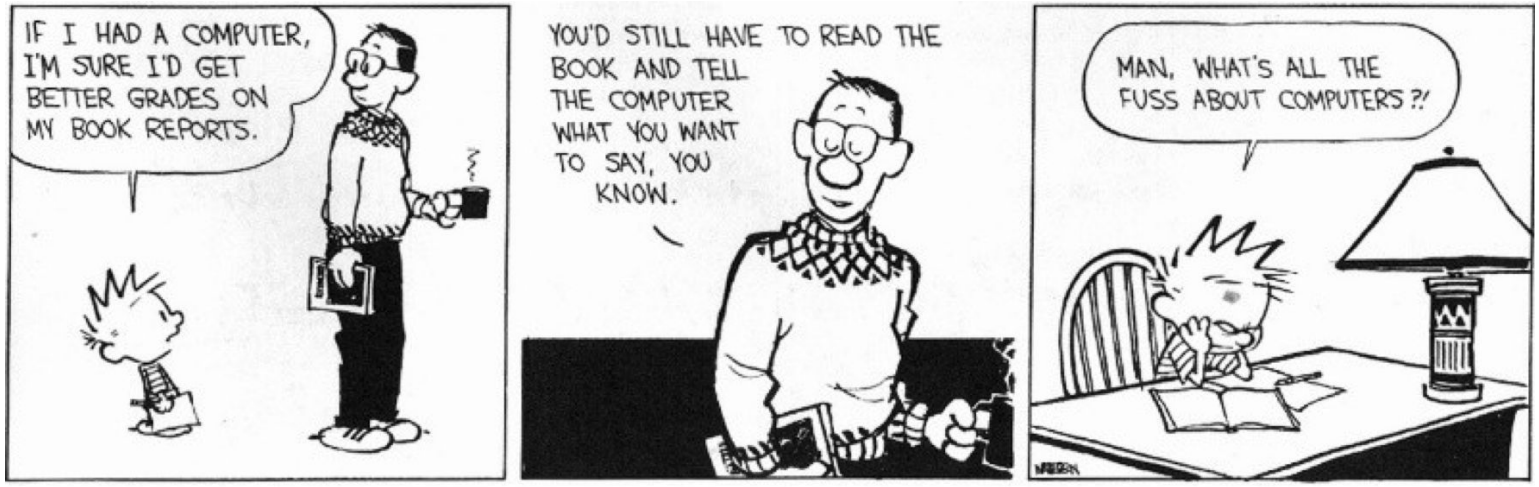
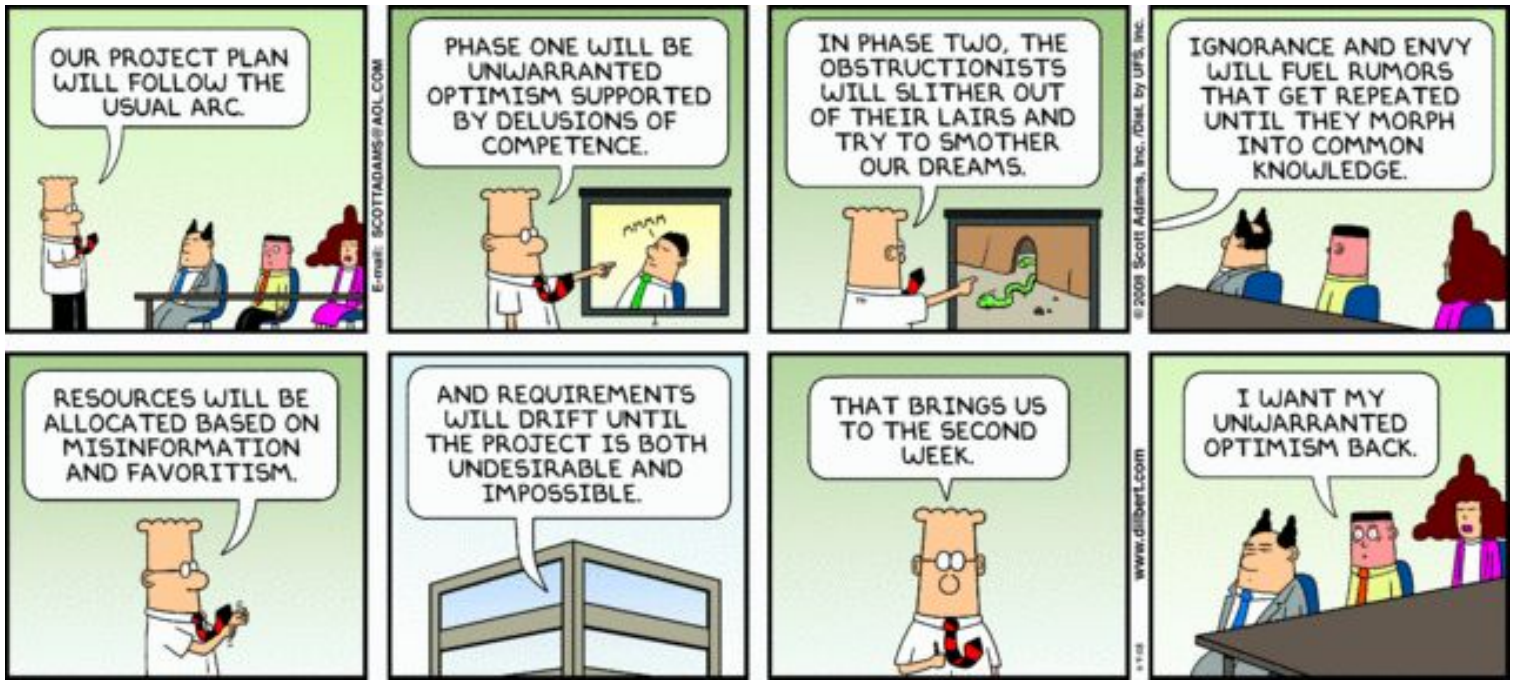


