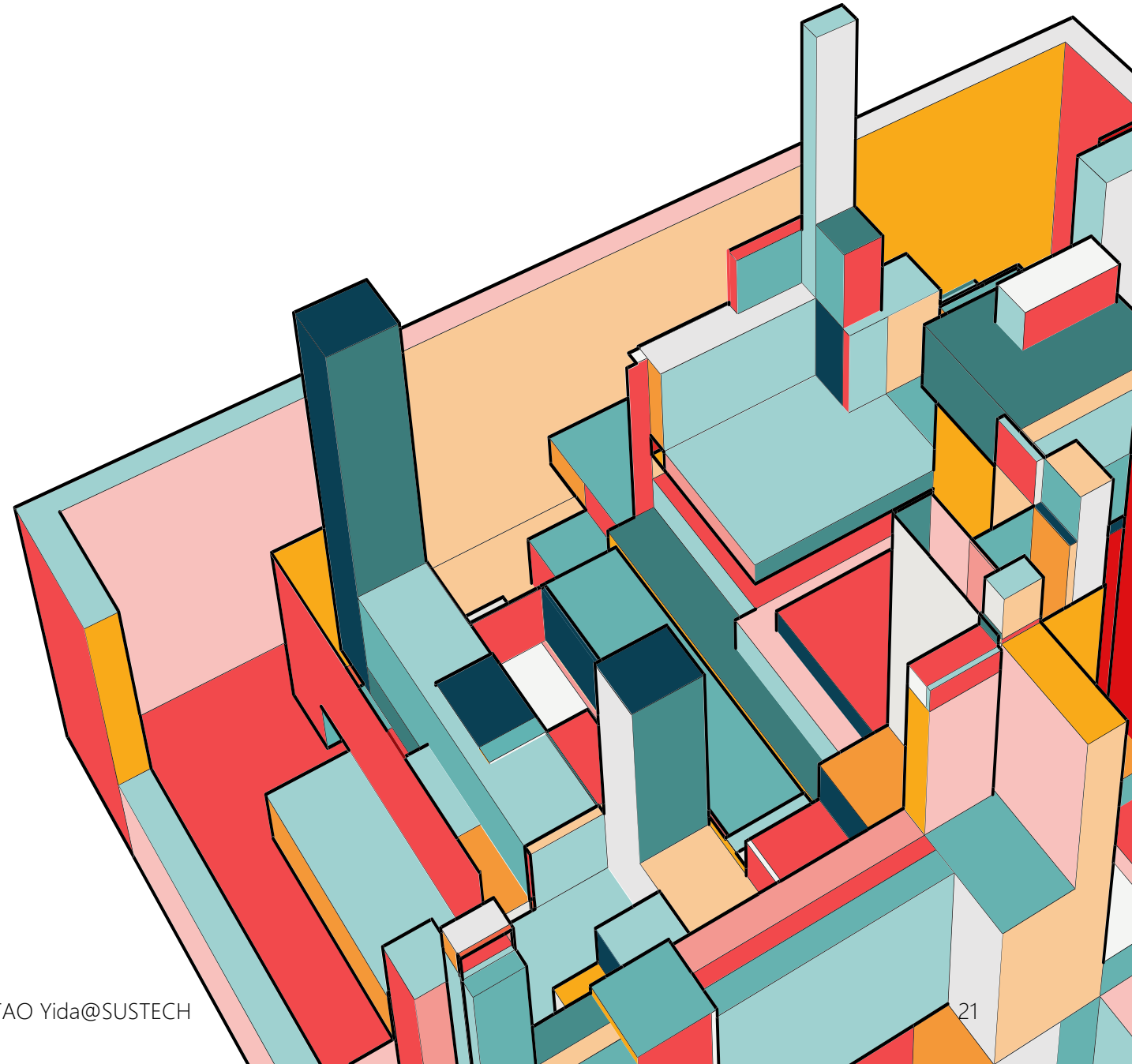


Software Engineering

- Software quality
- Engineering efforts
- Financial costs
- Resource costs (e.g., CPU time)
- Social impact
- ...

