Software and Security Engineering

Computer Science Tripos Part 1a

Cambridge University

Ross Anderson

Aims

- Introduce students to software engineering, and in particular to the problems of building
  - large systems
  - real-time systems
  - safety-critical systems
  - systems to withstand attack by capable opponents
- Illustrate what goes wrong with case histories
- Study software and security engineering practices as a guide to how mistakes can be avoided

Objectives

- At the end of the course you should know how writing programs with tough assurance targets, or in large teams, or both, differs from the programming exercises you've done so far
- You should appreciate the waterfall, spiral and agile models of development as well as the value of development and management tools, and the economics of the development lifecycle

Objectives (2)

- You should understand the various types of bugs, vulnerabilities and hazards, how to find them, and how to avoid introducing them
- And be prepared for your 1b group project!
- And your part 2 project, and later courses in security, systems etc.
- And you should start absorbing the lore!

Resources

- Recommended reading: R Anderson, 'Security Engineering' (3nd edition 2020), chapters 1–4, 7–9, and 2nd edition chapters 25–26 (available on my web page)
- Other books in the syllabus booklet will be replaced by online material
- Each lecture will consist of several video segments plus links to further reading

Outline

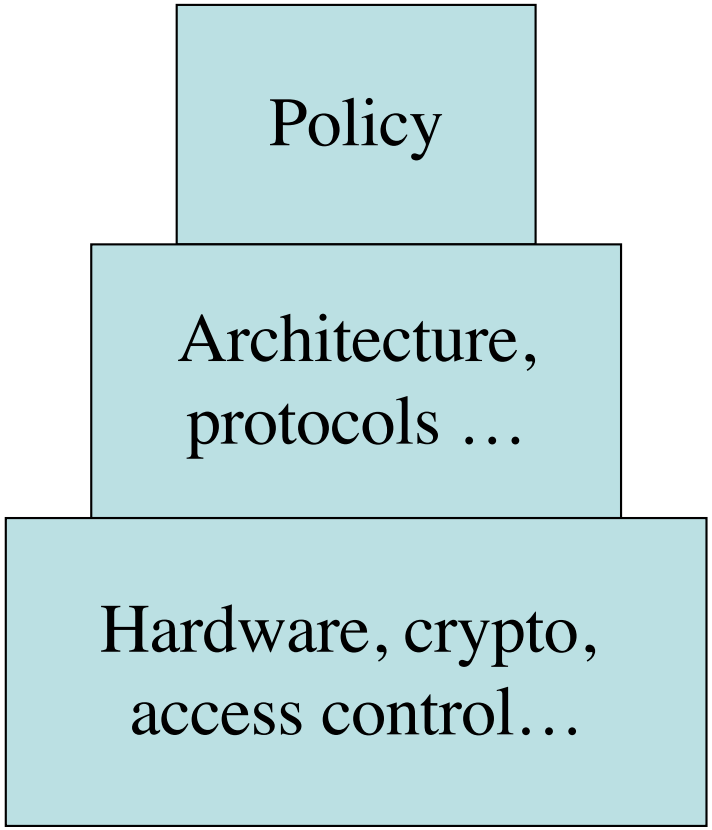
- Topics in logical order:
  - Security policy, safety case
  - Psychology and usability
  - Protocols, software bugs of different types
  - Safety engineering
  - Scale: the software crisis, software economics
  - Development: waterfall, agile, DevOps, …
  - Lecture from Dr Richard Sharp on SaaS
  - Critical systems: safety, security, sustainability

What is Security Engineering?

Security engineering is about building systems to remain dependable in the face of malice, error and mischance. As a discipline, it focuses on the tools, processes and methods needed to design, implement and test complete systems, and to adapt existing systems as their environment evolves.

Design Hierarchy

- What are we trying to do?
- How?
- With what?



Security vs Dependability

- The safety and security communities use different languages
- For us, dependability = reliability + security
- Reliability and security are often strongly correlated in practice
- But malice is different from error!
  - Reliability: "Bob will be able to read this file"
  - Security: "The Chinese Government won't be able to read this file"

Electric motor should not propel bicycle when speed > 15.5 mph

Clarifying terminology

- A *system* can be:
  - a product or component (PC, smartcard,…)
  - some products plus O/S, comms and infrastructure
  - the above plus applications
  - the above plus internal staff
  - the above plus customers / external users
- Common failing: policy drawn too narrowly

Clarifying terminology (2)

- A *subject* is a physical person
- A *person* can also be a legal person (firm)
- A principal can be
  - a person
  - equipment (PC, phone, smartcard, car…)
  - a role (the officer of the watch)
  - a complex role (Alice or Bob, Bob deputising for Alice)
- Sometimes you need to distinguish 'Bob's smartcard representing Bob who's standing in for Alice' from 'Bob using Alice's card in her absence'

Clarifying terminology (3)

- *Secrecy* is technical – mechanisms limiting the number of principals who can access information
- *Privacy* means control of your own secrets; 'informational self-determination'
- *Confidentiality* is an obligation to protect someone else's secrets
- Thus your medical privacy is protected by your doctors' obligation of confidentiality

Clarifying terminology (4)

- *Anonymity* has various meanings, from not being able to identify subjects to not being able to link their actions; it's often about access to metadata
- An object's *integrity* lies in its not having been altered since the last authorised modification
- *Authenticity* has two common meanings –
  - an object has integrity plus freshness
  - you're speaking to the right principal
- A cheque is an example of the first

Clarifying Terminology (5)

- *Trust* is hard! It has several meanings:
  1. a warm fuzzy feeling
  2. a trusted system or component is one that can break my security policy
  3. a trusted system is one I can insure
  4. a trusted system won't get me fired when it breaks
- I'm going to use number 2 (the NSA definition)
- E.g. a GCHQ person selling key material to a Chinese diplomat is trusted but not trustworthy (assuming their action was unauthorised)

Clarifying Terminology (6)

- An *error* is
  - a design flaw, or
  - a deviation from an intended state
- A *failure* is a nonperformance of the system, within specified environmental conditions
- *Reliability* is the probability of failure within a set period of time (typically mtbf, mttf)
- An *accident* is an undesired, unplanned event resulting in specified kind or level of loss

Clarifying Terminology (7)

- A *hazard* is a set of conditions on a system / its environment where failure can lead to an accident
- A *critical* system, process or component is one whose failure will lead to an accident
- *Risk* is the probability of an accident
- Thus: risk is hazard level combined with *danger* (probability hazard $\rightarrow$ accident) and *duration*; a metric might be the *micromort* (10^{-6} risk of death)
- *Uncertainty* is where the risk is not quantifiable
- *Safety* is simple: freedom from accidents

Clarifying Terminology (8)

- A *security policy* is a succinct statement of protection goals – typically less than a page of normal language
- A *protection profile* is a detailed statement of protection goals – typically dozens of pages of semi-formal language
- A *security target* is a detailed statement of protection goals applied to a particular system – and may be hundreds of pages of specification for both functionality and testing

Methodology 101

- Sometimes you do a top-down development. In that case you need to get the safety / security policy right in the early stages of the project
- Often it's iterative. Then the safety / security requirements can get ignored or confused
- In the safety-critical systems world there are methodologies for maintaining the safety case
- In both security and safety, the big problem is often maintaining dependability as the system – and the environment – evolve. (More on this later)

What often passes as ‘Policy’

1. This policy is approved by Management.
2. All staff shall obey this security policy.
3. Data shall be available only to those with a ‘need-to-know’.
4. All breaches of this policy shall be reported at once to Security.

What’s wrong with this?

Traditional government approach

- Start from the *threat model*: an insider who is disloyal (Burgess/MacLean, Aldrich Ames, Edward Snowden...) or careless (loose talk, reading secret papers on train, malware on PC...)
- So: limit the number of people you have to trust, and make it harder for them to be untrustworthy
- Basic idea since 1940: a clerk with 'Secret' clearance can read documents at 'Confidential' and 'Secret' but not at 'Top Secret'

Multilevel secure systems (MLS)

- Multilevel secure (MLS) systems are widely used in government
- They enforce standard handling rules for material at ‘Confidential’ ‘Secret’, ‘Top Secret’ etc.
- Resources have classifications; principals have clearances; clearance must equal or exceed classification; and information flows upwards only
- Enforcement independent of actions for most users
- Recall ‘mandatory access control’ from OS course

Formalising the Policy

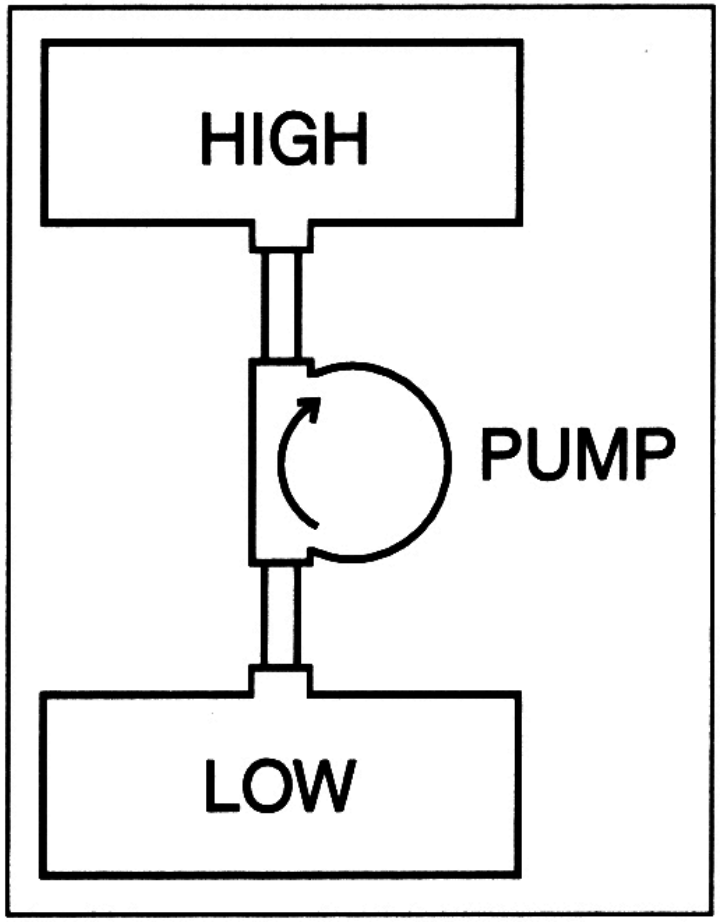
- Bell-LaPadula (1973):
  - *simple security policy*: no read up
  - **-policy*: no write down
- With these, one can prove that a system that starts in a secure state will remain in one
- Ideal: minimise the Trusted Computing Base (set of hardware, software and procedures that can break the security policy)

One problem: covert channels

- BLP lets malware move from Low to High, just not to signal down again!
- But: what if malware at High modulates shared resource (e.g. CPU usage) to signal to Low?
- And: how can you let messages from Low to High, if a delayed ack could be used to signal?
- Such a *covert channel* is a complex emergent property of whole systems. It limits the assurance we can get from information flow policies

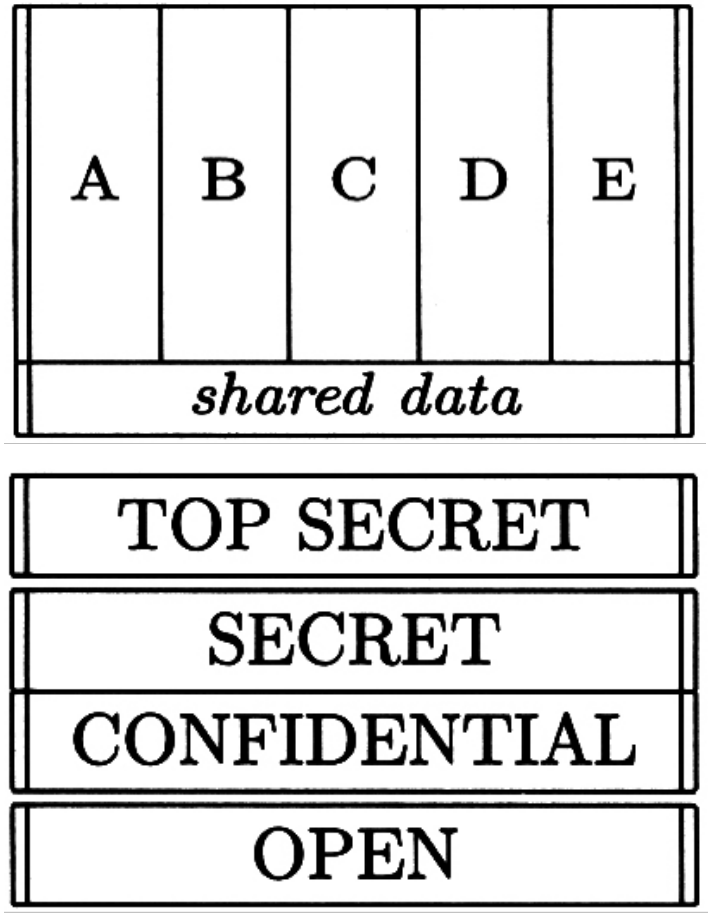
Typical MLS system

- Use architecture to get high assurance
- Idea: change a complex emergent property of the whole system into a simple property of a testable component
- But this is often harder than it looks!



Multilateral Security

- Sometimes the aim is to stop data flowing down
- Other times, you want to stop lateral flows
- Examples:
  - Intelligence, typically with compartments
  - Medical records
  - Competing clients of an accounting firm



Safety via Multilevel Integrity

- The Biba model – data may flow only down from high-integrity to low-integrity
- Dual of BLP: don't read down, or write up
- Examples:
  - Medical device with 'calibrate' and 'operate' levels
  - grid control with safety as highest level, operational control at next level, then billing etc
- Still about 'insiders' (errors, failures...)