COMPSCI 320

Lec 01 - Intro to Software Engineering

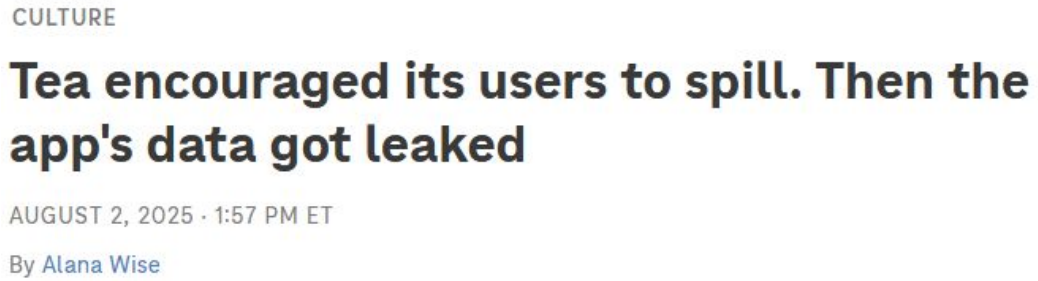
Slides from Madeline Endres & Gordon Anderson



What happened on July 25, 2025?



Software is Critical: Relationships



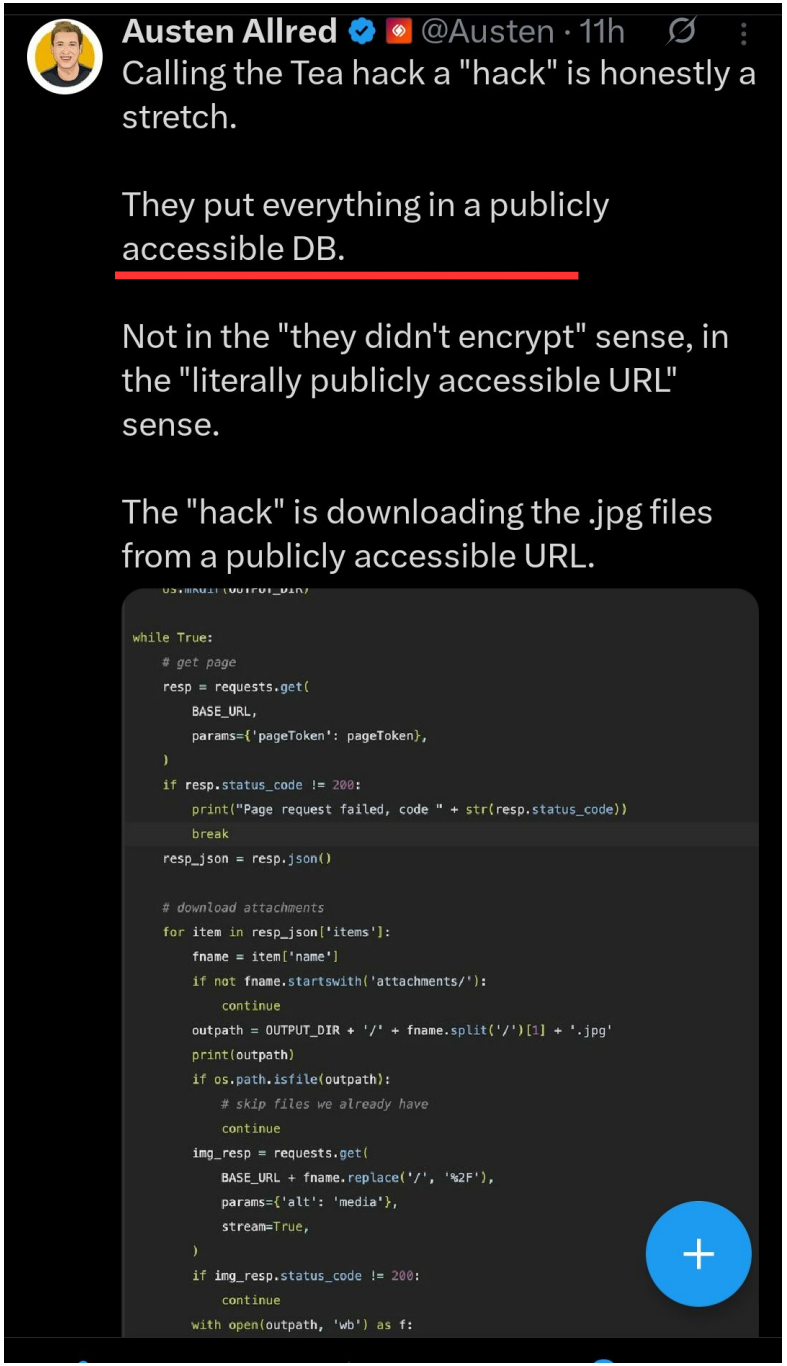
CULTURE

Tea encouraged its users to spill. Then the app's data got leaked

AUGUST 2, 2025 · 1:57 PM ET

By Alana Wise

- “Tea suffered a major data breach, revealing users' drivers' licenses, direct messages, selfies and other sensitive information.
- The team at Tea discovered the initial breach in the evening hours of July 25. As first reported by 404 Media, users on the notorious message board 4chan got a hold of users' sensitive data — including government IDs, which had at one point been used as a verification tool by the app — and leaked it elsewhere online.
- "As part of our ongoing investigation into the cybersecurity incident involving the Tea App, we learned that some direct messages (DMs) were accessed as part of the initial incident," a representative for the company told NPR.


Austen Allred @Austen · 11h
Calling the Tea hack a "hack" is honestly a stretch.
They put everything in a publicly accessible DB.
Not in the "they didn't encrypt" sense, in the "literally publicly accessible URL" sense.
The "hack" is downloading the .jpg files from a publicly accessible URL.
while True:
# get page
resp = requests.get(
BASE_URL,
params={'pageToken': pageToken},
)
if resp.status_code != 200:
print("Page request failed, code " + str(resp.status_code))
break
resp_json = resp.json()
# download attachments
for item in resp_json['items']:
fname = item['name']
if not fname.startswith('attachments/'):
continue
outpath = OUTPUT_DIR + '/' + fname.split('/')[1] + '.jpg'