LECTURE 1

- SE Overview
- Time & Change
- Scale & Efficiency
- Trade-offs & Costs



LIFE SPAN OF SOFTWARE

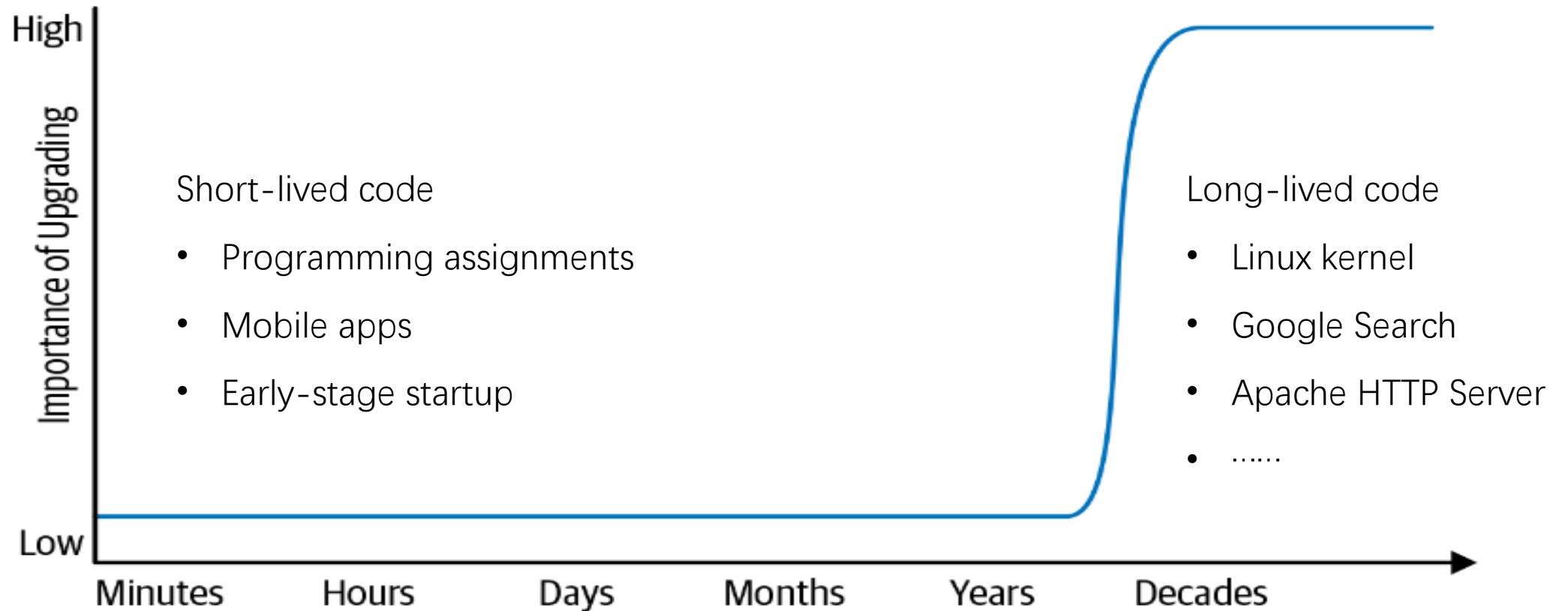


Figure 1-1. Life span and the importance of upgrades

LIFE SPAN OF SOFTWARE

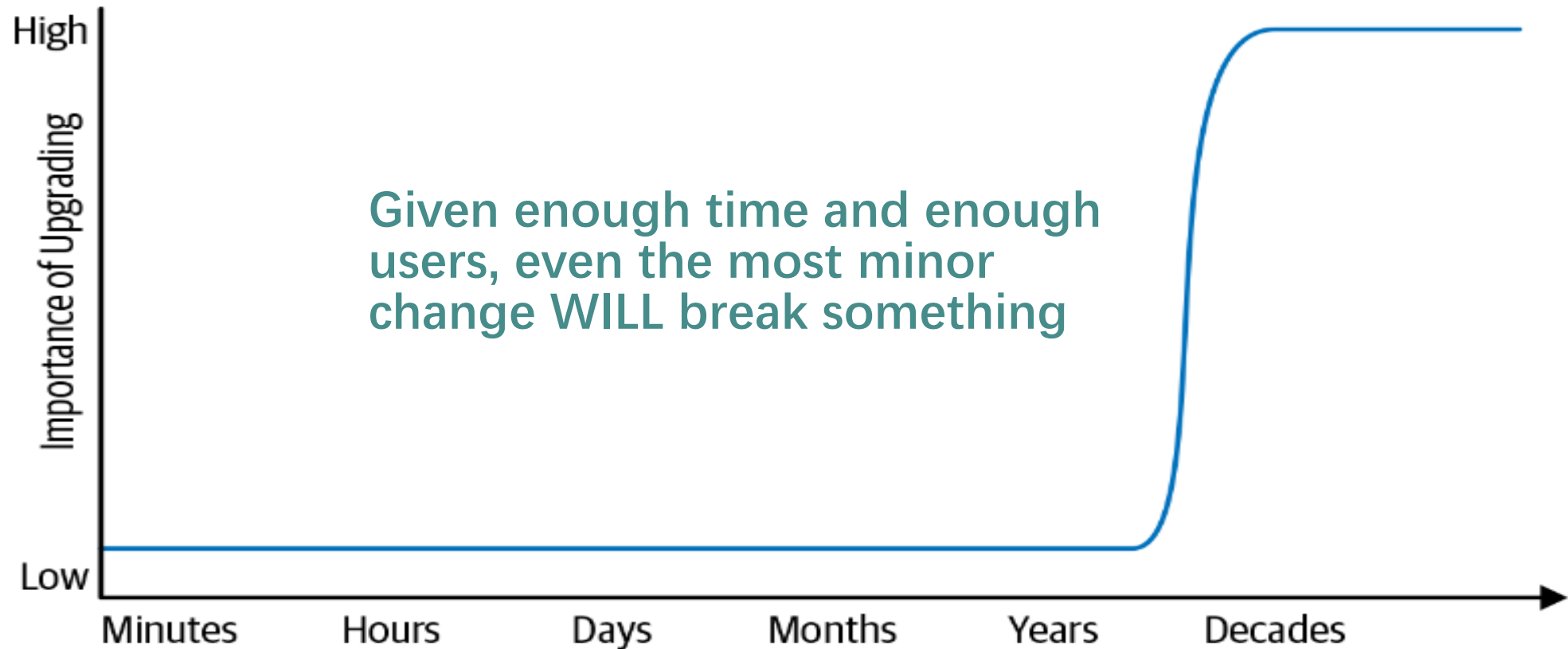


Figure 1-1. Life span and the importance of upgrades

HYRUM'S LAW

With a sufficient number of users of an API, it does not matter what you promise in the contract: all observable behaviors of your system will be depended on by somebody.

- Hyrum's Law is named after Google software engineer Hyrum Wright and states that even though you may design your API for extensibility and count on being able to evolve it, things can become more complicated.
- In particular, when you have many consumers, they may depend on things they shouldn't depend on.
- On the one hand, having more consumers is a good strategy to increase the value that an API generates.
- On the other hand, the more consumers there are, the more likely it is that (because of Hyrum's law) some of them are negatively affected by changes in the API that in theory shouldn't have affected anybody.

HYRUM'S LAW IN PRACTICE

```
>>> for i in {"apple", "banana", "carrot", "durian", "eggplant"}: print(i)
```

```
...
```

```
durian
```

```
carrot
```

```
apple
```

```
eggplant
```

```
banana
```

- User 1: assume that the hash iteration ordering is fixed and write some code that depends on the ordering
- User 2: assume that the hash iteration ordering is random, and use it as an inefficient random-number generator
- User 3: assume that the hash iteration ordering is random and never write any code that depend on the ordering.