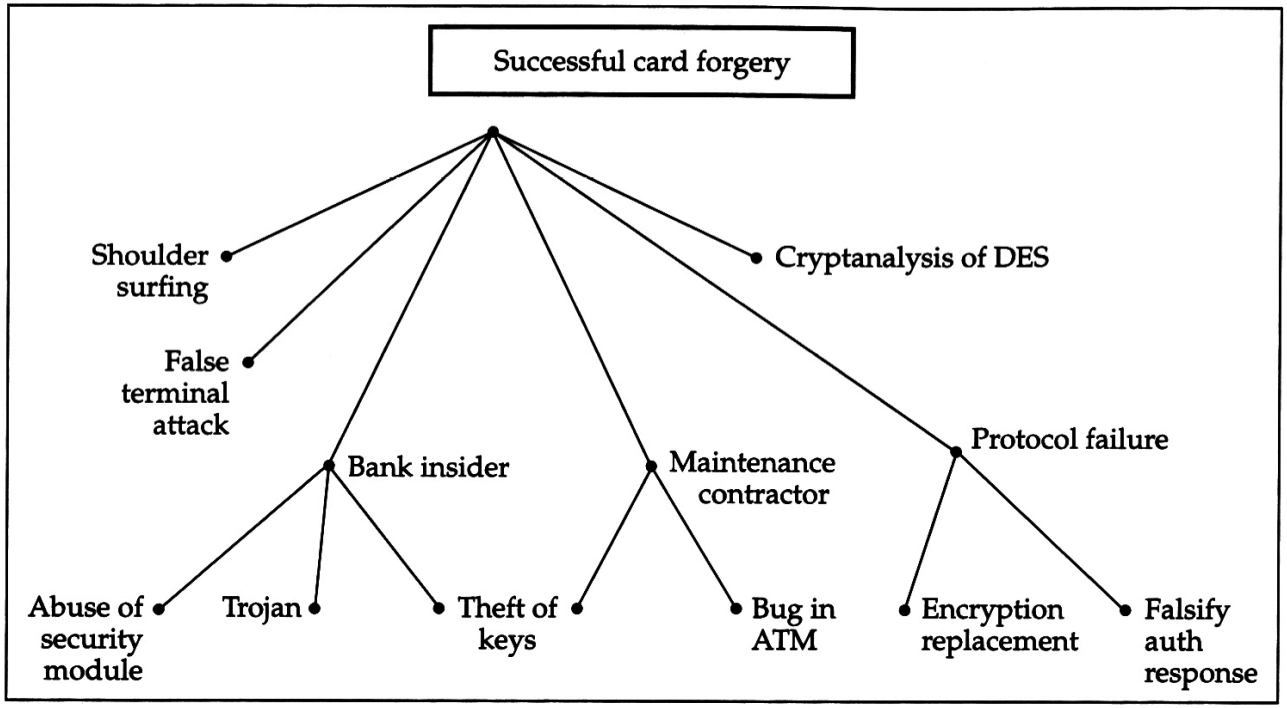
Architecture matters

- Lots of legacy protocols trust all network nodes
- E.g. DNP3 in control systems, CAN bus in cars
- IP address = trouble!
- Chrysler Jeep recall
- Bad node = trouble too
- So: separate subnets, capable firewalls

Safety policies

- Industries have their own standards, cultures, often with architectural assumptions
- Over 180 regulations for cars – e.g. ABS failure mustn't cause asymmetric braking
- In more mature industries safety standards tend to evolve
- Two approaches, depending on where the complexity is: top down or bottom up

Fault tree analysis (top down)



- Work back from each outcome we must avoid, to identify critical subsystems / staff / components
- This is the safety terminology; in security, a *threat tree*

Failure modes and effects analysis

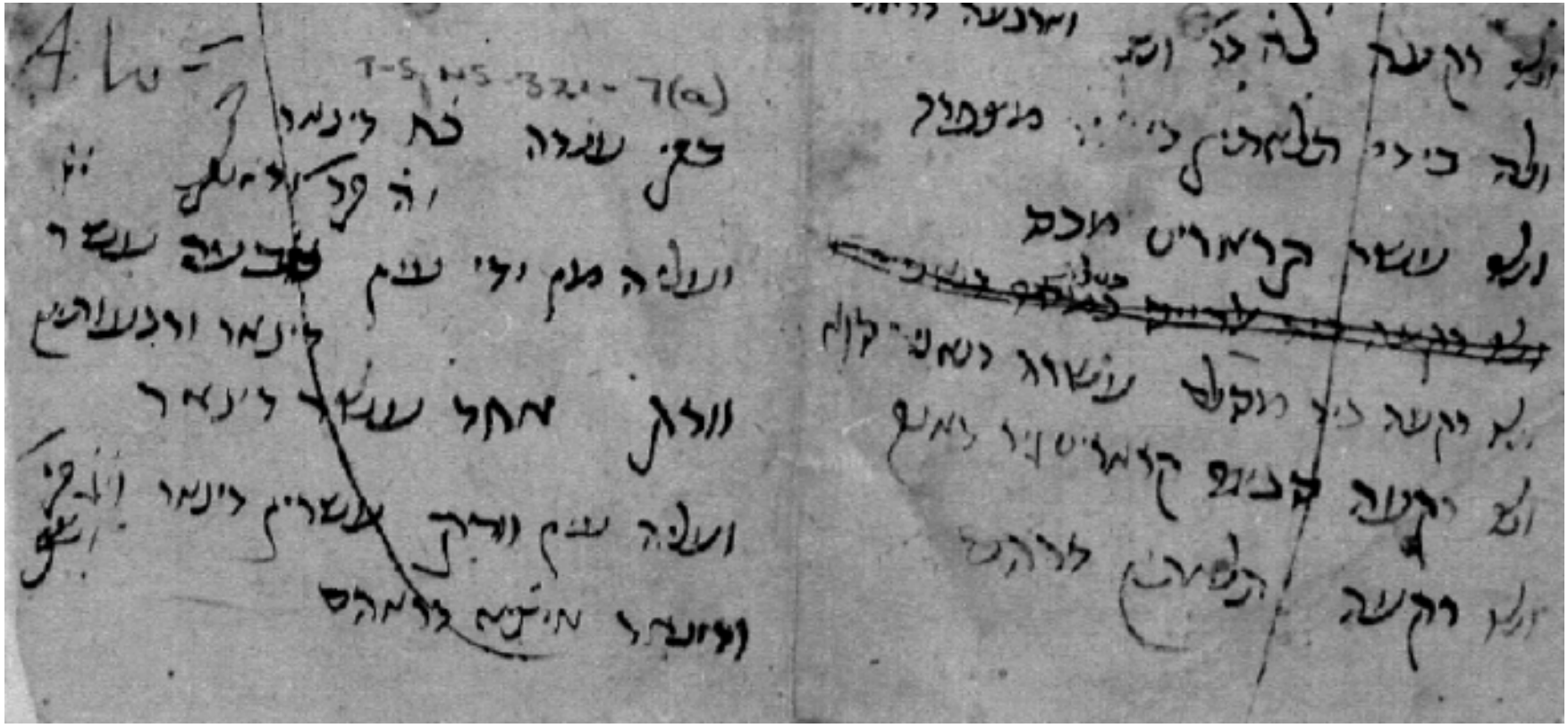
- The bottom-up approach is ‘failure modes and effects analysis’ (FMEA) – developed by NASA
- If you only have a few critical components you can just list all their failure modes: rocket motors, navigation, heat shields…
- Figure out what you’ll do about each
  - cut the probability by overdesign?
  - redundancy?
- Then work out how to deal with any interactions

Example – nuclear weapon safety

- Don't want Armageddon caused by a mad pilot, a stolen bomb, or a mad president
- So: for nuclear yield, we require
  - Authorisation: president/PM releases code
  - Environment: N seconds zero gravity
  - Intent: pilot puts key in bomb release
- Independent, simple, technical mechanisms tied to a control point

Bookkeeping, c. 3300 BC

Genizah Collection – c. 1100 AD



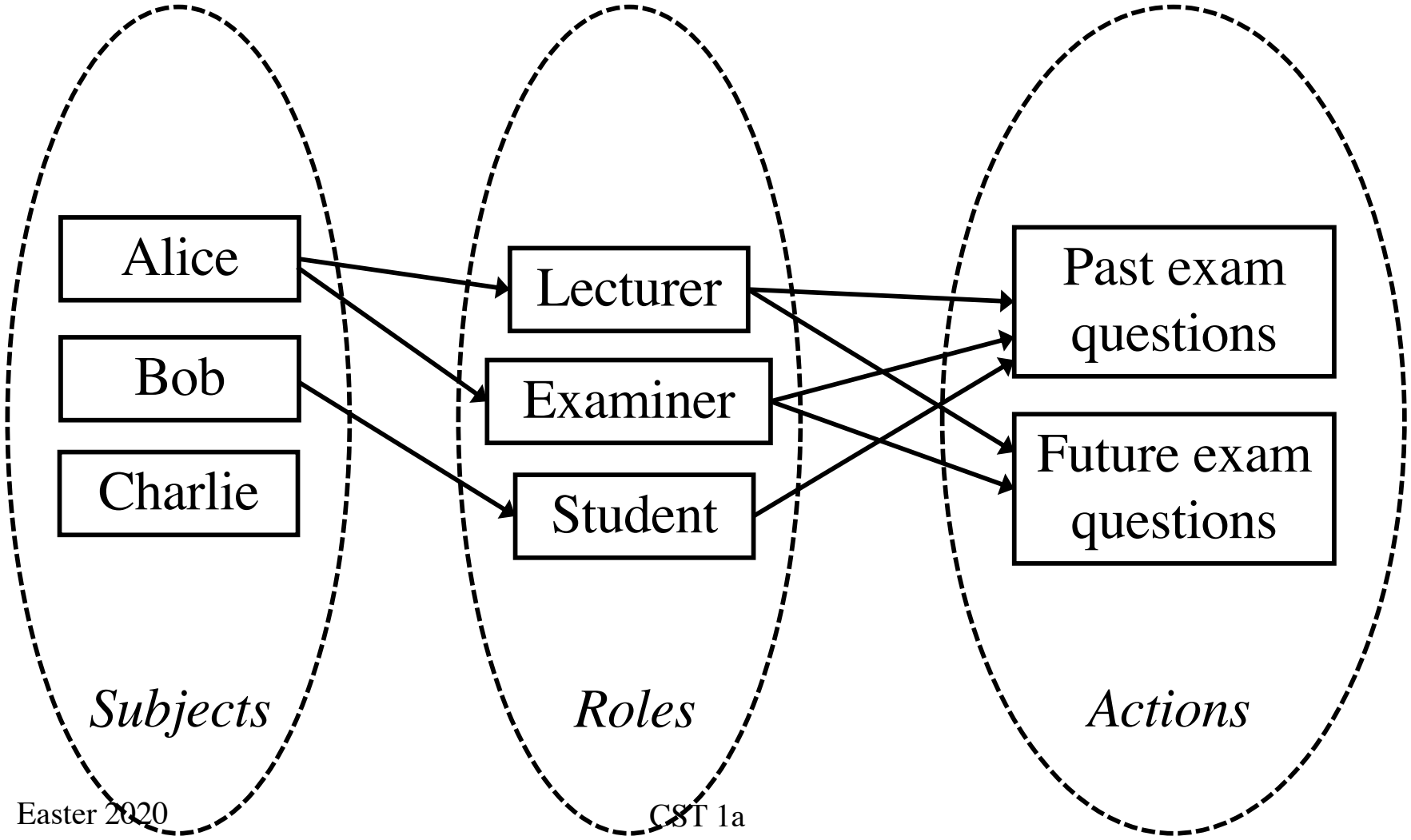
Double-entry bookkeeping

- How do you manage a business that's grown too big to staff with your own family members?
- Double-entry bookkeeping – each entry in one ledger is matched by opposite entries in another
  - E.g. firm sells £100 of goods on credit – credit the sales account, debit the receivables account
  - Customer pays – credit the receivables account, debit the cash account
- So bookkeepers have to collude to commit fraud

Separation of duties in practice

- Serial:
  - Lecturer gets money from EPSRC, charity, …
  - Lecturer gets Old Schools to register supplier
  - Gets stores to sign order form and send to supplier
  - Stores receives goods; Accounts gets invoice
  - Accounts checks delivery and tell Old Schools to pay
  - Lecturer gets statement of money left on grant
  - Audit by grant giver, university, …
- Parallel: two signatures (e.g. where transaction large, irreversible – as in bank guarantee)
- How would you design such a system?

Role-Based Access Control (RBAC) decouples policy and mechanism



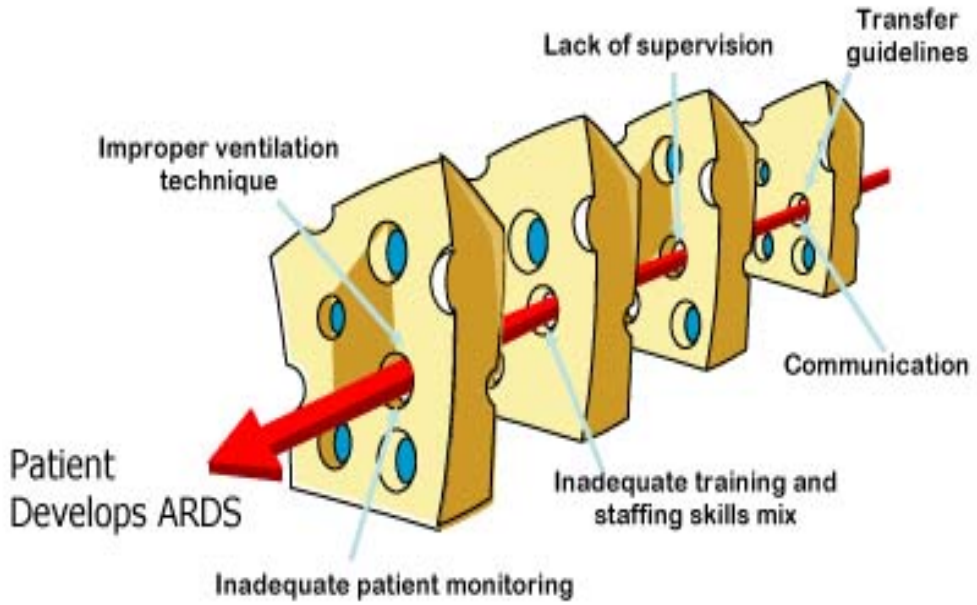
Scaling to big organisations

- Role-Based Access Control (RBAC) adds an extra indirection layer: ‘officer of the watch’, ‘branch accountant’, ‘charge nurse’
- Instead of managing 100,000 staff, you write a policy to manage a few dozen roles
- You still need to devise a good policy!
- Many operating systems offer support for RBAC, MLS etc – and there’s also internal technical use

Summary: security / safety policy

- What are we trying to do?
- Security: threat model, security policy
- Safety: hazard analysis, safety standard
- Refine to protection profile, safety case
- Typical mechanisms: usability engineering, firewalls, protocols, access controls...
- Make sure they work together!