print(outpath)
if os.path.isfile(outpath):
# skip files we already have
continue
img_resp = requests.get(
BASE_URL + fname.replace('/', '%2F'),
params={'alt': 'media'},
stream=True,
)
if img_resp.status_code != 200:
continue
with open(outpath, 'wb') as f:

Software is Critical: “Everything”

On 19 July 2024, American cybersecurity company CrowdStrike distributed a faulty update to its Falcon Sensor security software that caused widespread problems with Microsoft Windows computers running the software. As a result, roughly 8.5 million systems crashed and were unable to properly restart[1] in what has been called the largest outage in the history of information technology[2] and "historic in scale".[3]

The outage disrupted daily life, businesses, and governments around the world. Many industries were affected—airlines, airports, banks, hotels, hospitals, manufacturing, stock markets, broadcasting, gas stations, retail stores, and more—as were governmental services, such as emergency services and websites.[4][5] The worldwide financial damage has been estimated to be at least US$10 billion.[6]

Within hours, the error was discovered and a fix was released,[7] but because many affected computers had to be fixed manually,[8] outages continued to linger on many services.[9][10]

2024 CrowdStrike incident



Multiple blue screens of death caused by a faulty software update on baggage carousels at LaGuardia Airport, New York City

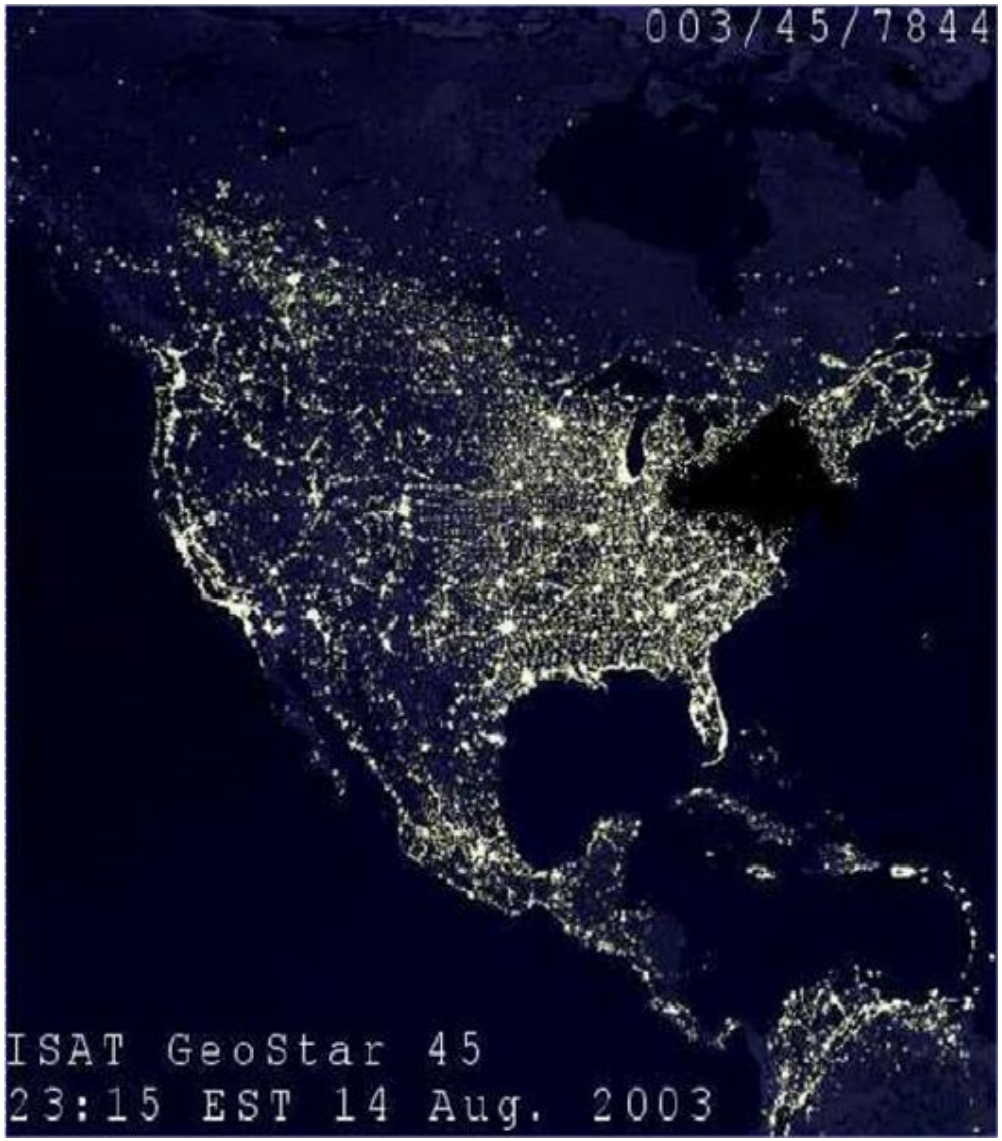
Date	19 July 2024
Location	Worldwide
Type	IT outage, computer crash
Cause	Faulty CrowdStrike software update
Outcome	~8.5 million Microsoft Windows operating systems crash worldwide, causing global disruption of critical services