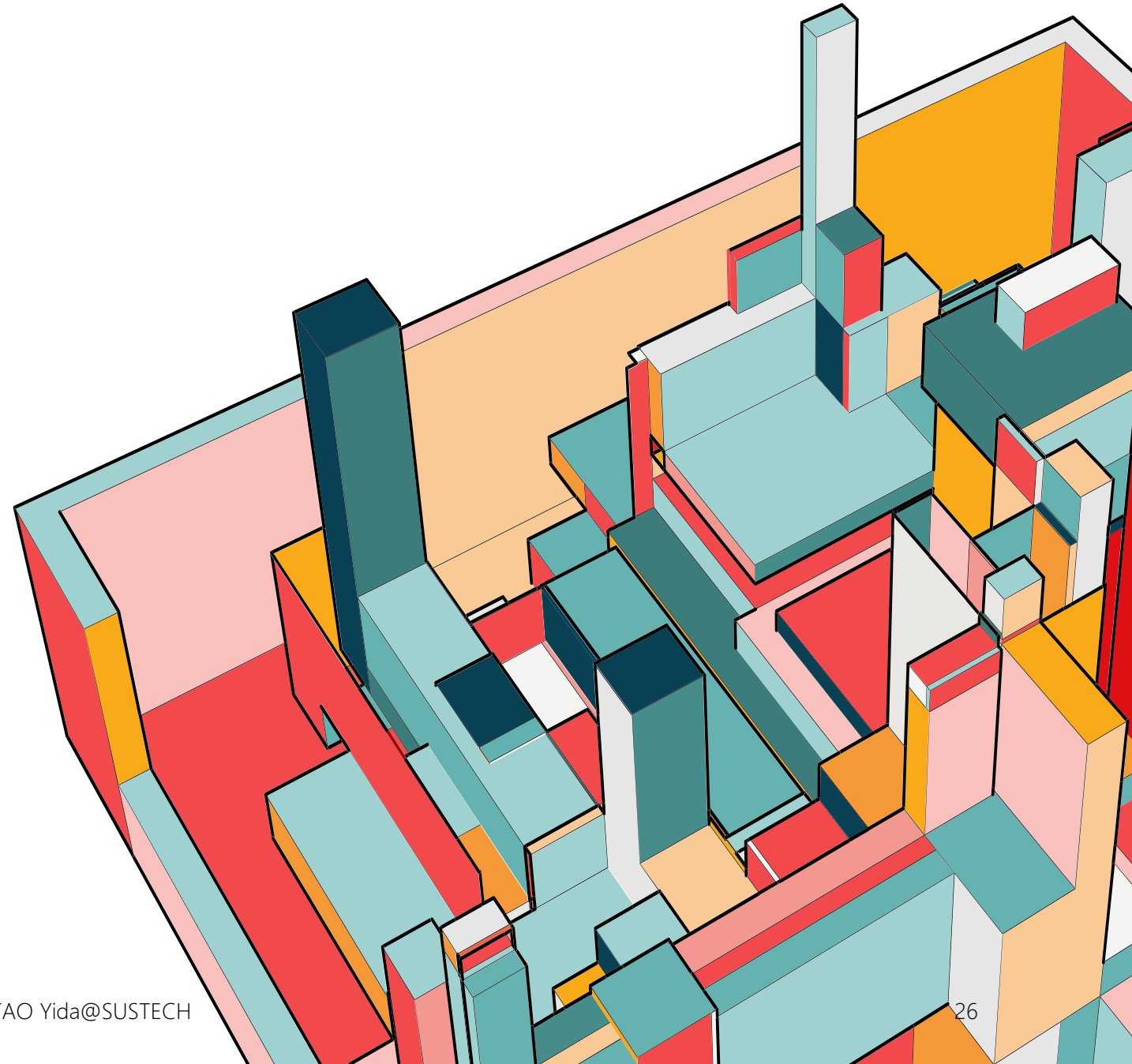
User 1 breaks if the hash ordering becomes randomized

User 2 breaks if the hash ordering becomes fixed

User 4 breaks if the results produced by user 3 are consumed by user 4, whose code depends on the order of the results

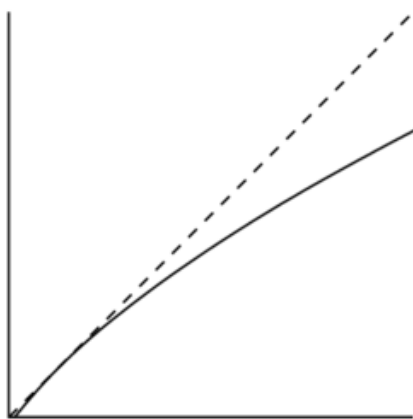
LECTURE 1

- SE Overview
- Time & Change
- Scale & Efficiency
- Trade-offs & Costs



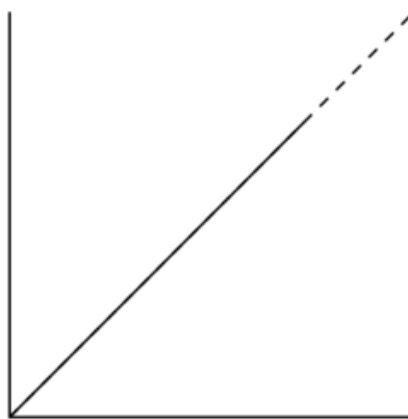
SCALABILITY

- Your organization's codebase is sustainable when you are able to change all of the things that you ought to change, safely, and can do so for the life of your codebase.
- If costs grow **superlinearly** over time, the operation clearly is **not scalable**.



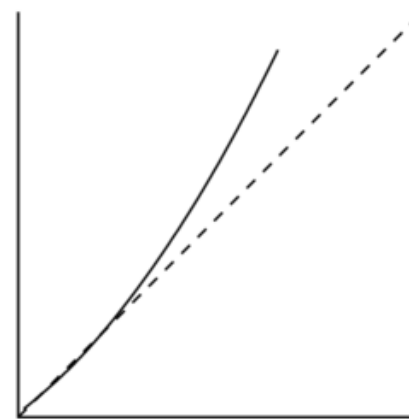
sublinear

(a)



linear

(b)