Defence in Depth

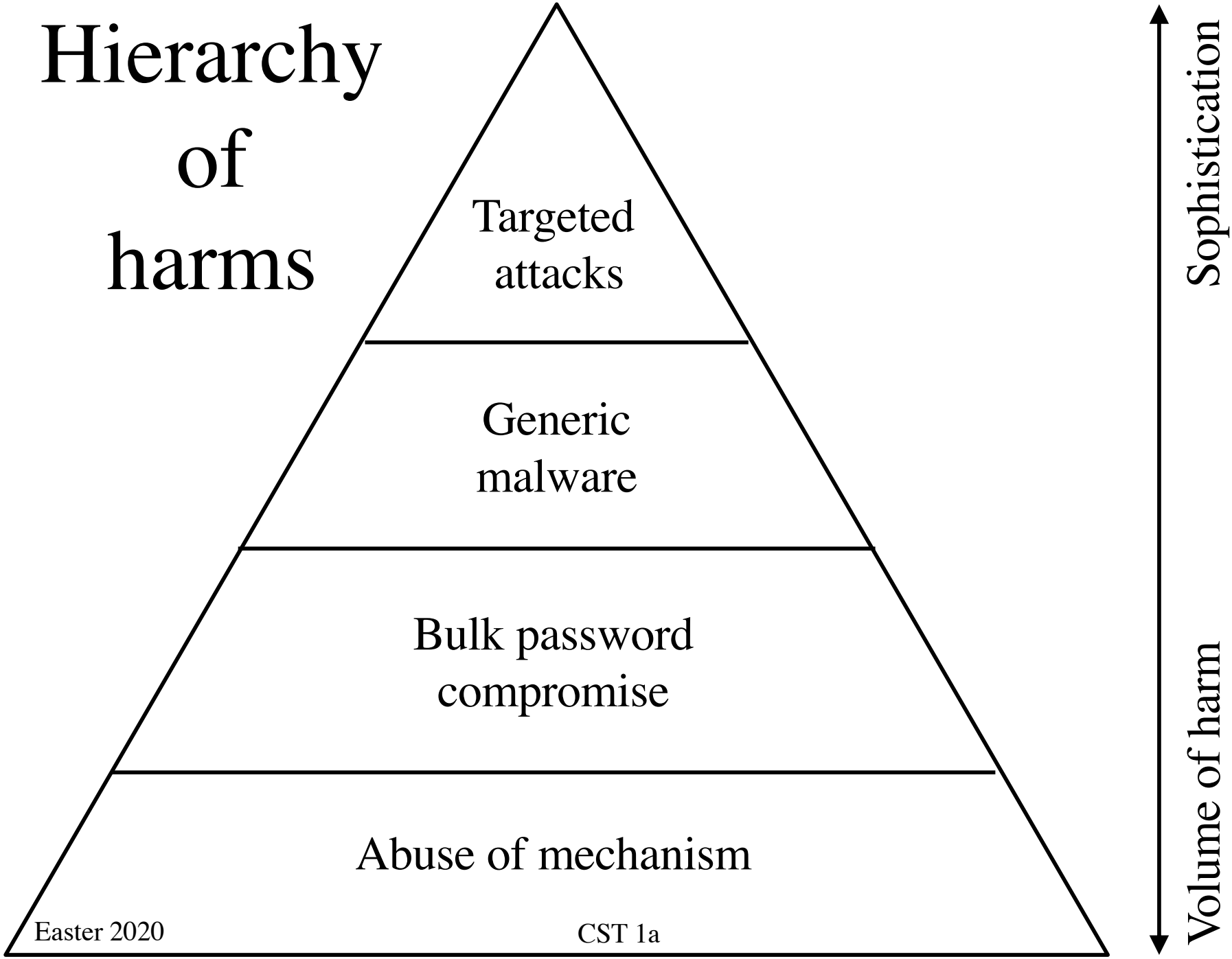


- Reason's 'Swiss cheese' model
- Stuff fails when holes in defence layers line up
- Thus: ensure human factors, software, and procedures complement each other (more later!)

Safety, security and human behaviour

- It can be tempting to ignore ‘user error’
- Most car crashes are user error, but we provide seat belts, airbags and crumple zones
- Compare 1959, 2009 Chevrolets in video
- Banks for years told victims of fraud “Our systems are secure so it must be your fault”
- Bank regulators too are now pushing back!

Hierarchy of harms



Abuse of standard mechanisms

- Just as a car crash is ‘abuse of mechanisms provided’, so are most scams and abuses
- Cambridge problem: crook runs website offering flat to let, so you send some money
- What can we do about cyber-bullying?
- Or doxxing?
- Or email telling your uncle of a lottery win?

Bulk password compromise

- In June 2012, 6.5m LinkedIn passwords stolen, cracked (encryption did not have a salt) and posted on a Russian forum
- Method: SQL injection (will discuss later)
- Passwords reused on other sites, from mail services to PayPal, were exploited there
- There have been many, many such exploits!
- What can we do about password reuse?

Phishing and social engineering

- Card thieves call victims to ask for PINs
- Generic phishing has been around since 2005
- A well-crafted lure sent to company staff ('from' the boss, etc) can get 30% yield
- Personalized to target: can be over 50%
- Some big consequences, e.g. John Podesta
- During analysis, try to think like a crook!

Usability of security / privacy advice

- Privacy law (summary): consent or anonymise (more in 1b ELE course)
- Both are much harder than they look, and get harder still as systems get more complex
- Automated collection by IoT devices, other people's phones etc makes it all harder still
- Look for privacy policies. How many give you any real choice?

Easter 2020

CST 1


CareFusion
Alaris® PC
PRIMARY INFUSION
RATE
VTBI
Press START


Options
Total Volume
Occlusion Alarm Setting
Primary
Secondary Bolus
Rate
Volume to be Infused
Run
Hold
On Off Charge
Battery
Silence
GRASEBY
500
Modular Infusion Pump


P Bag Vol: 250ml
Volume Left 246.9
A Infused 3.1
Press OK to start
START OK
STOP NO
BOLUS
ON OFF
BodyGuard 545


DOOR OPEN
CLOSE CLAMP
ALARM PUMPING ALERT
BACK LIGHT
SILENCE
TOT VOL STATUS
CLEAR TOT VOL
125
1000
TIME
STOP
ON OFF CHARGE
CLR
OPTIONS


SILENCE
STOP
START
POWER
ABBOTT
GEMSTAR
1-800-241-4002
ON / OFF
BACK-UP
CHANGE
OPTIONS
PURGE
HELP
NO
YES ENTER


1 SELECT mg/mL
2 SELECT µg/mL
3 SELECT mL
HOSPIRA
aim plus
OPTIONS
STOP
START
HELP
BACK-UP
CHANGE
SILENCE
PRIME
NO
YES/ENTER


Epidural
A:Bupivicaine/Fentanyl
500ml
Rate :
0 mL/hr
Pt Bolus 0
mmHg 0 720
ON OFF
info
STOP NO
START OK
PRIME BOLUS
cme medical
BODY GUARD 545


P Bag Vol: 250ml
Volume Left 246.9
A Infused 3.1
Press OK to start
START OK
STOP NO
BOLUS
ON OFF
BodyGuard 545


Epidural 11:35 90%
A:Bupivicaine/Fentanyl
500ml
Rate :
0 mL/hr
Pt Bolus 0
mmHg 0 720
ON OFF
info
STOP NO
START OK
PRIME BOLUS
cme medical
BODY GUARD 545

Medical device safety

- Usability problems with medical devices kill about the same number of people as cars do
- Biggest killer nowadays: infusion pumps
- Nurses typically get blamed, not vendors
- Avionics are safer, as incentives are more concentrated
- Read Harold Thimbleby's paper!

Psychology of safety and security

- Errors arise at different levels of the ‘stack’
  - We deal with novel problems in a conscious way
  - Frequently encountered problems are dealt with using rules we evolve, and are partly automatic
  - Over time, the rules give way to skill
- Our ability to automatise routine actions leads to absent-minded slips, or following a wrong rule
- There are also systematic limits to rationality in problem solving – ‘heuristics and biases’
- There’s social psychology too

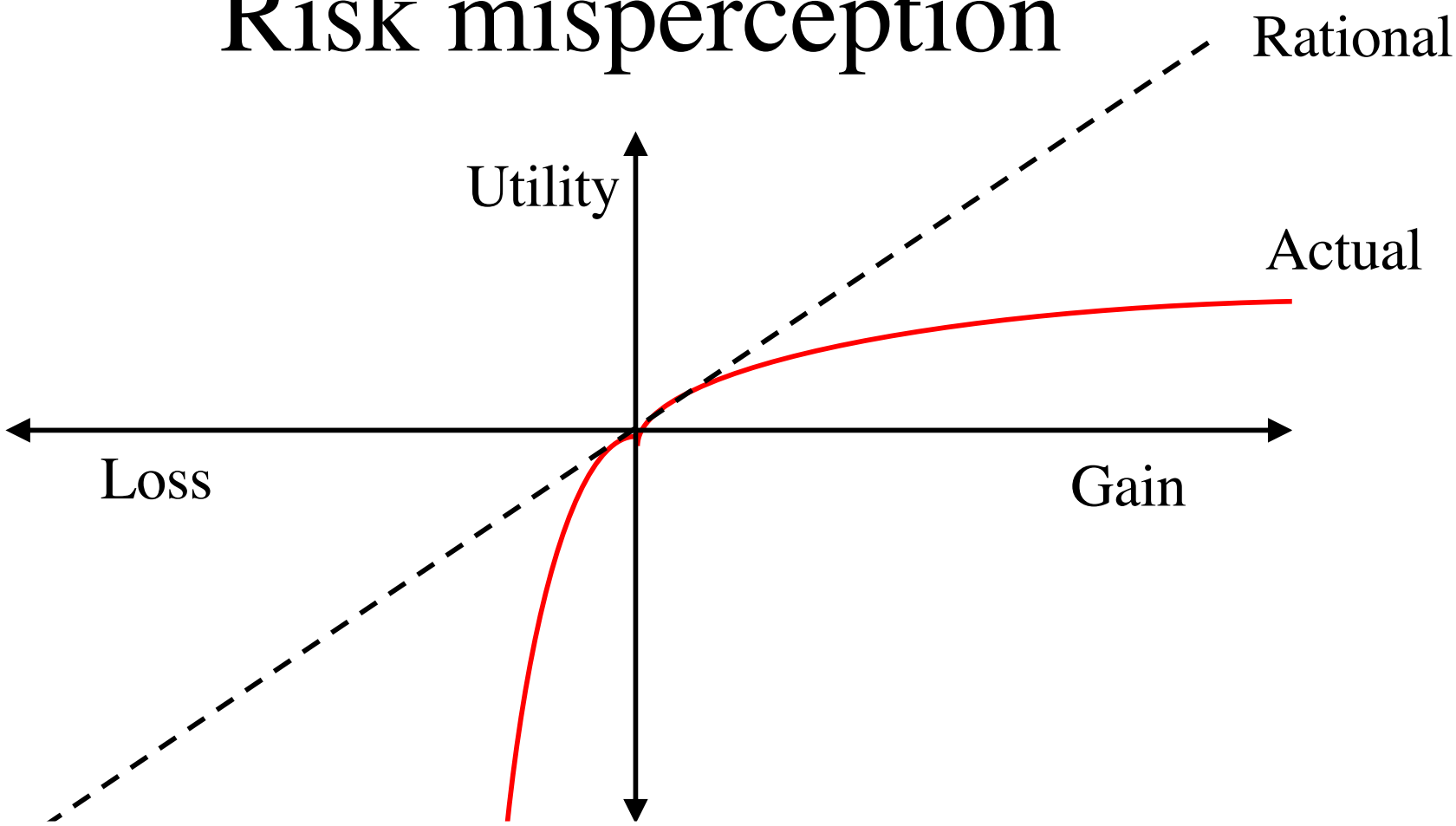
Error rates

- Skill is more reliable than knowledge, and practice really helps! Error rates (motor industry):
  - Inexplicable errors, stress free, right cues – 10^{-5}
  - Regularly performed simple tasks, low stress – 10^{-4}
  - Complex tasks, little time, some cues needed – 10^{-3}
  - Unfamiliar task dependent on situation, memory – 10^{-2}
  - Highly complex task, much stress – 10^{-1}
  - Creative thinking, unfamiliar complex operations, time short & stress high – ~1

Error types

- Slips and lapses
  - Forgetting plans, intentions; strong habit intrusion
  - Misidentifying objects, signals (often Bayesian)
  - Retrieval failures; tip-of-tongue, interference
  - Premature exits from action sequences, e.g. ATMs
- Rule-based mistakes; applying wrong procedure
- Knowledge-based mistakes
- Heuristics and biases based on how brains work!
- E.g. *prospect theory* models risk misperception

Risk misperception



People offered £10 or a 50% chance of £20 usually prefer the former; if offered a loss of £10 or a 50% chance of a loss of £20 they tend to prefer the latter!

Framing decisions about risk

- Decisions are heavily influenced by framing. E.g. the 'Asian disease problem' where the subject is making decisions on vaccination. Two options put to subjects. First:

  A: "200,000 lives will be saved"

  B: "with p=1/3, 600,000 saved; but p=2/3 none saved"
- Here 72% choose A over B!
- Second option is

  C: "400,000 will die"

  D: "with p =1/3, no-one will die, p=2/3, 600,000 die"
- Here 78% prefer D over C!
- This is also why marketers talk 'discount' or 'saving' – and fraudsters know that people facing losses take more risks

Social psychology

- The social brain hypothesis
- Authority: Stanley Milgram showed that over 60% of all subjects would inflict a potentially fatal shock on a ‘student’ if ordered to do so by a ‘teacher’
- Philip Zimbardo’s Stanford Prison Experiment suggested that roles alone might be enough!
- Conformity: Solomon Asch showed most people would deny obvious facts (like relative line length) to conform with others

Social psychology (2)

- Reciprocation is built-in: even monkeys do tit-for-tat! So give a gift to get things going
- People prefer to believe people they like, or people like them, and seek out opinions that confirm their existing beliefs
- Salespeople push for a commitment now as this can leave the sales prospect feeling a need for consistency later
- Scarcity can also spur people to action
- The above are all standard sales techniques!

Fraud psychology

- All the above plus
  - Appeal to the mark's kindness
  - Appeal to the mark's dishonesty
  - Distract them so they act automatically
  - Arouse them so they act viscerally
- See Stajano and Wilson on hustling, and "The Real Hustle" videos on YouTube; for more detail, see Modic and Lea

Users' mental models

- Explore how your users see the problem – the 'folk beliefs'
  - threats seen as 'viruses' which could be mischievous, or crime tools;
  - 'hackers' who may be seen as graffiti artists or burglars or targeting big fish;
  - Or simply as 'bad neighbourhoods' online!
- People are more likely to follow security advice consistent with their mental model

Affordances: Johnny Can’t Encrypt

Why Johnny Can’t Encrypt:
A Usability Evaluation of PGP 5.0

Alma Whitten
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
alma@cs.cmu.edu

J. D. Tygar[1]
EECS and SIMS
University of California
Berkeley, CA 94720
tygar@cs.berkeley.edu

Abstract

User errors cause or contribute to most computer security failures, yet user interfaces for security still tend to be clumsy, confusing, or near-nonexistent. Is this simply due to a failure to apply standard user interface design techniques to security? We argue that, on the contrary, effective security requires a different

1 Introduction

Security mechanisms are only effective when used correctly. Strong cryptography, provably correct protocols, and bug-free code will not provide security if the people who use the software forget to click on the encrypt button when they need privacy, give up on a

The power of defaults

- What actions do you make natural?
- Most people won't opt in, or opt out; they go with the default
  - Governments try to set socially optimal defaults (e.g. you must opt out of pensions)
  - Facebook privacy settings: advertiser-friendly?
  - What else? (discuss in supervisions)
- What might be done about this?