What went wrong?

CrowdStrike's own post-incident investigation identified several errors that led to the release of a fault update to the "Crowdstrike Sensor Detection Engine":[13][*non-primary source needed*]

- The channel files were validated using Regex patterns with wildcards and loaded into an array instead of using a parser for this purpose.
- In the programming language C, the length of arrays must be treated and checked separately. However, the length was not checked before access. An array with 21 fields was expected, but the channel file was in an older data format with only 20 fields.
- In the unit tests, only the *happy path* was tested. Regression tests for compatibility with the older data format were not conducted.
- In manual tests, only valid data was tested.
- The channel files did not contain a version number field that was checked.
- There were no staggered rollouts, but the update was distributed to all customers simultaneously. Even critical infrastructure was not specially treated.
- The software does not access the system on Microsoft Windows through a suitable application programming interface but runs as a driver in ring 0 to have elevated privileges on the operating system. However, a crash in this area leads to a blue screen of death, which stops the operating system.

Find Western Mass

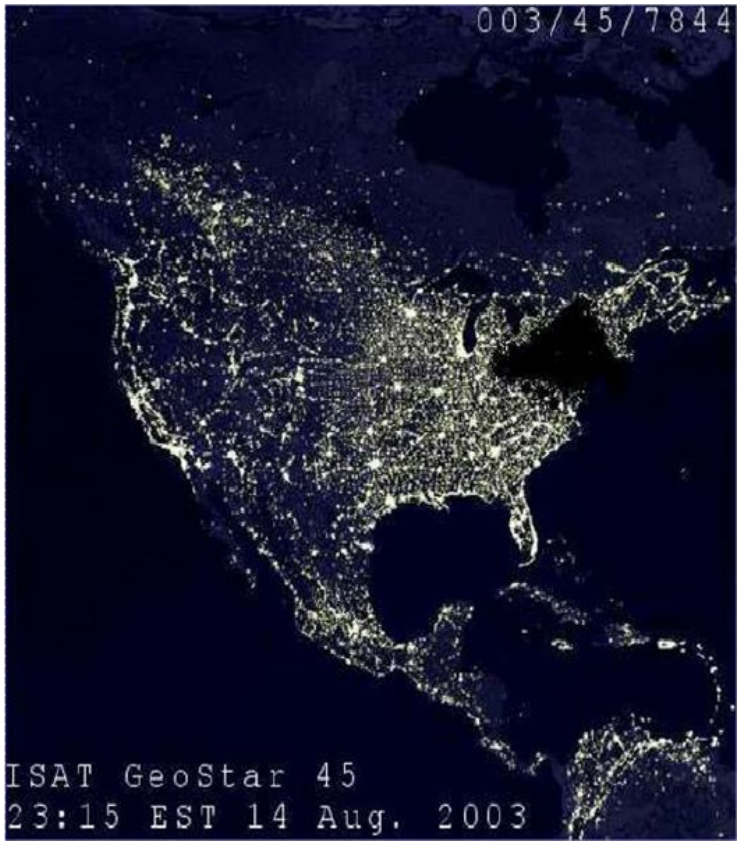


Software is Critical: Power

The **Northeast blackout of 2003** was a widespread power outage that occurred throughout parts of the Northeastern and Midwestern United States and the Canadian province of Ontario on Thursday, August 14, 2003, just after 4:10 p.m. EDT.[1]

Some power was restored by 11 p.m. Most did not get their power back until two days later. In other areas it took nearly a week or two for power to be restored.[2] At the time, it was the world's second most widespread blackout in history, after the 1999 Southern Brazil blackout.[3][4] The outage, which was much more widespread than the Northeast Blackout of 1965, affected an estimated 10 million people in Ontario and 45 million people in eight U.S. states.