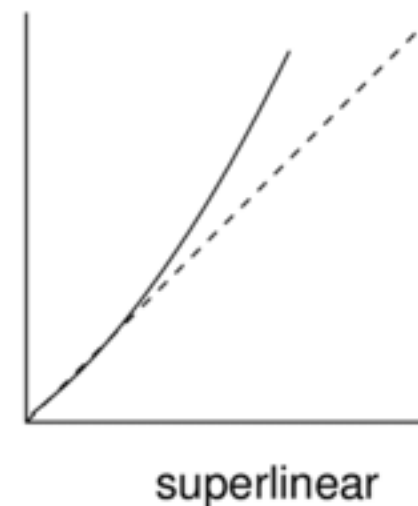
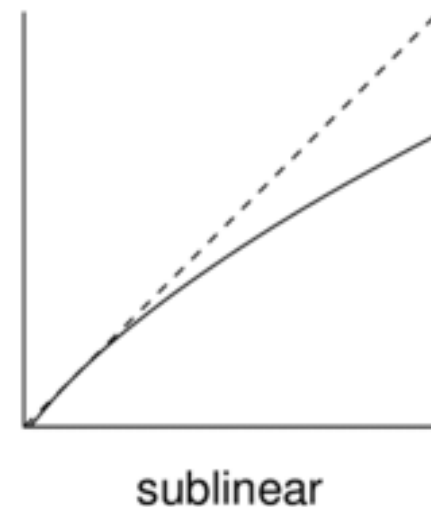


superlinear

(c)

SCALABILITY

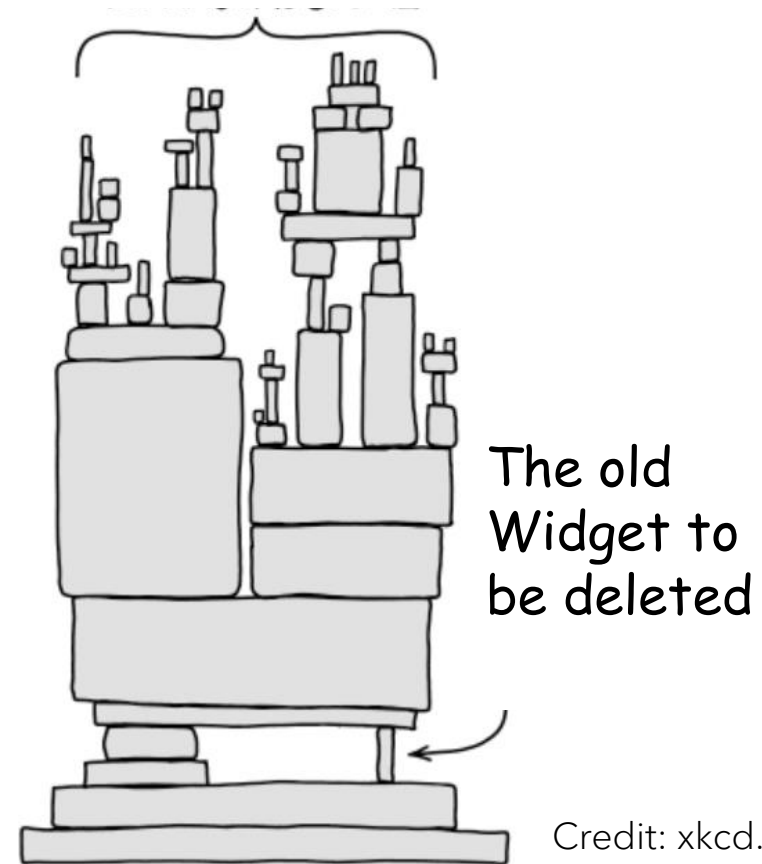
- X axis: the demand (e.g., codebase)
- Y axis: resources costs
 - How many additional computing resources (e.g., memory, storage) are needed?
 - How many additional human involvement is needed?
 - How long does it take to do a full build?
 - How long does it take to pull a fresh copy of the repository?
 - How much will it cost to upgrade to a new language version?
 -



CASE STUDY: DEPRECATION

- A new Widget has been developed.
- The decision is made that everyone should use the new one and stop using the old one.
- To motivate this, project leads say “We’ll delete the old Widget on August 15th; make sure you’ve converted to the new Widget.”

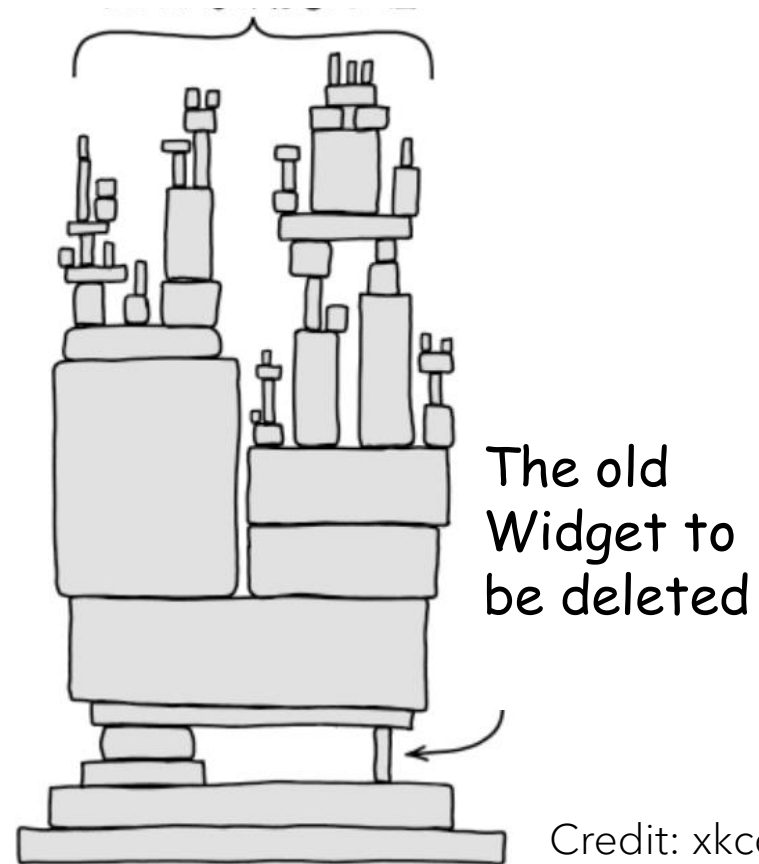
All the nice projects that are built



CASE STUDY: DEPRECATION

- This type of approach might work in a small software setting, but quickly fails as both the depth and breadth of the dependency graph increases.
- Teams depend on an ever-increasing number of Widgets, and a single build break can affect a growing percentage of the company.