Economics versus psychology

- Most people don't worry enough about computer security
- How could this be fixed, and why is it not likely to be?
- Most people worry too much about terrorism
- How could this be fixed, and why is it not likely to be?

The compliance budget

- 'Blame and train' is not the best approach!
- It's often rational to ignore warnings
- People will spend only so much time obeying rules, so choose the rules that matter
- Rule violations are often an easier way of working, and sometimes necessary, so watch them, measure them and adapt to them
- The 'right' way of working should be easiest; the defaults should be safe

Where should the path be?

Differences between people

- Both risk appetite and the ability to perform certain tasks varies widely across subgroups of the population, including by
  - age
  - gender
  - education
- So if you give customers complex password rules and anti-phishing advice, you might discriminate illegally!

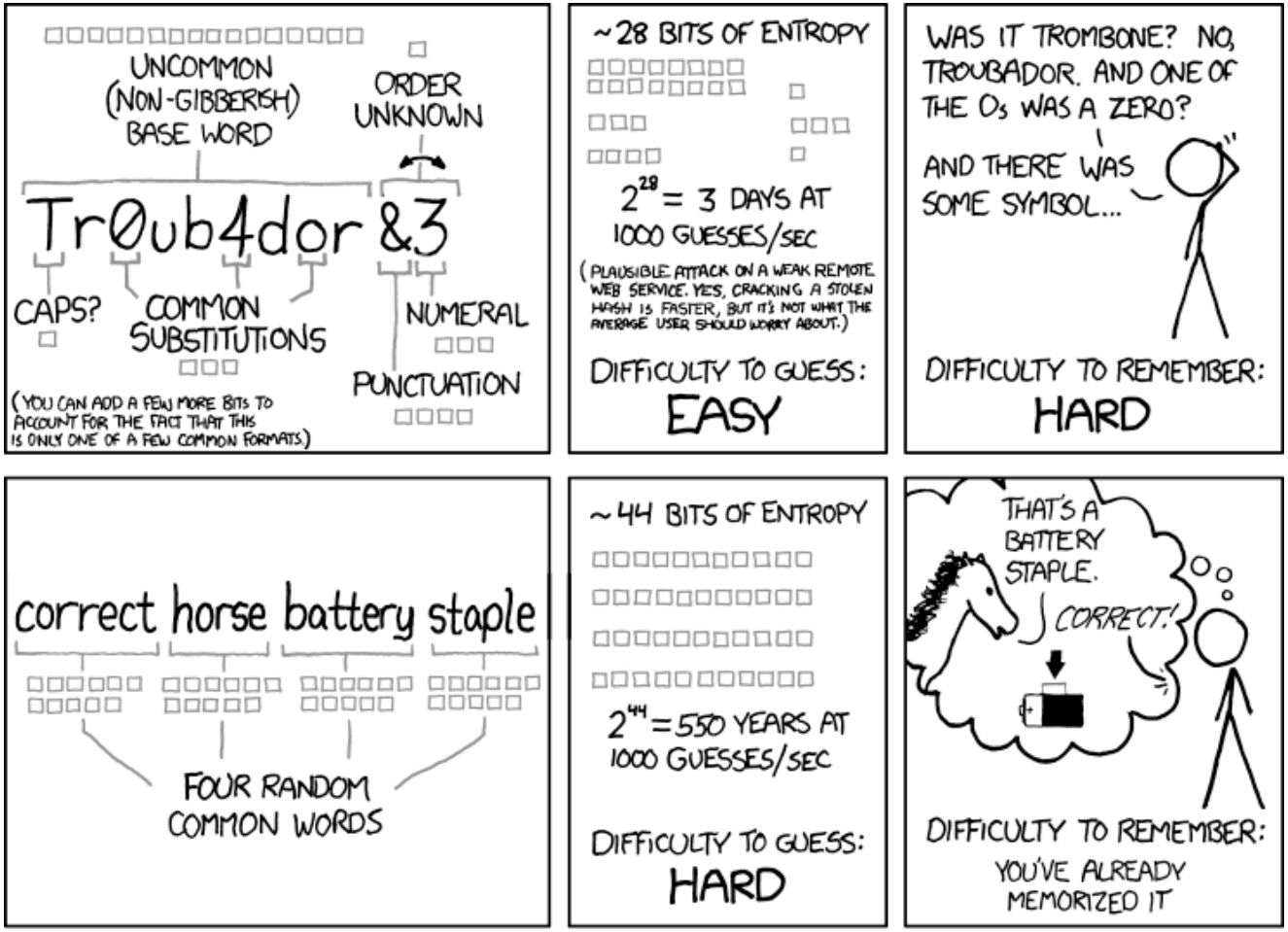
Passwords

- Cheapest way to authenticate, but 3 issues:
  - Will users enter passwords correctly?
  - Will they remember them, or will they choose weak ones or write them down?
  - Can they be tricked into revealing them?
- Advice is often like ‘choose something you can’t remember and don’t write it down’
- We know lots about password / PIN choice!

Can you train users?

- Experiment with first-year NatScis
  - Control group of 100 (+ 100 more observed)
  - Green group: use a memorable phrase
  - Yellow group: choose 8 chars at random
- Expected strength Y > G > C; got Y=G > C
- Expected resets Y > G > C; got Y=G=C
- But we had 10% noncompliance
- So if it matters, maybe measure entropy?

XKCD


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor&3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS:
EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER:
HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS:
HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER:
YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Password guessing

- Sometimes you can limit guessing
- E.g. bank card PINs – 3 guesses in the card and 3 online
- Enforced by hardware tamper-resistance and software in both card and bank server
- But: if the typical person has five cards with the same PIN, how many wallets do you need to find before you get lucky?

Password guessing (2)

- Bad guys sometimes get the password file anyway
- Salt: don't store $\{0\}_P$, but [Np, $\{Np\}_P$]
- Slow attacks further by multiple encryption
- Add breach reporting laws
- Externalise problem using Oauth protocol?
- So is authentication a natural cloud service? (after all, Google knows where you are)

Externalities

- One firm's action has side-effects for others
- Password sharing a conspicuous example; we have to enter credentials everywhere
- Everyone wants recovery questions too
- Many firms train customers in unsafe behaviour from clicking on external links to entering payment data in frames
- Much 'training' amounts to victim blaming

Incremental guessing

- Of Alexa top 500 websites, 26 use primary account number + exp date
- 37 use PAN + postcode (numeric digits only for some, add door number for others)
- 291 ask for PAN + expdate + CVV2
- Aamir Ali et al: iterated guessing works!
- Some paper receipts have PAN + expdate
- Some websites whitelist good customers

MAT HONAN GEAR 08.06.12 08:01 PM

HOW APPLE AND AMAZON SECURITY FLAWS LED TO MY EPIC HACKING

Mat Honan hack

- Get Mat's billing address from whois
- Call Amazon to add a credit card (then you see last 4 digits of others), then again to add email
- Apple password reset needs billing address plus last 4 digits of credit card
- Gmail password reset: sends a message to the backup email (Matt's apple @me.com account)
- Hackers wiped Matt's phone, Macbook and Gmail, then sent racist tweets from his Twitter

Security Protocols

- Security protocols are a second intellectual core of security engineering
- They are where cryptography and system mechanisms (such as access control) meet
- They introduce an important abstraction, and illustrate adversarial thinking
- They often implement policy directly
- And they are much older than computers…

Real-world protocol

- Ordering wine in a restaurant
  - Sommelier presents wine list to host
  - Host chooses wine; sommelier fetches it
  - Host samples wine; then it's served to guests
- Security properties?

Real-world protocol

- Ordering wine in a restaurant
  - Sommelier presents wine list to host
  - Host chooses wine; sommelier fetches it
  - Host samples wine; then it's served to guests
- Security properties
  - Confidentiality – of price from guests
  - Integrity – can't substitute a cheaper wine
  - Non-repudiation – host can't falsely complain

Car unlocking protocols

- Principals are the engine controller E and the car key transponder T
- Static ($T \rightarrow E$: KT)
- Non-interactive

  $T \rightarrow E$: T, $\{T,N\}_{KT}$
- Interactive

  $E \rightarrow T$: N

  $T \rightarrow E$: $\{T,N\}_{KT}$
- N is a ‘nonce’ for ‘number used once’. It can be a sequence number, a random number or a timestamp
- Can include a command, e.g. ‘lock’, ‘unlock’, ‘open boot’

Identify Friend or Foe (IFF)

- Basic idea: fighter challenges bomber

  $F \rightarrow B: N$

  $B \rightarrow F: \{N\}_K$

- What can go wrong?

Identify Friend or Foe (IFF)

- Basic idea: fighter challenges bomber

  $F \rightarrow B: N$

  $B \rightarrow F: \{N\}_K$

- What if the bomber reflects the challenge back at the fighter's wingman?

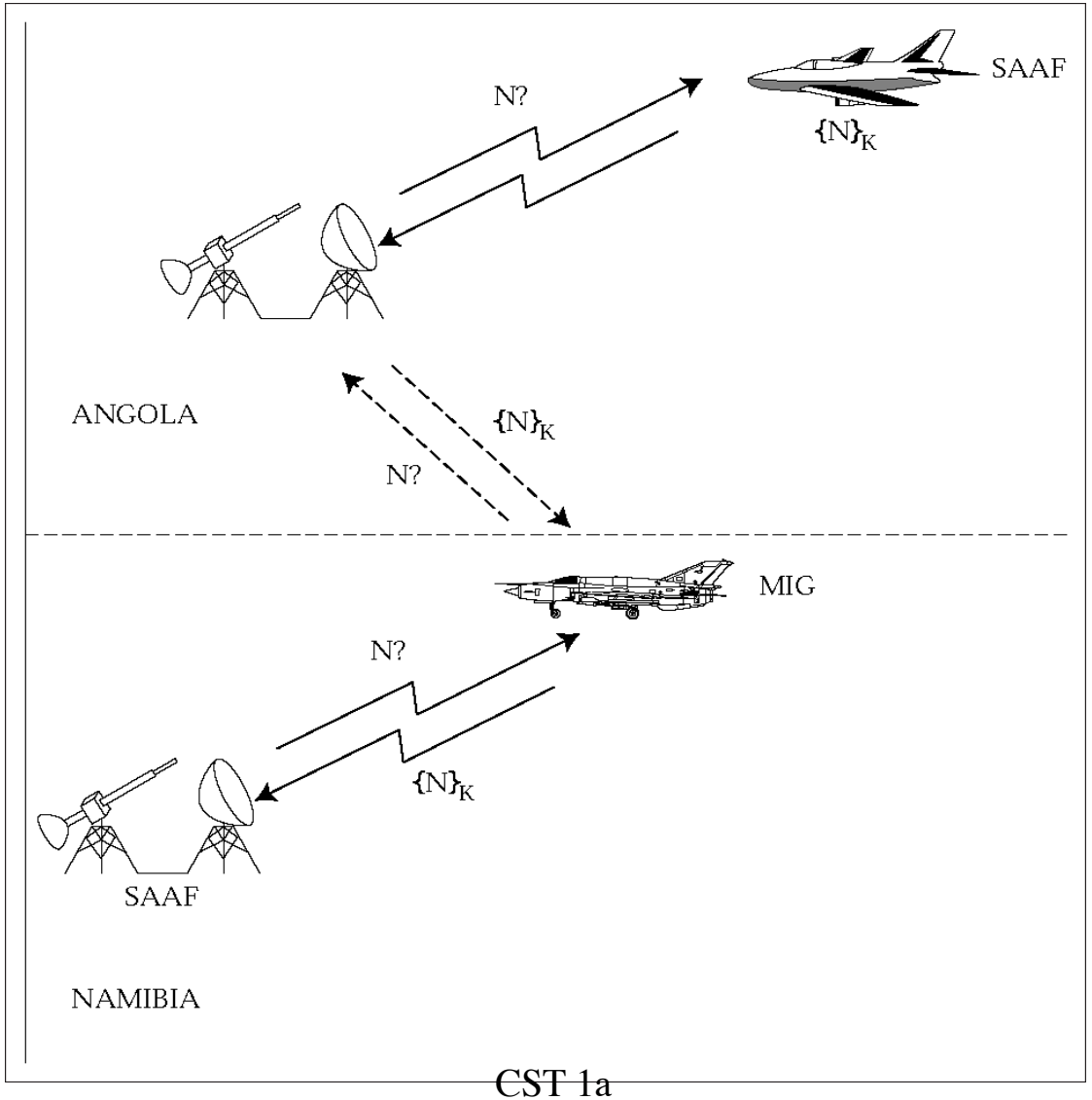
  $F \rightarrow B: N$

  $B \rightarrow F: N$

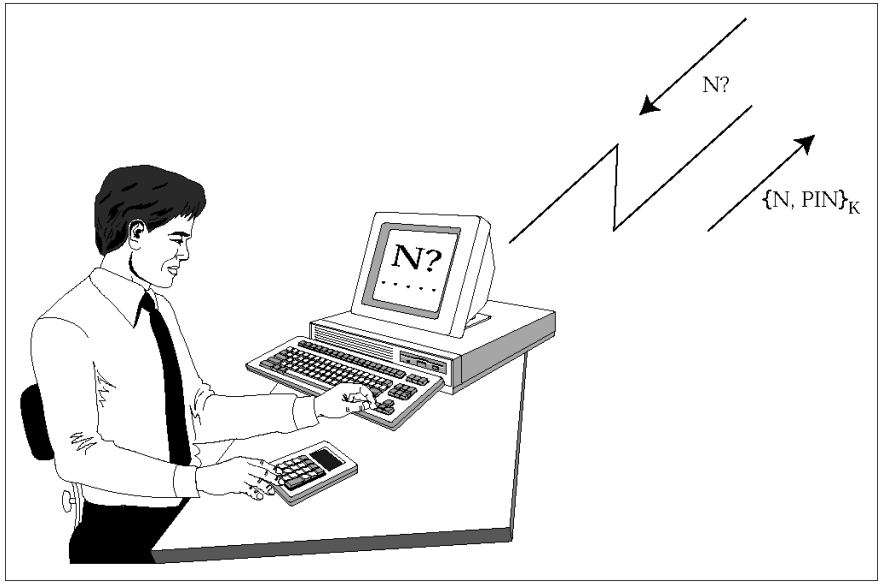
  $F \rightarrow B: \{N\}_K$

  $B \rightarrow F: \{N\}_K$

IFF (2)



Two-factor authentication



$S \rightarrow U: N$

$U \rightarrow P: N, PIN$

$P \rightarrow U: \{N, PIN\}_{KP}$

- How do you go about hacking this?