The blackout's primary cause was a programming error or "bug" in the alarm system at the control room of FirstEnergy Corporation, an Akron, Ohio-based company. The lack of an alarm left operators unaware of the need to re-distribute power after overloaded transmission lines hit unpruned foliage, triggering a "race condition" in the energy management system software, a bug affecting the order of operations in the system. What would have been a manageable local blackout cascaded into massive widespread distress on the electric grid.



Software is Critical: Taylor Swift

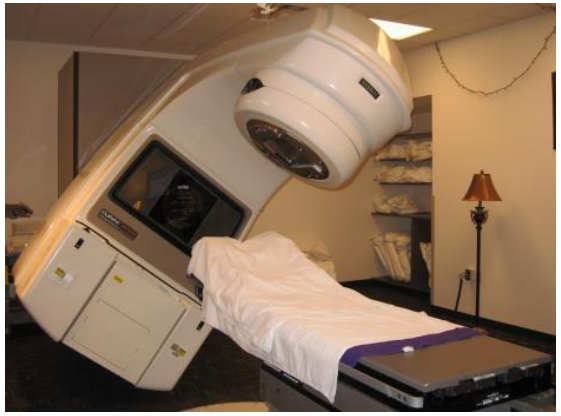


Website crash [edit]

The Eras Tour recorded an incredibly high demand for tickets, ultimately exposing significant issues in operational preparedness.[31][32] On November 15, the day of the pre-sale, Ticketmaster's website crashed following "historically unprecedented demand with millions showing up", halting the presale.[2] In less than an hour of availability, the ticketing platform's servers were "unable to answer", with users experiencing turbulence—"either completely logged out or in a queue 2,000-plus people strong that appeared frozen" as per *Rolling Stone*. Ticketmaster immediately published a statement saying they are working to fix the issues "as the site was unprepared to accommodate the sheer force of hundreds of thousands of Swift fans",[18] and subsequently reported that "hundreds of thousands of tickets" had already been sold and postponed the remainder of the presale, including the Capital One presale to November 16.[33] The public on-sale was later canceled due to "extraordinarily high demands on ticketing systems and insufficient remaining ticket inventory to meet that demand".[34] The Eras Tour sold over two million tickets on its first day of presale alone, breaking the all-time record for the most concert tickets sold by an artist in a single day.[3]

Software is Critical: Healthcare (!)

- Therac-25 radiation therapy machine
- At least six accidents in which patients were given massive overdoses of radiation
- Because of concurrent programming errors, it sometimes gave its patients radiation doses that were hundreds of times greater than normal, resulting in death or serious injury



What Is Software Engineering?



Consider:

In groups of 2, discuss some of the differences between:

1. A coding assignment in an intermediate programming course.

2. Building an application for a commercial organization.

Have one group member write down responses.

Some responses

Course Project

- Requirements specified up front

- Explicitly taught needed programming knowledge
- No real users
- Made individually or with a partner

- No need to maintain the code
- No need to document what you built for other developers
- Code won't be modified after submission