All the nice projects that are built

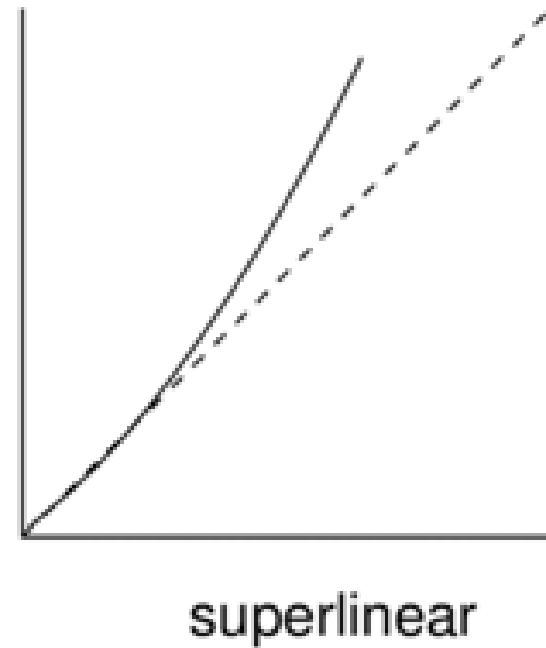


Credit: xkcd.com

CASE STUDY: DEPRECATION

- This type of approach might work in a small software setting, but quickly fails as both the depth and breadth of the dependency graph increases.
- Teams depend on an ever-increasing number of Widgets, and a single build break can affect a growing percentage of the company.

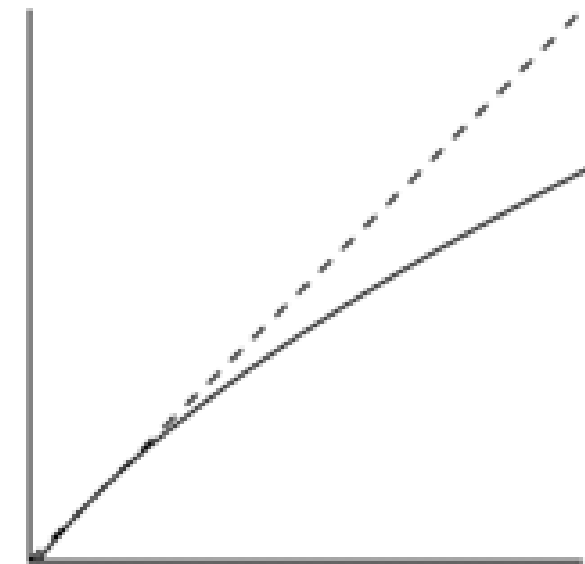
Dependent users/companies quickly grow:
not scalable



CASE STUDY: DEPRECIATION

- A dedicated group of experts in the infrastructure team
 - Either do the update in place, in backward-compatible fashion.
 - Or do the work to migrate their users to new versions
- Having a dedicated group of experts execute the change **scales better** than asking for more maintenance effort from every user