Card Authentication Protocol

- Lets banks use EMV cards in online banking
- Users compute codes for access, authorisation
- A good design would take PIN and challenge / data, encrypt to get response
- But the UK one first tells you if the PIN is correct
- What can go wrong with this?

Key management protocols

- Suppose Alice and Bob each share a key with Sam, and want to communicate?
  - Alice calls Sam and asks for a key for Bob
  - Sam sends Alice a key encrypted in a blob only she can read, and the same key also encrypted in another blob only Bob can read
  - Alice calls Bob and sends him the second blob
- How can they check the protocol's fresh?

Kerberos

- Uses ‘tickets’ based on encryption with timestamps to manage authentication in distributed systems (Windows, Linux, ...)

  $A \rightarrow S$: A, B

  $S \rightarrow A$: $\{T_S, L, KAB, B, \{T_S, L, KAB, A\}_{KBS}\}_{KAS}$

  $A \rightarrow B$: $\{T_S, L, KAB, A\}_{KBS}, \{A, T_A\}_{KAB}$

  $B \rightarrow A$: $\{T_A+1\}_{KAB}$

- Here S is the ticket-granting server giving access to the resource B

Europay-MasterCard-Visa (EMV)

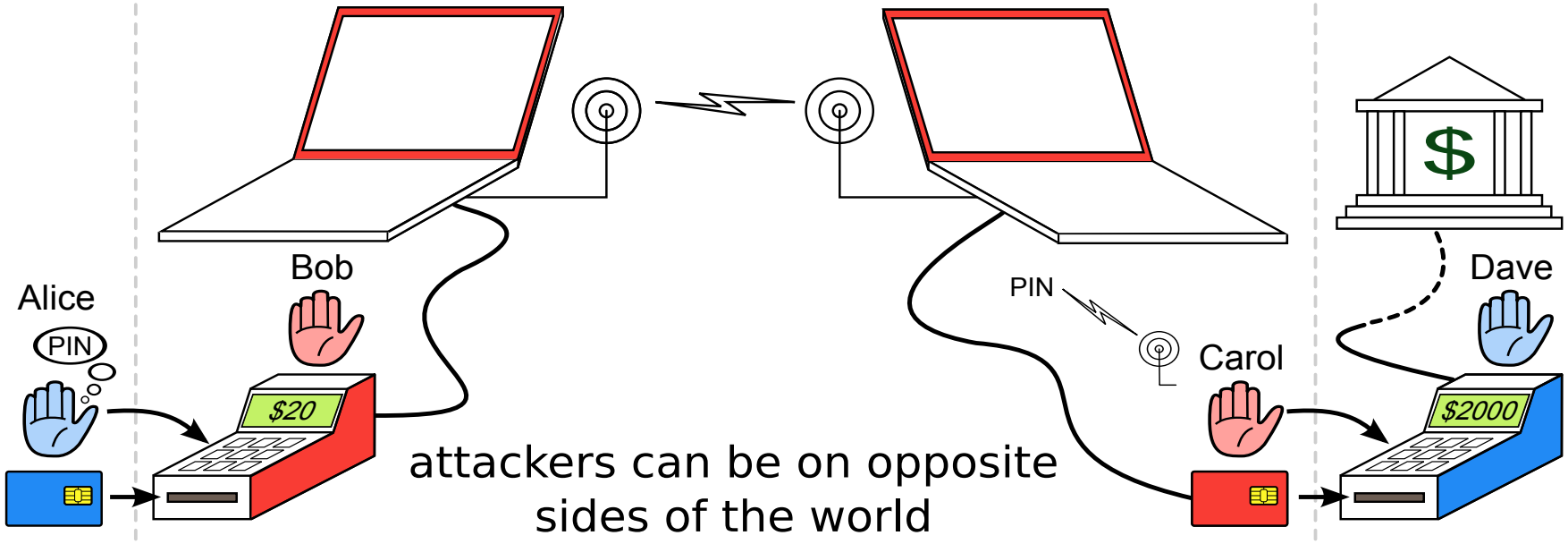
- $C \rightarrow M$: $sig_B\{C, card\_data\}$
- $M \rightarrow C$: N, date, Amt, PIN (if PIN used)
- $C \rightarrow M$: $\{N, date, Amt, trans\_data\}_{KCB}$
- $M \rightarrow B$: $\{\{N, date, Amt, trans\_data\}_{KCB}, trans\_data\}_{KMB}$
- $B \rightarrow M \rightarrow C$: $\{OK\}_{KCB}$

How might you attack this?

What about a false terminal?

- Replace a terminal's insides with your own electronics
- Capture cards and PINs from victims
- Use them to do a man-in-the-middle attack in real time on a remote terminal in a merchant selling expensive goods

The relay attack



Attacks in the real world

- The relay attack is almost unstoppable, but it was too hard to scale!
- What the bad guys did initially was mag-strip fallback fraud
- PEDs tampered at Shell garages by ‘service engineers’ (PED supplier went bust)
- BP Girton: 200+ customers found their cards cloned and used in Thailand, 2008

The No-PIN attack (2010)

- $C \rightarrow M$: $sig_B\{C, exp\}$
- $M \rightarrow \acute{C}$: N, date, Amt, PIN
- $\acute{C} \rightarrow C$: N, date, Amt
- $C \rightarrow M$: $\{N, date, Amt, trans\_data\}_{KCB}$
- $M \rightarrow B$: $\{\{N, date, Amt, trans\_data\}_{KCB}, trans\_data'\}_{KMB}$
- $B \rightarrow M$: $\{OK\}_{KCB}$

Fixing the 'No PIN' attack

- In theory: might compare card data with terminal data at terminal, acquirer, or issuer
- In practice: has to be the issuer (terminal and acquirer incentives are poor)
- Barclays introduced a fix July 2010; removed Dec 2010 (too many false positives?); banks asked for student thesis to be taken down from web instead
- Eventually fixed for UK transactions in 2016!
- Real problem: EMV spec now far too complex

The preplay attack (2014)

- In EMV, the terminal sends a random number N to the card along with the date d and the amount Amt
- The card authenticates N, d, X using the key it shares with the bank, KCB
- What happens if I can predict N for date d?
- Answer: given access to your card, I can precompute an authenticator for Amt, d!

Public key crypto revision

- You saw Diffie-Hellman in Discrete Maths
- Public key encryption lets you encrypt data using the public encryption key of some user, say Alice
- We'll write $\{X\}_A$ in our protocol notation
- She can decrypt it using her private decryption key
- Digital signatures are the other way round; only the holder of the private signature key can sign but anyone can verify

Public key crypto revision (2)

- Anthony sends a message in a box to Brutus
- But the messenger's loyal to Caesar, so Anthony puts a padlock on it
- Brutus adds his own padlock and sends it back to Anthony
- Anthony removes his padlock and sends it to Brutus who can now unlock it
- Is this secure?

Public key crypto revision (3)

- Naïve electronic version (doing arithmetic mod p):

  $A \rightarrow B$: M^{rA}

  $B \rightarrow A$: M^{rArB}

  $A \rightarrow B$: M^{rB}

- But encoding messages as group elements can be tiresome so instead Diffie-Hellman goes:

  $A \rightarrow B$: g^{rA}

  $B \rightarrow A$: g^{rB}

  $A \rightarrow B$: $\{M\}g^{rArB}$

Public key crypto revision (4)

- Developing Diffie-Hellman into El Gamal public key encryption: start with a generator g mod p
- Alice chooses her private key xA
- She publishes her public key $yA = g^{xA} \pmod p$
- Bob encrypts message M under yA by choosing a session key r and forming

  $$\{M\}yA = g^r, yA^r.M$$

- Alice decrypts by calculating $(g^r)^{xA} = yA^r$ and divides out to get M

Public-key Needham-Schroeder

- Proposed in 1978:

  $A \rightarrow B$: $\{NA, A\}_{KB}$

  $B \rightarrow A$: $\{NA, NB\}_{KA}$

  $A \rightarrow B$: $\{NB\}_{KB}$
- The idea is that they then use $NA \oplus NB$ as a shared key
- Is this OK?

Public-key Needham-Schroeder (2)

- Attack found eighteen years later, in 1996:

  $A \rightarrow C$: $\{NA, A\}_{KC}$

  $C \rightarrow B$: $\{NA, A\}_{KB}$

  $B \rightarrow C$: $\{NA, NB\}_{KA}$

  $C \rightarrow A$: $\{NA, NB\}_{KA}$

  $A \rightarrow C$: $\{NB\}_{KC}$

  $C \rightarrow B$: $\{NB\}_{KB}$

- Fix: explicitness. Put all names in all messages

Public key certification

- One way of linking public keys to principals is to physically install them on machines (IPSEC, SSH)
- Another is trust on first use: set up keys, then verify manually that you're speaking to the right principal (Signal, Bluetooth simple pairing)
- Another is certificates. Sam signs Alice's public key (and/or signature verification key)

  $CA = sig_S\{T_S,L,A,K_A,V_A\}$
- This is the basis of SSL / TLS

Transport Layer Security (TLS)

- Customer C calls server S

  $C \rightarrow S$: C, C#, NC

  $S \rightarrow C$: S, S#, NS, CS

  $C \rightarrow S$: $\{K0\}_S$

  $C \rightarrow S$: crypto hash of K0, NC, NS, etc

  $S \rightarrow C$: crypto hash of K0, NS, NC, etc

- This has been proved to be secure (Larry Paulson, 1999)
- So what could possibly go wrong?

What goes wrong

- Although abstract TLS is proven secure, real implementations break about annually
- Attacks: send bad packets and observe error messages, or measure the time it takes to encrypt, or scavenge memory ...
- Writing crypto code is hard (the compiler tries to optimise away your defensive code)
- Protocol extension, composition break stuff
- Later courses have many more details

What goes wrong (2)

- Governments may demand weak ciphers, or attack or coerce the certification authority
- See if you can find the Turkish government cert in your browser...
- More: read Snowden, Diginotar, certificate pinning, ‘Keys under doormats’
- For critical stuff (your startup’s software update key), do you need your own CA?

What goes wrong (3)

- 'Leverage' – sharing infrastructure – can be attractive but is often a snare
- Suppose that we had a protocol for users to sign hashes of payment messages :

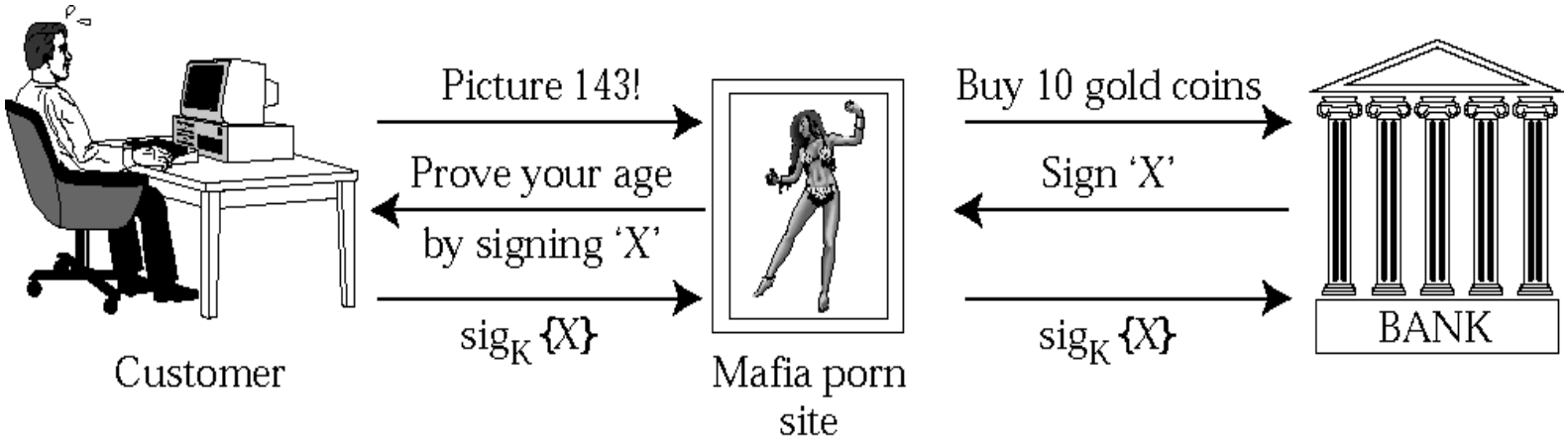
  $C \rightarrow M$: order

  $M \rightarrow C$: X [= hash(order, amount, date, …)]

  $C \rightarrow M$: $sig_K\{X\}$
- How might this be attacked?

‘Chosen protocol attack’

The Mafia asks people to sign a random challenge as proof of age for porn sites!