Commercial Application

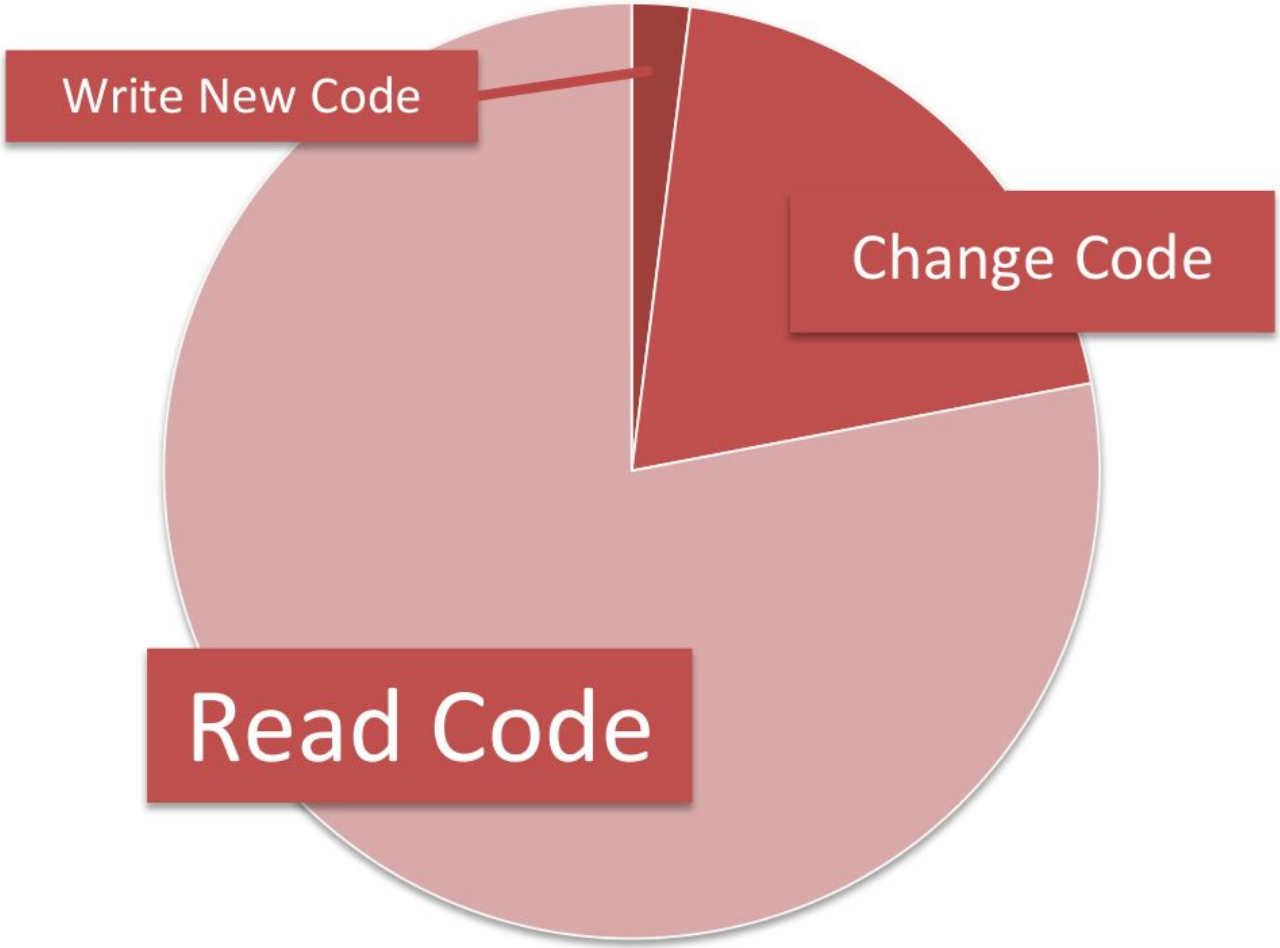
- You need to **learn the requirements** from the customer
- You may need to learn new languages and find resources!
- You have **real users!**
  - Need to consider security, privacy, user needs, scalability, reliability

- **Made by a team**!
- Needs to be **maintainable**
- Needs to be **documented**
- **Code will be modified over time**

What Is Software Engineering?

- The majority of industrial software engineering is *not* writing code.
- The dominant activities in software engineering are **comprehension** and **maintenance**.





"**Understanding code** is **by far** the activity at which professional developers spend most of their time."

[Peter Hallam. *What Do Programmers Really Do Anyway?* Microsoft.]