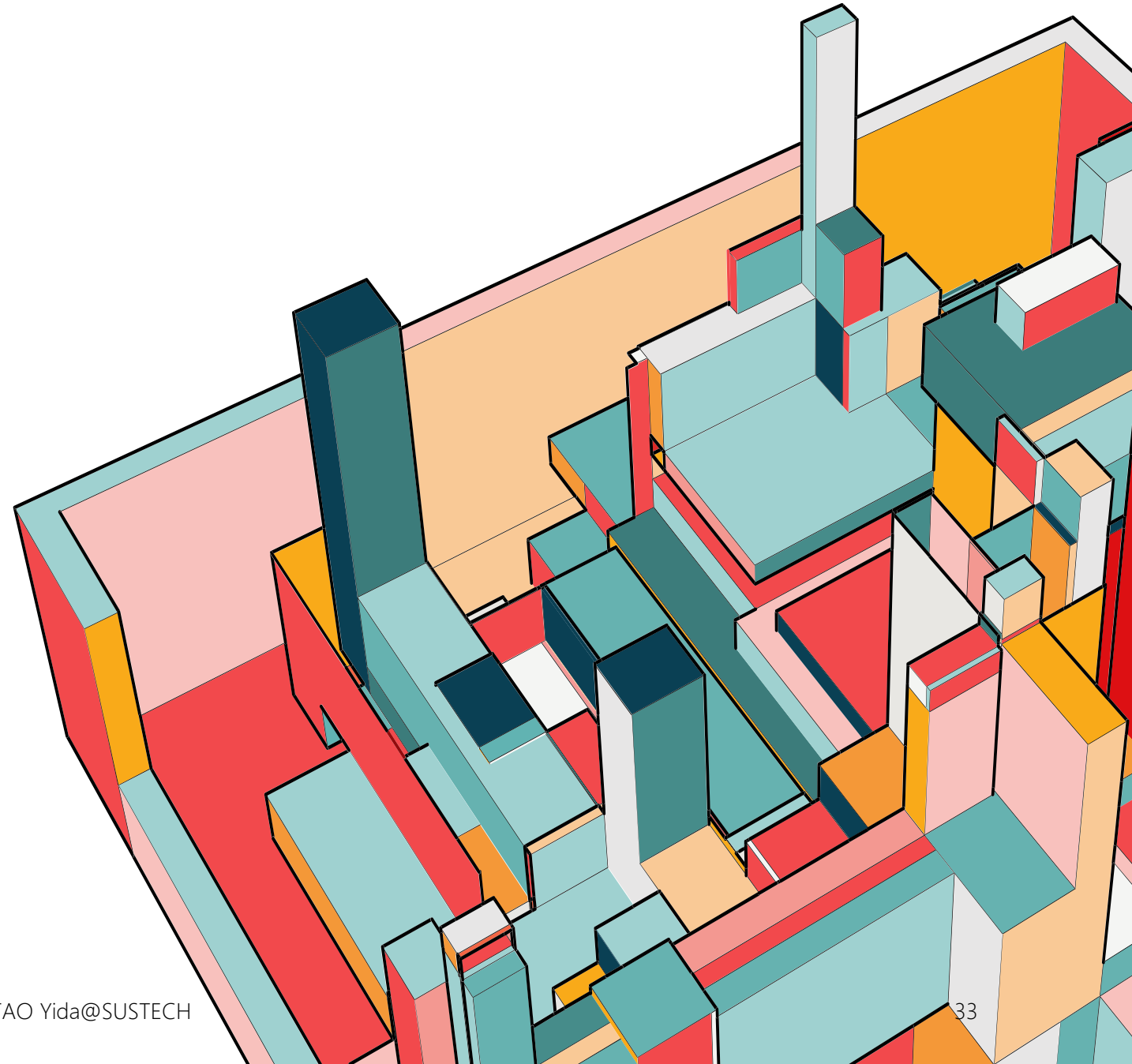
A small group of experts: scales better



sublinear

LECTURE 1

- SE Overview
- Time & Change
- Scale & Efficiency
- **Trade-offs & Costs**



MAKING GOOD DECISIONS

- Now that we've understand
 - how to program
 - the lifetime of the software we're maintaining
 - how to maintain the software as we scale up with more engineers and new features
- All that's left is to make good decisions
 - (Bad) "Because I said so", "Because everyone else does this way"
 - (Good) We should be able to explain our work when deciding between **the general costs** for two engineering options.



COST

- Financial costs (e.g., money)
- Resource costs (e.g., CPU time)
- Personnel costs (e.g., engineering effort)
- Transaction costs (e.g., what does it cost to take action?)
- Opportunity costs (e.g., what does it cost to not take action?)
- Societal costs (e.g., what impact will this choice have on society at large?)

Which can be quantified, which cannot?

MEASURABLE QUANTITIES

- Some quantities are measurable or can at least be estimated
 - CPU costs, RAM costs, Engineering hours, etc.
- Making decisions over measurable quantities are relatively straightforward
 - “If I spend two weeks changing this linked-list into a higher-performance structure, I’m going to use five gibibytes more production RAM but save two thousand CPUs. Should I do it?”
 - Not only does this question depend upon the relative cost of RAM and CPUs, but also on personnel costs (two weeks of support for a software engineer) and opportunity costs (what else could that engineer produce in two weeks?).

SUBTLE QUANTITIES

- Some of the quantities are subtle and hard to quantify
 - We don't know how much engineer-time this will take
 - The productivity of developers
 - How do you measure the engineering cost of a poorly designed API?
 - How do you quantify the societal impact of a product choice?
- For this type of decision, there is no easy answer.