Entomology

- What sort of bugs can we expect?
- Bugs in the code
  - Arithmetic
  - Syntactic
  - Logic
- Bugs around the code
  - Code injection
  - Usability traps (for programmers)

Arithmetic bug – Patriot missile

- Failed to intercept an Iraqi SCUD missile in Gulf War 1 on Feb 25 1991; SCUD struck US barracks in Dhahran; 28 dead
- Other SCUDs hit Saudi Arabia, Israel

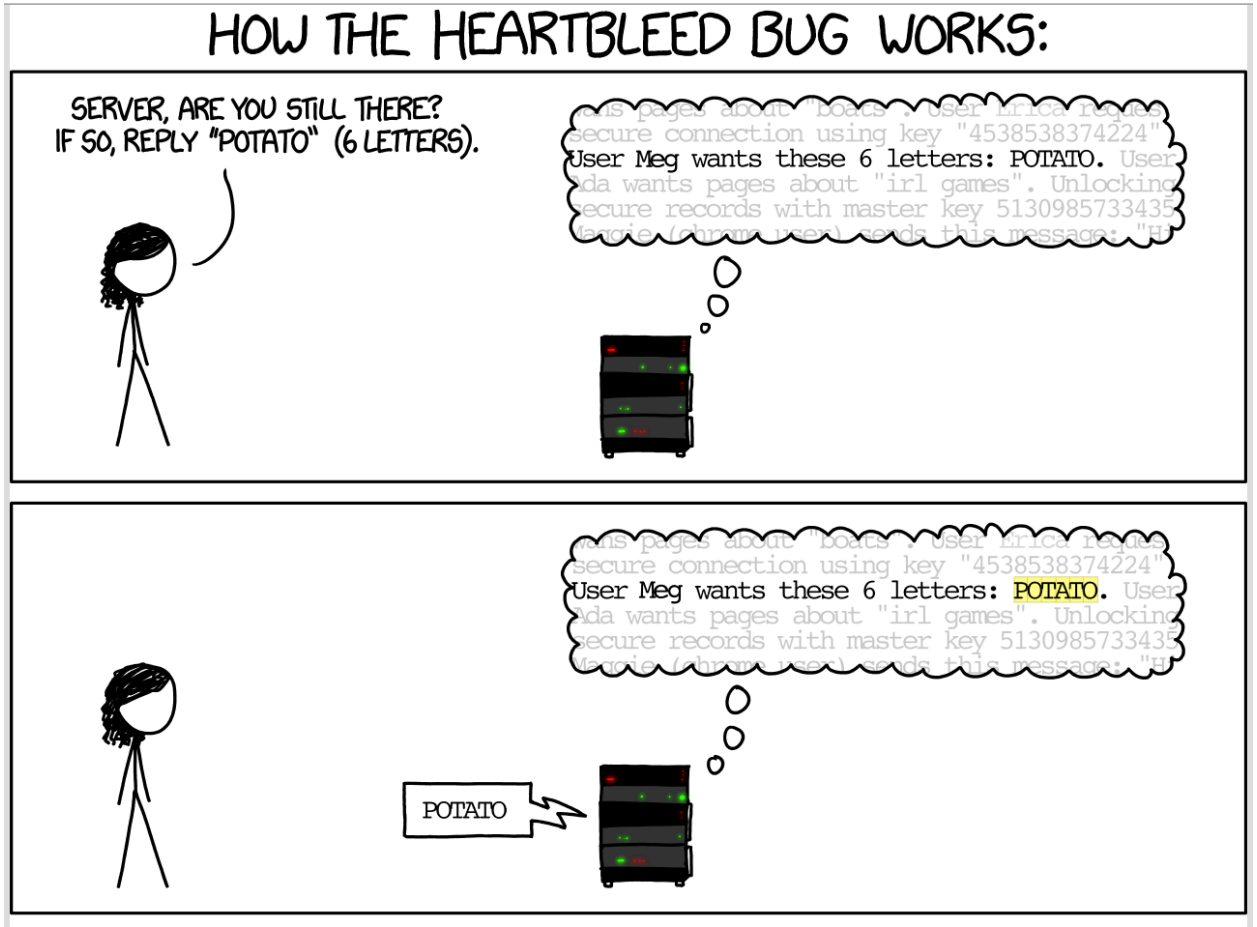
Patriot missile (2)

- It was a bug in the arithmetic
  - measured time in 1/10 sec, truncated from .0001100110011…
  - when system upgraded from air-defence to anti-ballistic-missile, accuracy increased
  - but not everywhere in the (assembly language) code!
  - modules got out of step by 1/3 sec after 100h operation
  - not found in testing as spec only called for 4h tests
- Critical system failures are typically multifactorial
- Still, years later, the Boeing 787 must be rebooted every 51h or it becomes unsafe!

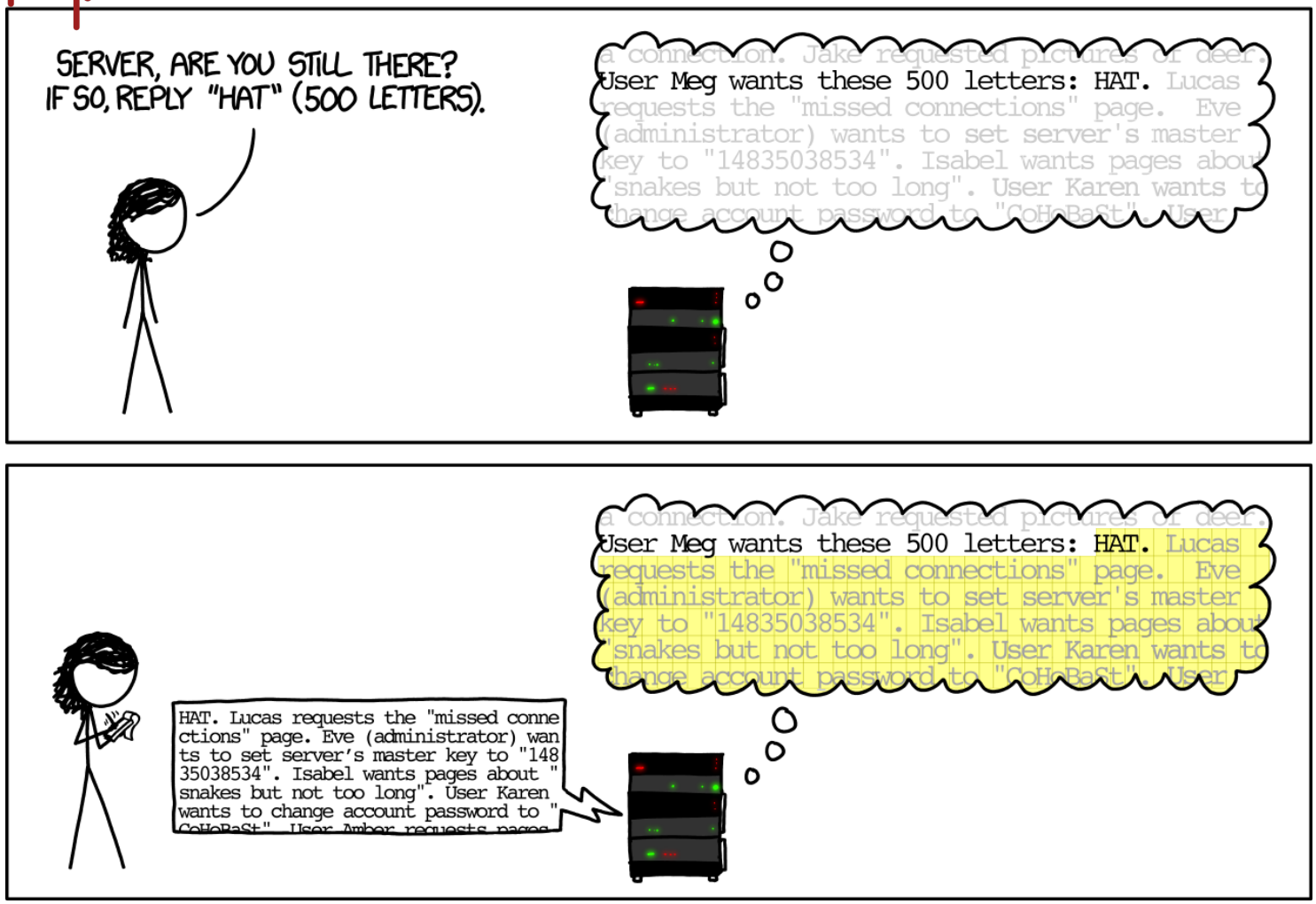
Syntactic bugs

- By this we mean bugs that arise from the features of a specific language.
- In java
  - `1+2+""="3"`
  - `""+1+2="12"`
- Can anyone explain the following?
  - perl -e 'printf("%d\n", "information" == "")'
  - perl -e 'printf("%d\n", "automation" == "")'

Heartbleed, by XKCD



Heartbleed (2)



Logic bugs

- In April 2014, the Heartbleed bug forced rapid reissue of most TLS certificates
- Missing bounds check in the OpenSSL code for the heartbeat TLS extension
- A *buffer over-read* can leak the private key, as well as user data, passwords, cookies etc
- White House tussle on NSA ‘equity issue’: they had exploited the bug for 2 years

Notification / clean-up

12th March 2012	Bug introduced (OpenSSL 1.0.1)
1st April 2014	Google secretly reports vuln
3rd April 2014	Codenomicon reports vuln
7th April 2014	Fix release, public announcement
9th May 2014	57% of website still using old TLS certificates
20th May 2014	1.5% of 800,000 most popular websites still vulnerable

Intel AMT Bug

- AMT allows sysadmins remote access to a machine, even when switched off (so long as mains power still on)
- Provides full access to machine, regardless of OS
- A sketch of the authentication protocol between machine and remote party is as follows:

  C $\rightarrow$ S: "Hi. I'd like to connect"

  S $\rightarrow$ C: "Please encrypt *X* with our secret key"

  C $\rightarrow$ S: "Here are the first *x* bytes of $\{X\}_{KCS}$"
- It also worked for $x = 0$

Concurrency bugs

- Recall the preplay attack on EMV?
- A generic security failure is “time of check to time of use” flaw (TOCTTOU)
- Race conditions: See Therac-25 case, later
- Another issue is synchronisation. See “The bug heard round the world”: the first Shuttle launch aborted when they couldn’t sync the five guidance computers (more on redundancy later)

Analogue code injection

- Clallam Bay jail had inmate payphones
- Inmate dials number to which recorded voice says: “If you will accept a collect call, please press the number 3 on your handset twice. The caller will now say his name”
- This can be sent in English or Spanish

Analogue code injection

- Clallam Bay jail had inmate payphones
- Inmate dials number to which recorded voice says: “If you will accept a collect call, please press the number 3 on your handset twice. The caller will now say his name”
- Hack: select Spanish then speak your name as “To hear this message in English, please type 33.”

Code injection

- Is it ethical for Burger King to run an ad that says “OK Google, what is the Whopper Burger?”

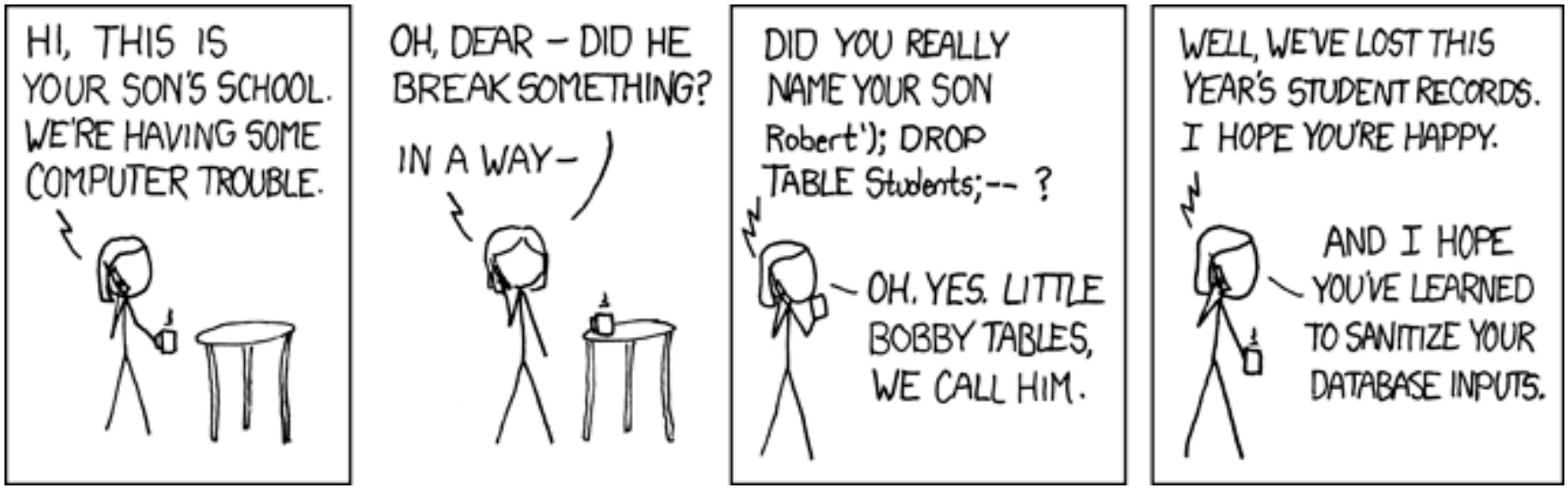
Code injection

- Is it ethical for Burger King to run an ad that says “OK Google, what is the Whopper Burger?”
- Their ad people had changed the wikipedia page; it was then defaced, then locked down
- Google then blacklisted that specific phrase
- (Back in the 80s – demo of ‘FORMAT C:’)

Buffer overflows

- In 1988, the Morris worm brought down the Internet by spreading rapidly in Unix boxes
- It had a list of passwords to guess, but also used three buffer overflow attacks
- These used a remote command (finger, rsh) with a long argument that overran the stack
- The extra bytes were interpreted as code
- Full details later in 1b Security course

SQL injection



- $sql = "INSERT INTO Students (Name) VALUES ('" . $studentName . "');"; execute_sql($sql);
- So, “sanitize all inputs” or ”don’t create SQL statements that include outside data”?

Software countermeasures

- Operating system
  - Address space layout randomisation
  - Data execution prevention
- Tool choice
  - Strongly typed languages
- Defensive programming
  - 1949: EDSAC coders check arithmetic
  - Now: assertions

Software countermeasures (2)

- Secure coding standards
  - MS standards for C, Bjarne Stroustrup for C++
  - Google: set libraries of user-facing code
  - Much else
- Contracts (in the Eiffel language)
- API analysis (can less trusted code that calls your libraries manipulate them?)
- Fuzzing, Coverity and other analysis tools

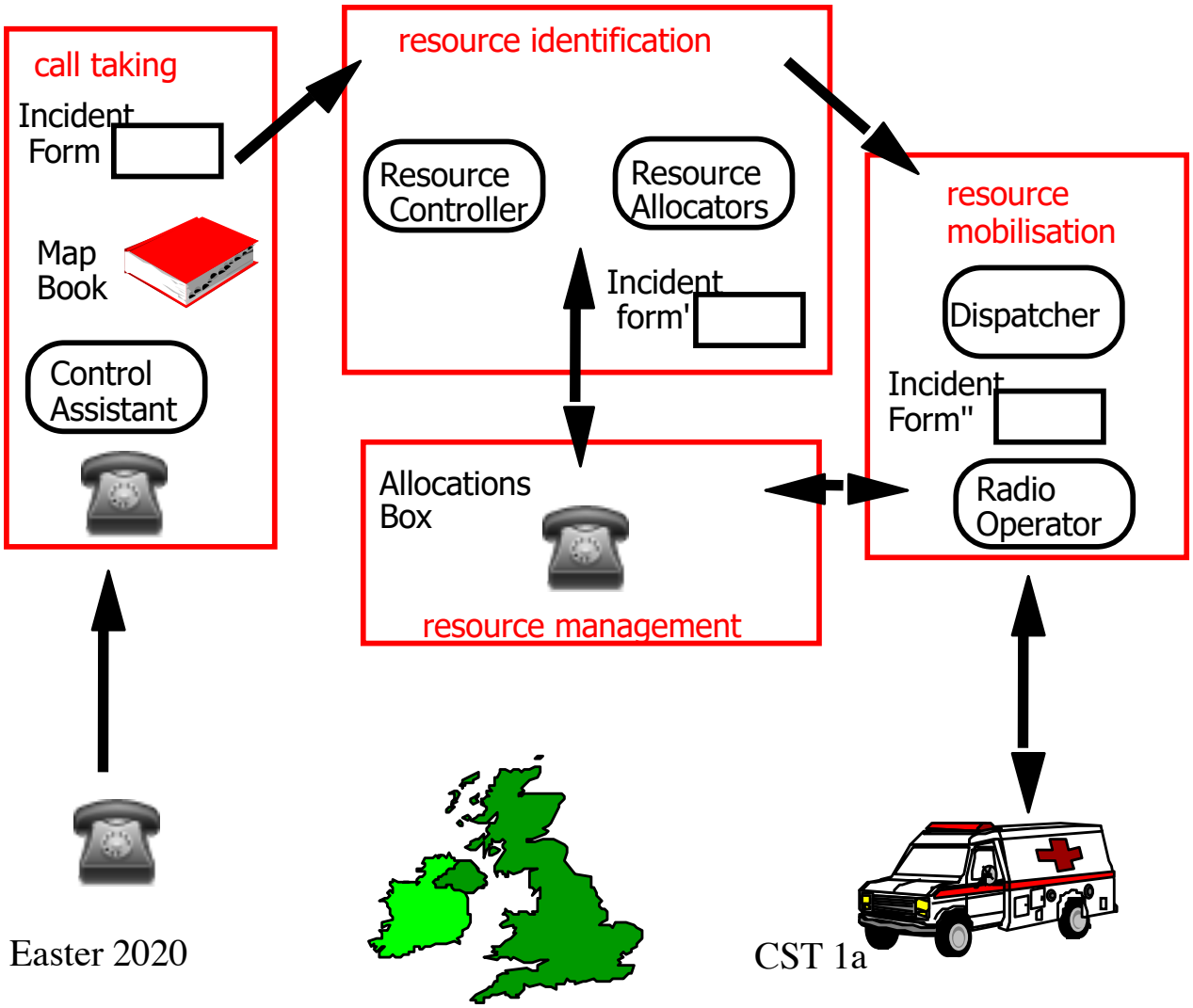
The ‘Software Crisis’