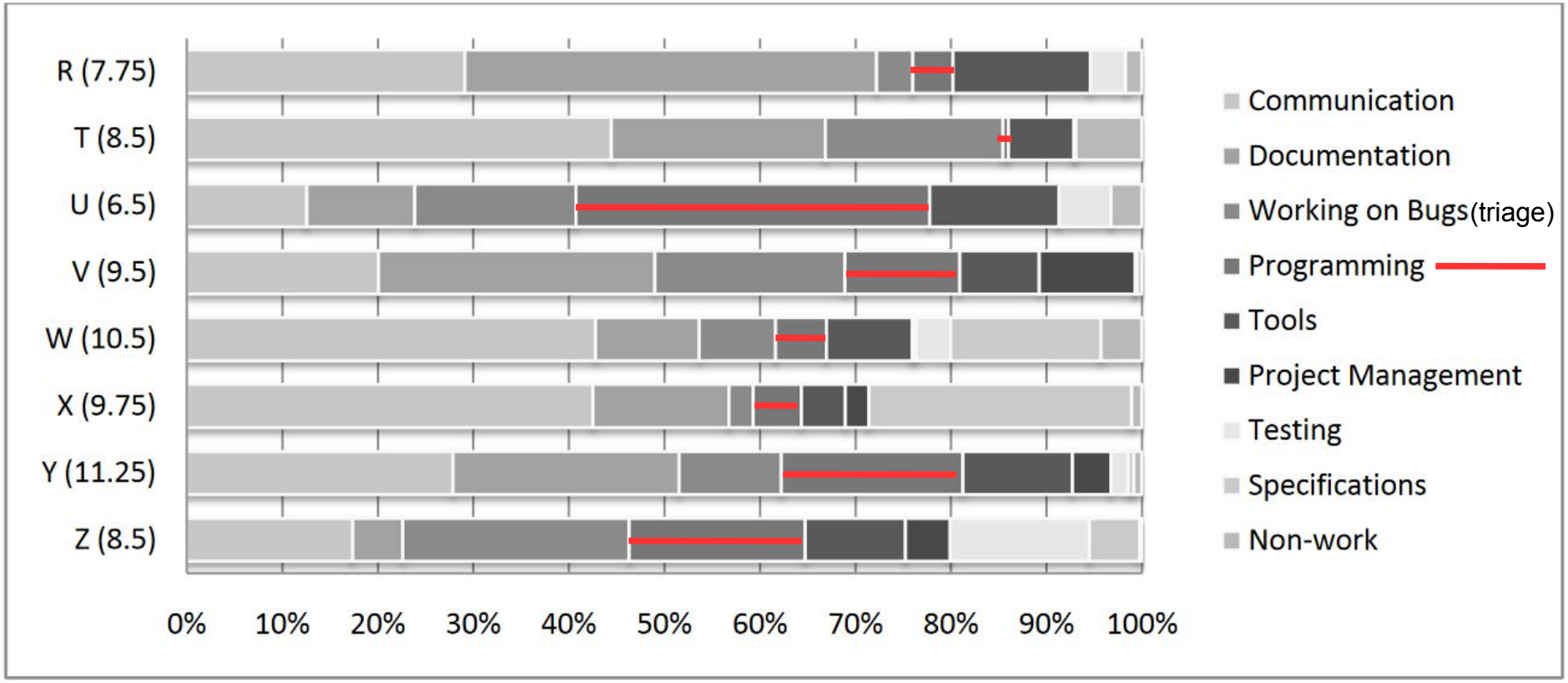


Figure 1. Tasks by time for *each* subject, normalized by the total time + time where events overlapped in each observation. Total observation time in hours is listed in parentheses after each subject's identification letter.

- Hour logging of new devs (1-7 months) at Microsoft: programming is 10-20% of the time.

[Begel and Simon. *Novice Software Developers, All Over Again*. Computing Education Research, September 2008. Microsoft.]

IV. How Developers Spend Their Time

Figure 3 summarizes the average distribution of activities of the developers and their sessions in our dataset.

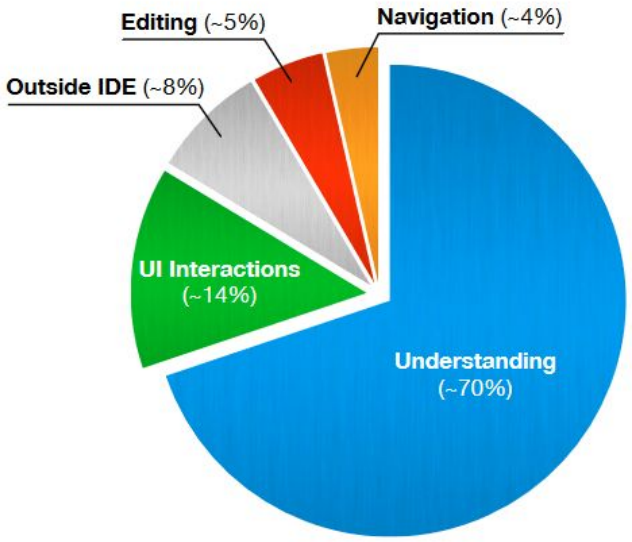
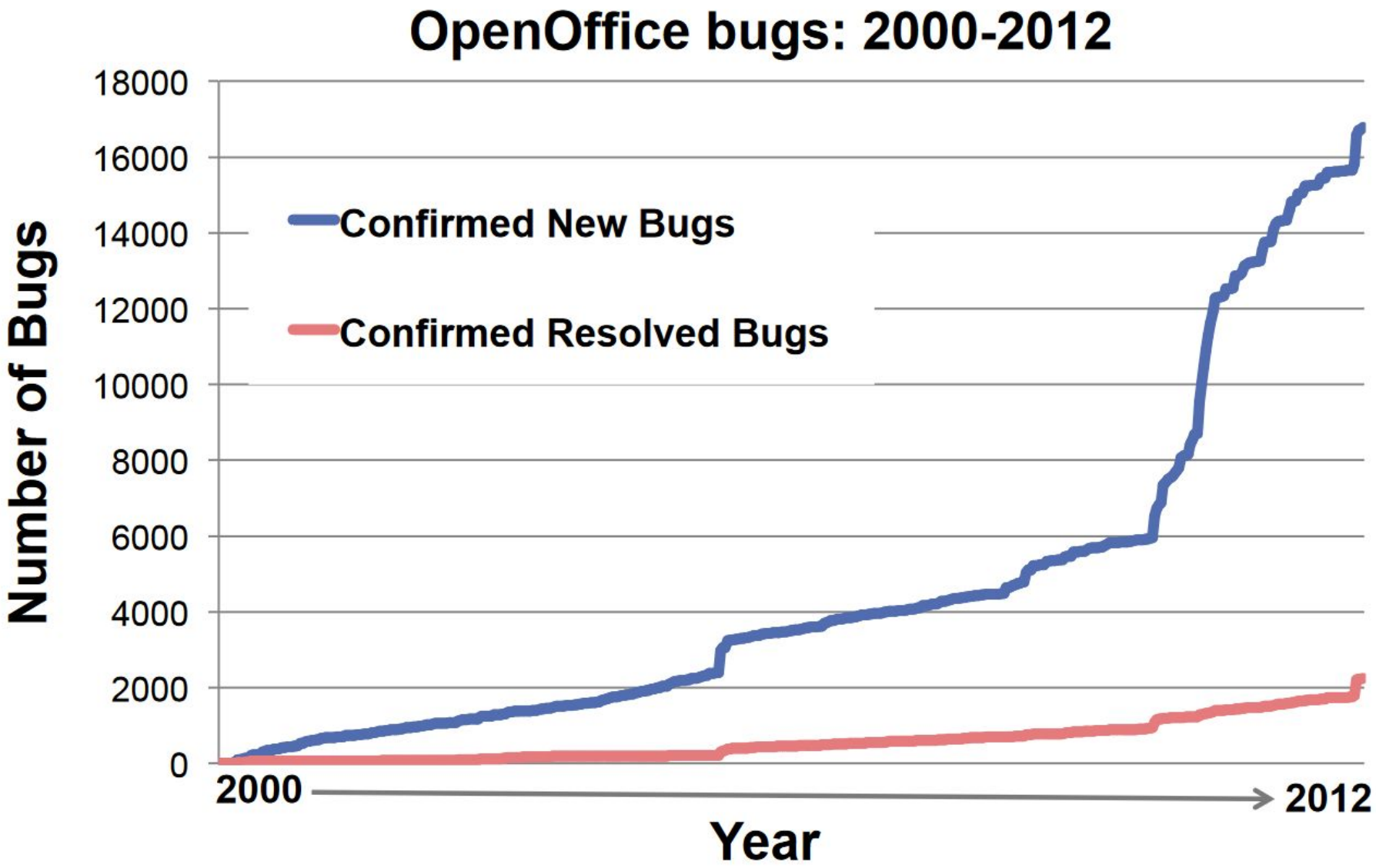