A GENERAL RULE OF REDUCING COST - SHIFT LEFT

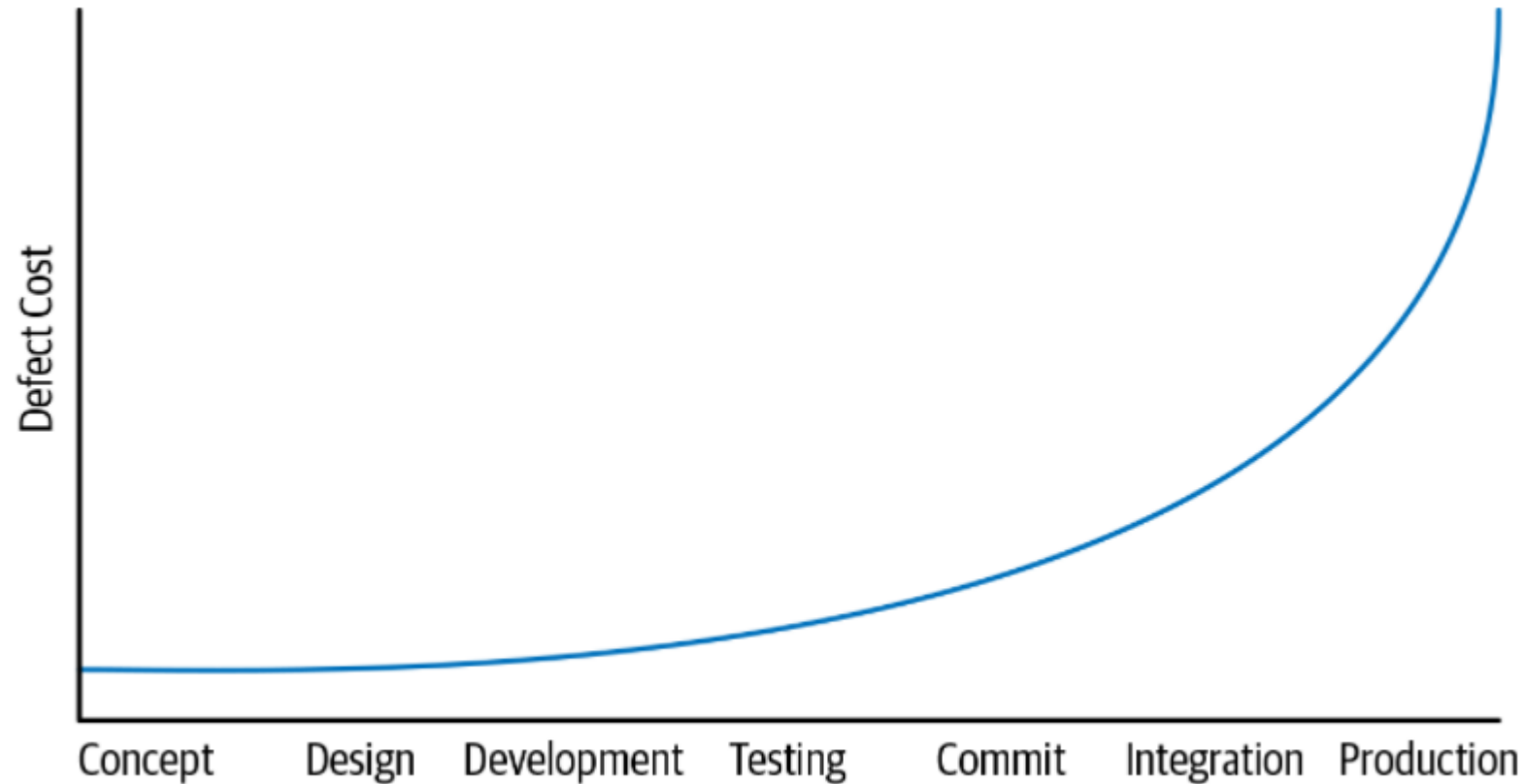


Figure 1-2. Timeline of the developer workflow

A GENERAL RULE OF REDUCING COST - SHIFT LEFT

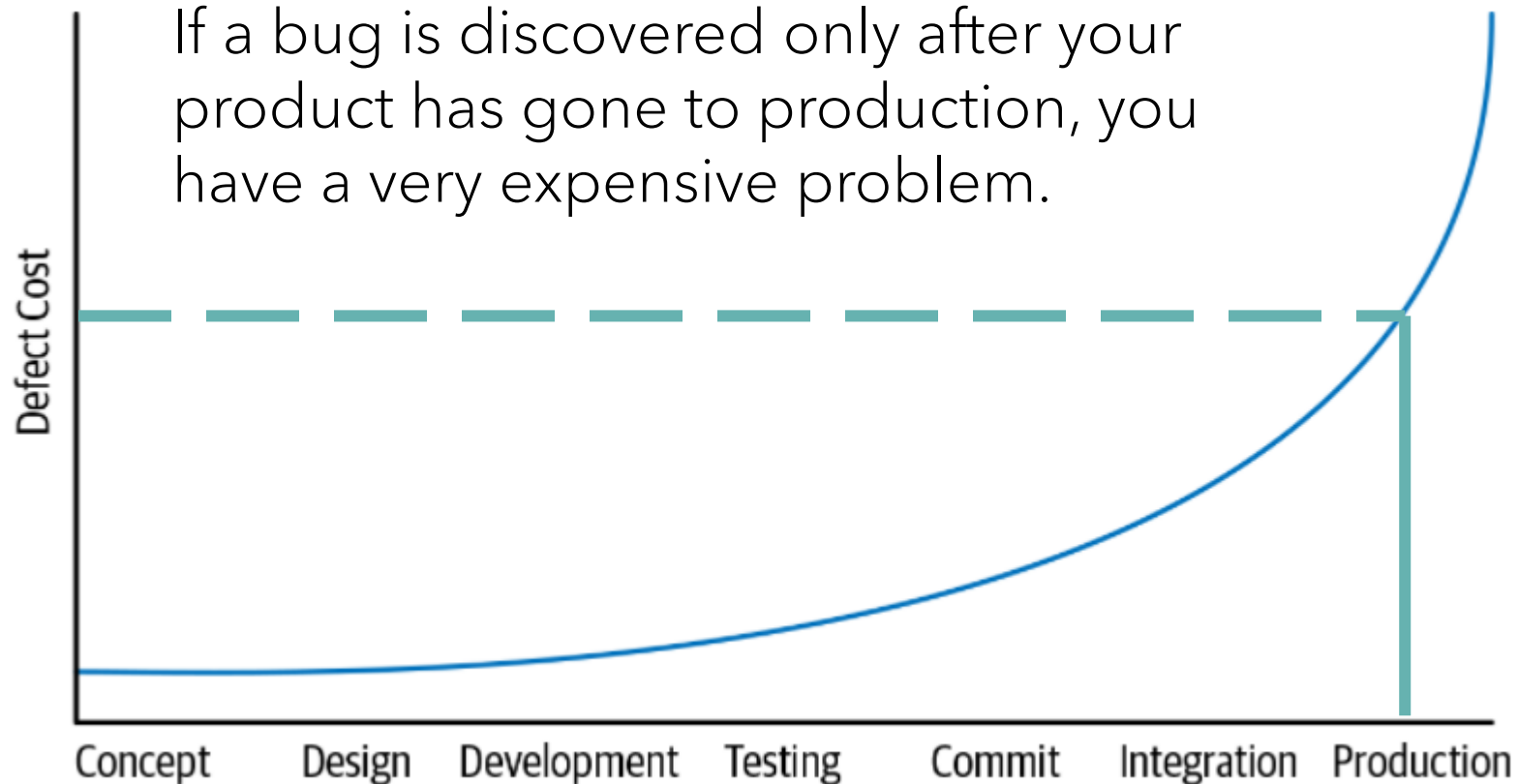


Figure 1-2. Timeline of the developer workflow