- Big software projects – and the maintenance of big systems – are really hard!
- Cost and risk scale nonlinearly with size and complexity
- Many large projects are late, over budget, or don’t work well or at all (NPfIT, DWP…)
- Some cost billions (Ariane 5, NPfIT)
- Others cost lives (Therac 25, Boeing 737)
- Some combine the above (LAS)

The London Ambulance Service disaster

- Attempt to automate ambulance dispatch in 1992 failed conspicuously with London being left without service for a day
- Hard to say how many deaths could have been avoided; estimates ran as high as 20
- Led to CEO being sacked, public outrage
- Widely cited example of project failure because it was thoroughly documented (and the pattern's been repeated again and again)

The manual implementation



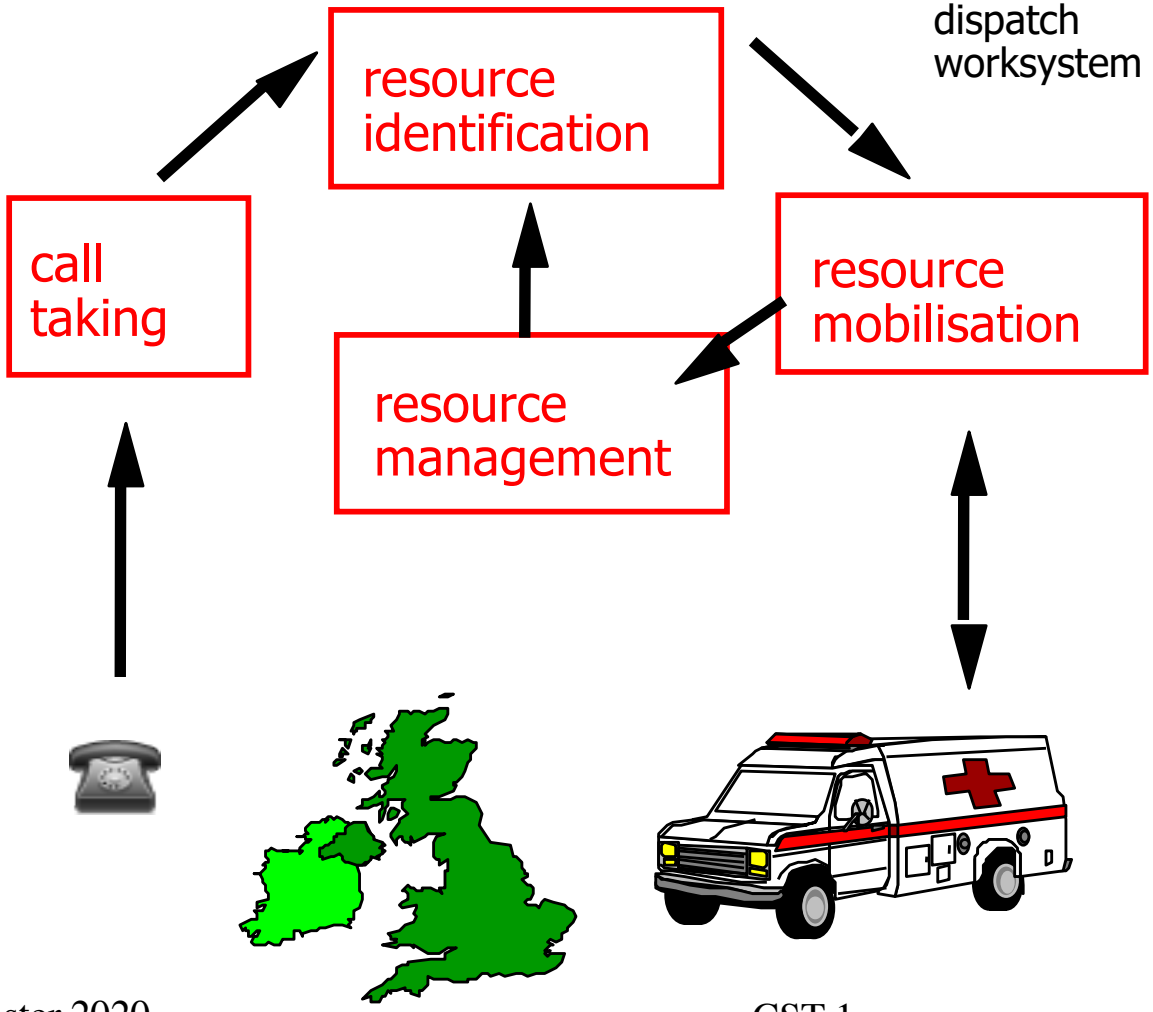
Original dispatch system

- 999 calls written on paper tickets; map reference looked up; conveyor to central point
- Controller deduplicates tickets and passes to three divisions – NW / NE / S
- Division controller identifies vehicle and puts note in its activation box
- Ticket passed to radio controller
- This all takes about 3 minutes and 200 staff of 2700 total. Some errors (esp. deduplication), some queues (esp. radio), call-backs tiresome

Project context

- Attempt to automate in 1980s failed – system failed load test
- Industrial relations poor – pressure to cut costs
- Public concern over service quality
- SW Thames RHA decided on fully automated system: responder would email ambulance
- Consultancy study said this might cost £1.9m and take 19 months – provided a packaged solution could be found. AVLS would be extra

Computer-aided dispatch system



- Large
- Real-time
- Critical
- Data rich
- Embedded
- Distributed
- Mobile components

Tender process

- Idea of a £1.5m system stuck; idea of AVLS added; proviso of a packaged solution forgotten; new IS director hired
- Tender 7/2/1991 with completion deadline 1/92
- 35 firms looked at tender; 19 proposed; most said timescale unrealistic, only partial automation possible by 2/92
- Tender awarded to consortium of Systems Options Ltd, Apricot and Datatrak for £937,463 – £700K cheaper than next lowest bidder!

First phase

- Design work ‘done’ July
- Main contract signed in August
- LAS told in December that only partial automation by January deadline – front end for call taking, gazetteer, docket printing
- Progress meeting in June had already minuted a 6 month timescale for an 18 month project, a lack of methodology, no full-time LAS user, and SO’s reliance on ‘cozy assurances’ from subcontractors

From phase 1 to phase 2

- Server never stable in 1992; client and server lockup
- Phase 2: radio messaging with blackspots and congestion. Couldn't cope with 'established working practices'
- Yet management decided to go live 26/10/92
- CEO: "No evidence to suggest that the full system software, when commissioned, will not prove reliable"
- Independent review had called for volume testing, implementation strategy, change control … It was ignored!
- On 26 Oct, the room was reconfigured to use terminals, not paper. There was no backup…

LAS disaster

- Vicious circle on 26/7 October:
  - system progressively lost track of vehicles
  - exception messages scrolled up off screen and were lost
  - incidents held as allocators searched for vehicles
  - callbacks from patients increased causing congestion
  - data delays → voice congestion → crew frustration → pressing wrong buttons and taking wrong vehicles → many vehicles sent to an incident, or none
  - slowdown and congestion leading to collapse
- Switch back to semi-manual operation on 26th and to full manual on Nov 2 after crash

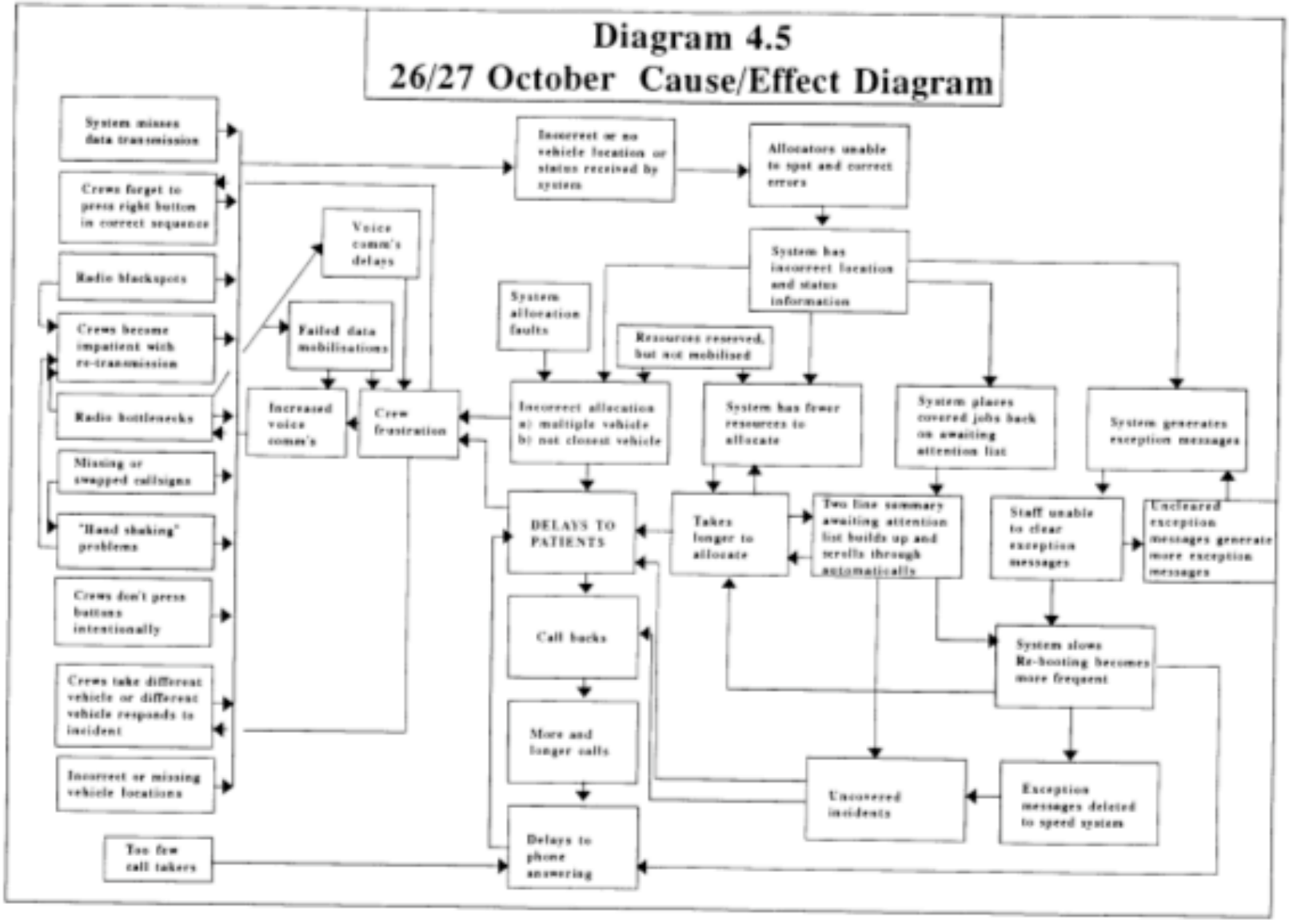


Diagram 4.5
26/27 October Cause/Effect Diagram
System misses data transmission
Crews forget to press right button in correct sequence
Radio blackspots
Crews become impatient with re-transmission
Radio bottlenecks
Missing or swapped callsigns
"Hand shaking" problems
Crews don't press buttons intentionally
Crews take different vehicle or different vehicle responds to incident
Incorrect or missing vehicle locations
Too few call takers
Voice comm's delays
Failed data mobilisations
Increased voice comm's
Crew frustration
Incorrect or no vehicle location or status received by system
Allocators unable to spot and correct errors
System has incorrect location and status information
System allocation faults
Resources reserved, but not mobilised
Incorrect allocation a) multiple vehicle b) not closest vehicle
System has fewer resources to allocate
System places covered jobs back on awaiting attention list
System generates exception messages
DELAYS TO PATIENTS
Takes longer to allocate
Two line summary awaiting attention list builds up and scrolls through automatically
Staff unable to clear exception messages
Uncleared exception messages generate more exception messages
Call backs
More and longer calls
Delays to phone answering
System slows Re-booting becomes more frequent
Uncovered incidents
Exception messages deleted to speed system