Fig. 3. How do developers spend their time?

into sprees and those into activities. In the end, 31,609 development activities originated from the 5 million events recorded with DFlow. We measured the time spent by developers in 5 distinct and disjunct categories: *understanding*, *navigation*, *editing*, *UI interactions*, and *time spent outside of the IDE*.

Our results reinforce common claims about the role of program understanding: On average, developers spend 70% of their time performing program comprehension. In addition, developers spend 14% of their time in fiddling with the UI of the IDE, which calls for novel and more efficient user interfaces. The time spent for editing and navigating source code is respectively 5% and 4%. The large part of development is occupied by mental processes (*i.e.*, understanding) and, in the remaining time, a developer has to deal with inefficient user interfaces to read, write, and browse source code. We

[Minelli, Mocci, Lanza. *I know what you did last summer: an investigation of how developers spend their time*. ICPC 2015.]



[Fry. *Leveraging Light-Weight Analyses to Aid Software Maintenance*. 2014]

Handling Complexity

- The biggest challenge in developing software is managing risk.
- Risk is a measure of the unknown.
- ***Discuss:*** What are some unknowns in building software?

Risk Factors

Do we know **what** to build?
Do we know **how** to build it?
Do we have the **resources** to build it?
Do we have the **time** to build it?

Rarely known exactly, must estimate.

Once we estimate, is it smooth sailing?

Anti-Patterns of Development

Anti-patterns are patterns of development that arise due to lack of following good process models.

“Fire Drill”	Management waits until the last possible moment to allow developers to proceed with design and implementation; then they want results almost immediately.
“Mushroom Management”	Keep the developers in dark and feed them fertilizer. End-user interaction prohibited.
“Death March”	The schedule is 50% too short, size of the staff is only half as large as necessary, budget is 50% too small, number of features is 50% greater than comparable successful projects.

http://antipatterns.com/dev_cat.htm

Handling Complexity

- Humans have found that **following an engineering process** is one of the best ways to handle complexity and to manage risk.
- There are many kinds of engineering processes for software development.
- We will look at a few and use one in particular in this course.
- First, let's look at what engineering is in general.

What is Engineering?

- The process of ***implementing*** an ***idea*** according to a set of ***specifications***.
- Building something that is ***needed***.
- This process should produce ***high-quality*** product(s).
- We should be able to ***repeat*** the same process with the same high-quality results for other projects.

Project Stakeholders

Major parties (or “entities”) with an interest in the system under development:

Customer: proposes the system.

Developer: analyzes, designs and builds the system.

User: who the system is designed for.

Investor: provides financial backing.

Note that these are *categories*, not individuals. There can be other stakeholders. Anyone who the system *significantly affects* can be considered a stakeholder.

“My favorite operational definition of engineering is '**design under constraint**.' Engineering is creating, designing what can be, but it is constrained by nature, by cost, by concerns of safety, reliability, environmental impact, manufacturability, maintainability, and many other such 'ilities.'”

[Bill Wulf, NAE President, The Urgency of Engineering Education Reform, 2008]

Software Engineering

“My favorite operational definition of engineering is '**design under constraint**.' Engineering is creating, designing what can be, but it is constrained by nature, by cost, by concerns of safety, reliability, environmental impact, manufacturability, maintainability, and many other such 'ilities.'”

[Bill Wulf, NAE President, The Urgency of Engineering Education Reform, 2008]

“[Software Engineering is] The Establishment and use of sound **engineering principles** in order to obtain, **economically,** software that is **reliable** and works **efficiently** on **real** machines.”

[Bauer 1975, S. 524]

Summary

- Software development is often a ***chaotic*** process that is difficult to quantify and control.
- Software Engineering is about managing ***risk***.
- Software Engineering is a ***human process***. It's not mainly about writing code.
- Software process models are an attempt to ***manage*** risk by defining steps in the process of development.
- Try to control variables of ***time and resources*** to produce high-quality software on time.