A GENERAL RULE OF REDUCING COST - SHIFT LEFT

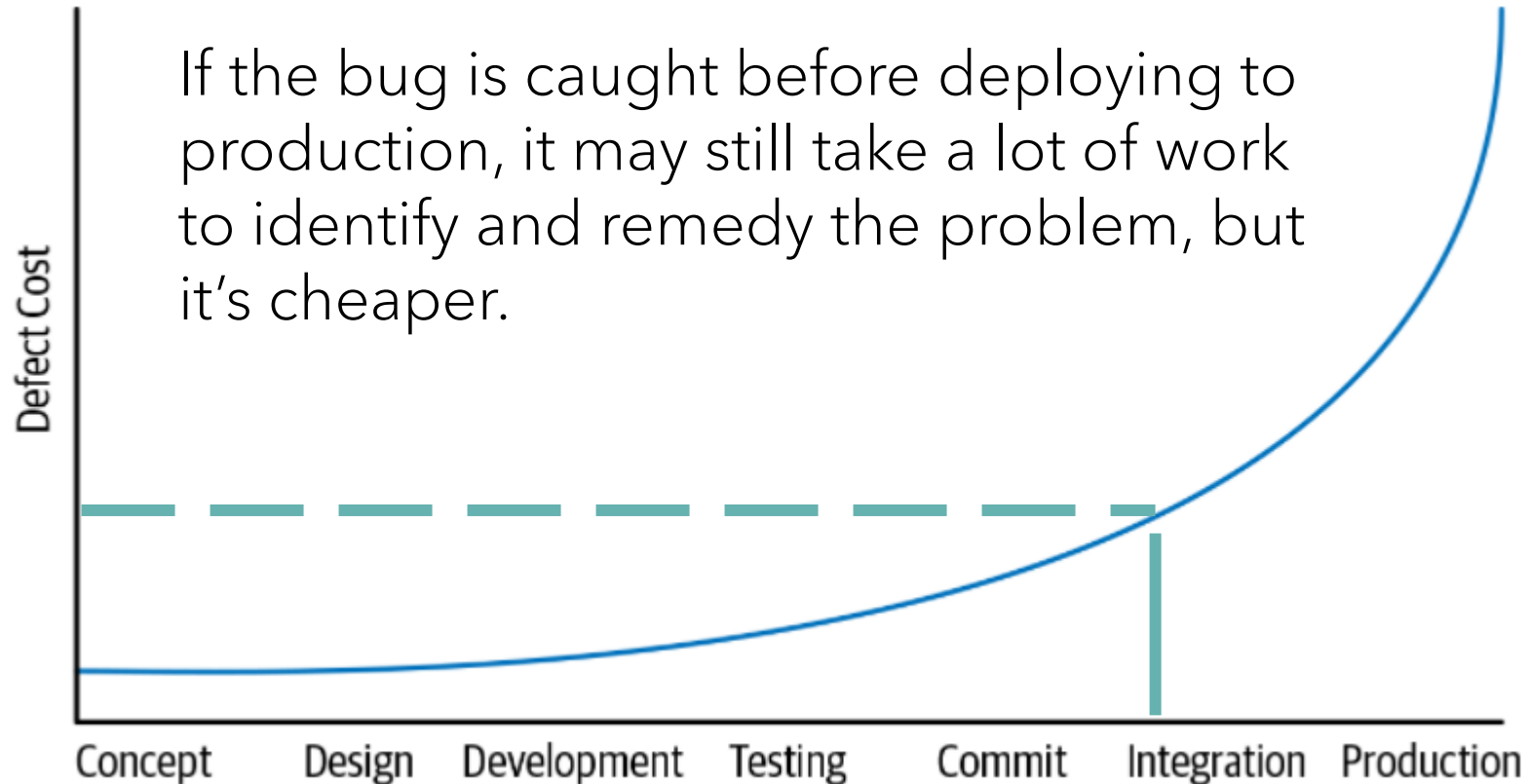


Figure 1-2. Timeline of the developer workflow

A GENERAL RULE OF REDUCING COST - SHIFT LEFT

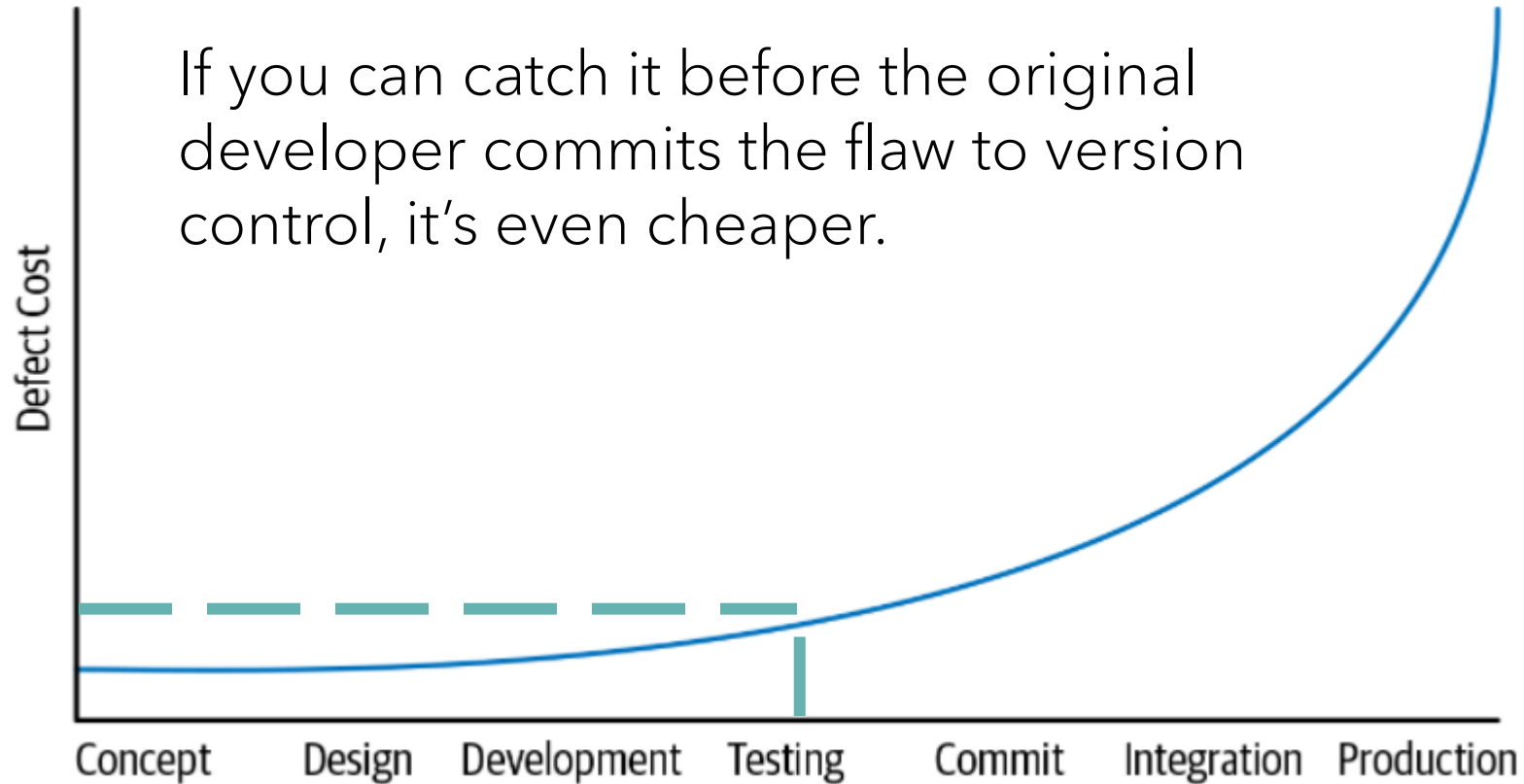


Figure 1-2. Timeline of the developer workflow

NEXT

- Version Control
- Build Systems