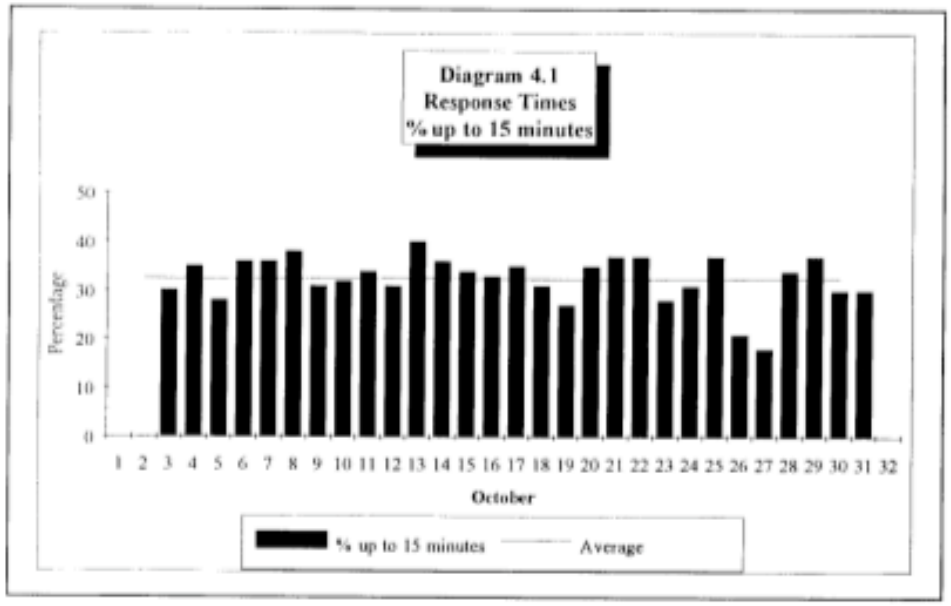


Diagram 4.1
Response Times
% up to 15 minutes
Percentage
0
10
20
30
40
50
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
October
% up to 15 minutes
Average

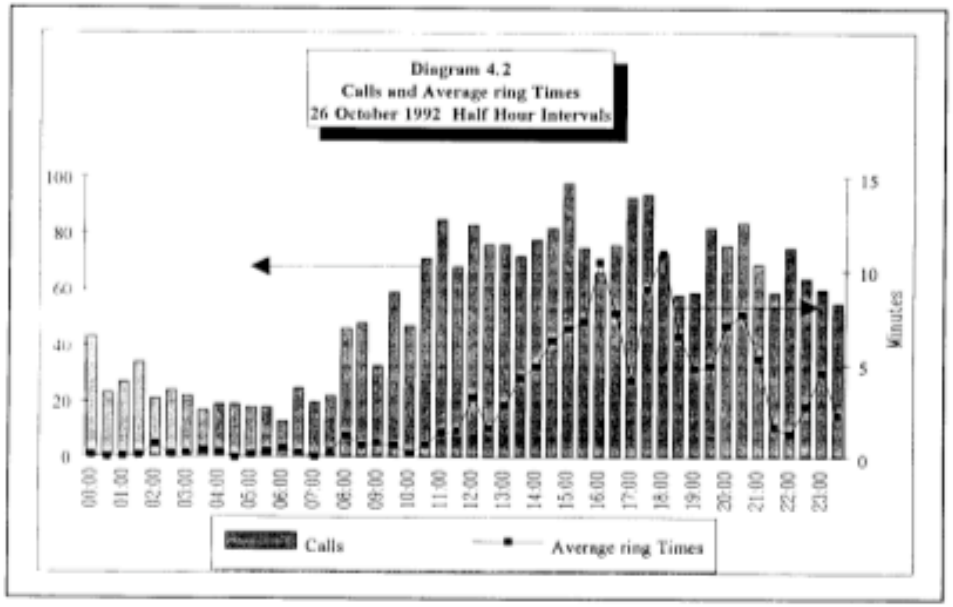


Diagram 4.2
Calls and Average ring Times
26 October 1992 Half Hour Intervals
100
80
60
40
20
0
15
10
5
0
Minutes
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
Calls
Average ring Times

Collapse

- Entire system descended into chaos:
  - e.g., one ambulance arrived to find the patient dead and taken away by undertakers
  - e.g., another answered a ‘stroke’ call after 11 hours, 5 hours after the patient had made their own way to hospital
- People probably died as a result
- Chief executive resigns

What went wrong – specification

- LAS ignored advice on cost and timescale
- Procurers insufficiently qualified and experienced
- No systems view
- Specification was inflexible but incomplete: it was drawn up without adequate consultation with staff
- Attempt to change organisation through technical system
- Ignored established work practices and staff skills

What went wrong – project

- Confusion over who was managing it all
- Poor change control, no independent QA, suppliers misled on progress
- Inadequate software development tools
- Ditto datacomms, with effects not foreseen
- Poor interface for ambulance crews
- Poor control room interface

What went wrong – go-live

- System went live with known serious faults
  - slow response times
  - workstation lockup
  - loss of voice comms
- Software not tested under realistic loads or as an integrated system
- Inadequate staff training
- No back up, short of full manual operation!

LAS as a case study

- Maybe a third of all big projects go wrong
- You'll work on some for sure!
- They're usually hushed up
- The London Ambulance Service disaster could not be, so we have a full report
- Read it!
- And read lots of other case studies too

NHS National Programme for IT

- Like LAS, an attempt to centralise power and change working practices
- Earlier failed attempt in the 1990s
- The February 2002 Blair meeting
- Five LSPs plus national contracts: £12bn
- Most systems years late and/or didn't work
- Coalition government: NPfIT 'abolished'
- See case history written by MPP students in 2014 (linked from course materials page)

Next – ‘smart meters’

- Idea: expose consumers to market prices, get peak demand shaving, make use salient
- EU Electricity Directive 2009: 80% by 2020
- Labour 2009: £10bn centralised project to save the planet and help fix supply crunch in 2017
- 2010: became part of the coalition agreement
- Escaped controls as ‘not an IT project’
- Government: wanted deployment by 2015 election! Then: 2017 manifesto. Then …
- Incentives wrong, tech getting obsolete…
- Failure of Ontario project to save any energy

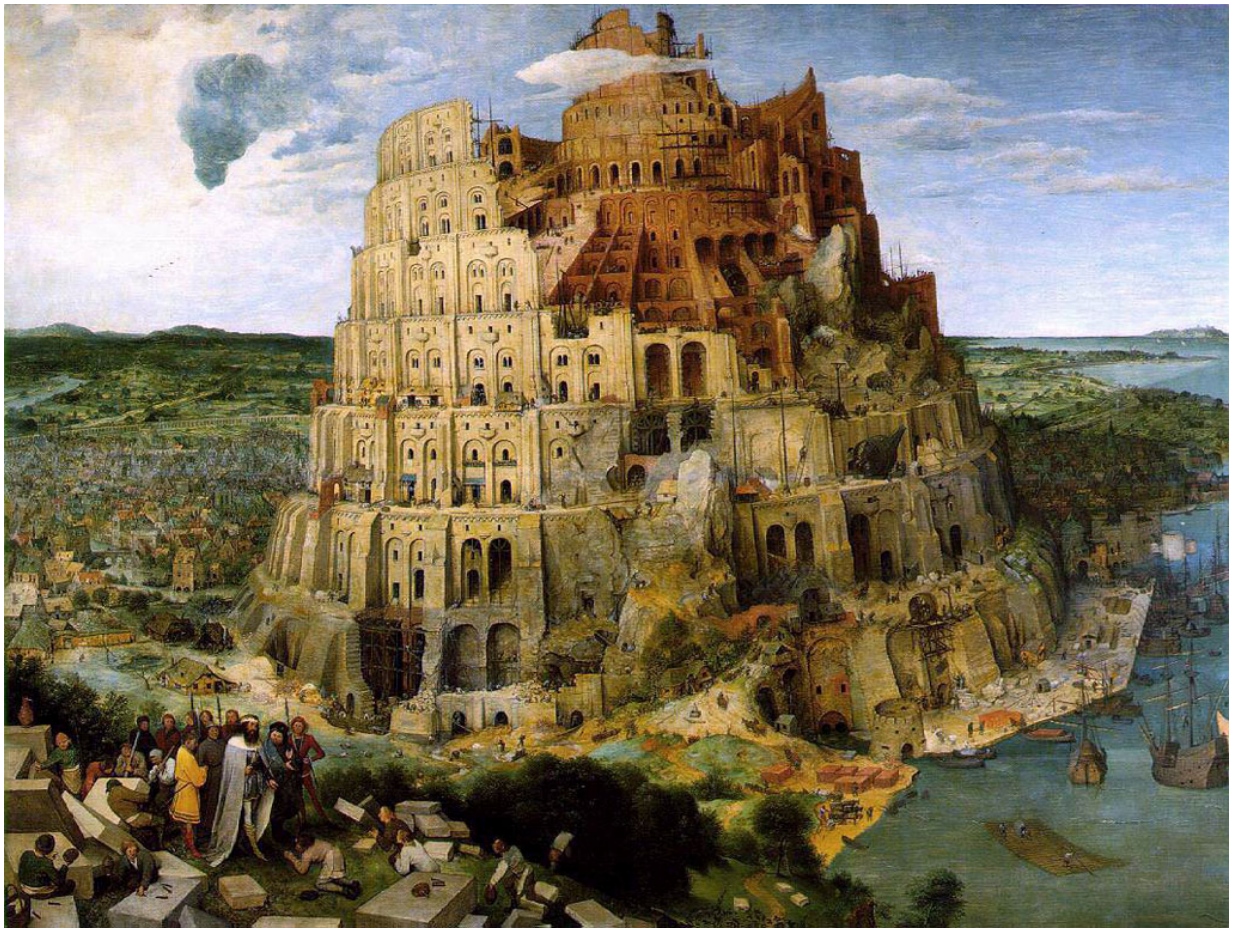
Next – Universal Credit

- Idea: unify hundreds of welfare benefits and mitigate poverty trap by tapered withdrawal as claimants start to earn
- Supposed to go live Oct 2013! But many dependencies, such as on real-time tax data…
- Started migrating over the simple cases but welfare is complex so progress is slow
- Multiple reports from the National Audit Office, the latest (2018) saying ‘no practical alternative’
- Now: really serious stress test with pandemic

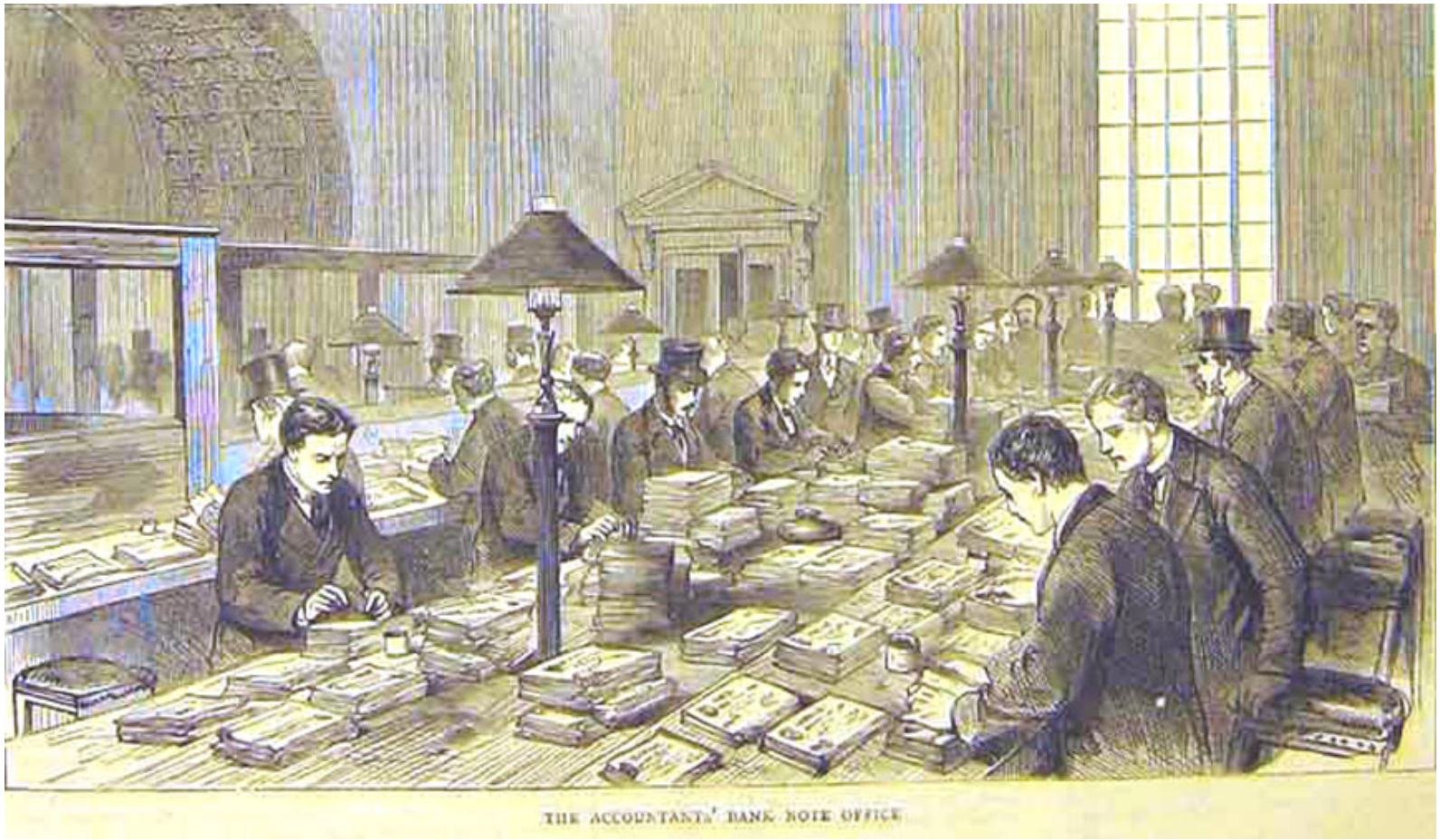
Managing complexity

- Software engineering is about managing complexity at a number of levels
  - At the micro level, bugs arise in protocols etc because they're hard to understand
  - As programs get bigger, interactions between components grow at $O(n^2)$ or even $O(2^n)$
  - Systems are built of ever more components
  - With complex socio-technical systems, we can't predict reactions to new functionality
- Most failures of really large projects are down to wrong, changing, or contested requirements

Project failure, c. 1500 BC



Complexity, 1870 – Bank of England



Complexity 1876 – Dun, Barlow & Co

Nineteenth century view

- Charles Babbage, 'On Contriving Machinery'
  - "It can never be too strongly impressed upon the minds of those who are devising new machines, that to make the most perfect drawings of every part tends essentially both to the success of the trial, and to economy in arriving at the result"

Complexity 1906 – Sears, Roebuck

- Continental-scale mail order meant specialization
- Big departments for single bookkeeping functions
- Beginnings of automation

Complexity 1940 – First National Bank of Chicago

1960s – the ‘software crisis’

- In the 1960s, large powerful mainframes made even more complex systems possible
- People started asking why project overruns and failures were so much more common than in mechanical engineering, shipbuilding…
- ‘Software engineering’ was coined in 1968
- The hope was that we could things under control by using disciplines such as project planning, documentation and testing

How is software different?

- Many things that make writing software fun also make it complex and error-prone:
  - joy of solving puzzles and building things from interlocking moving parts
  - stimulation of a creative task with continuous learning
  - pleasure of working with a tractable medium, ‘pure thought stuff’
  - satisfaction of making stuff that’s useful to others
  - you can improve the world by making the output depend on the inputs in any novel way you can imagine

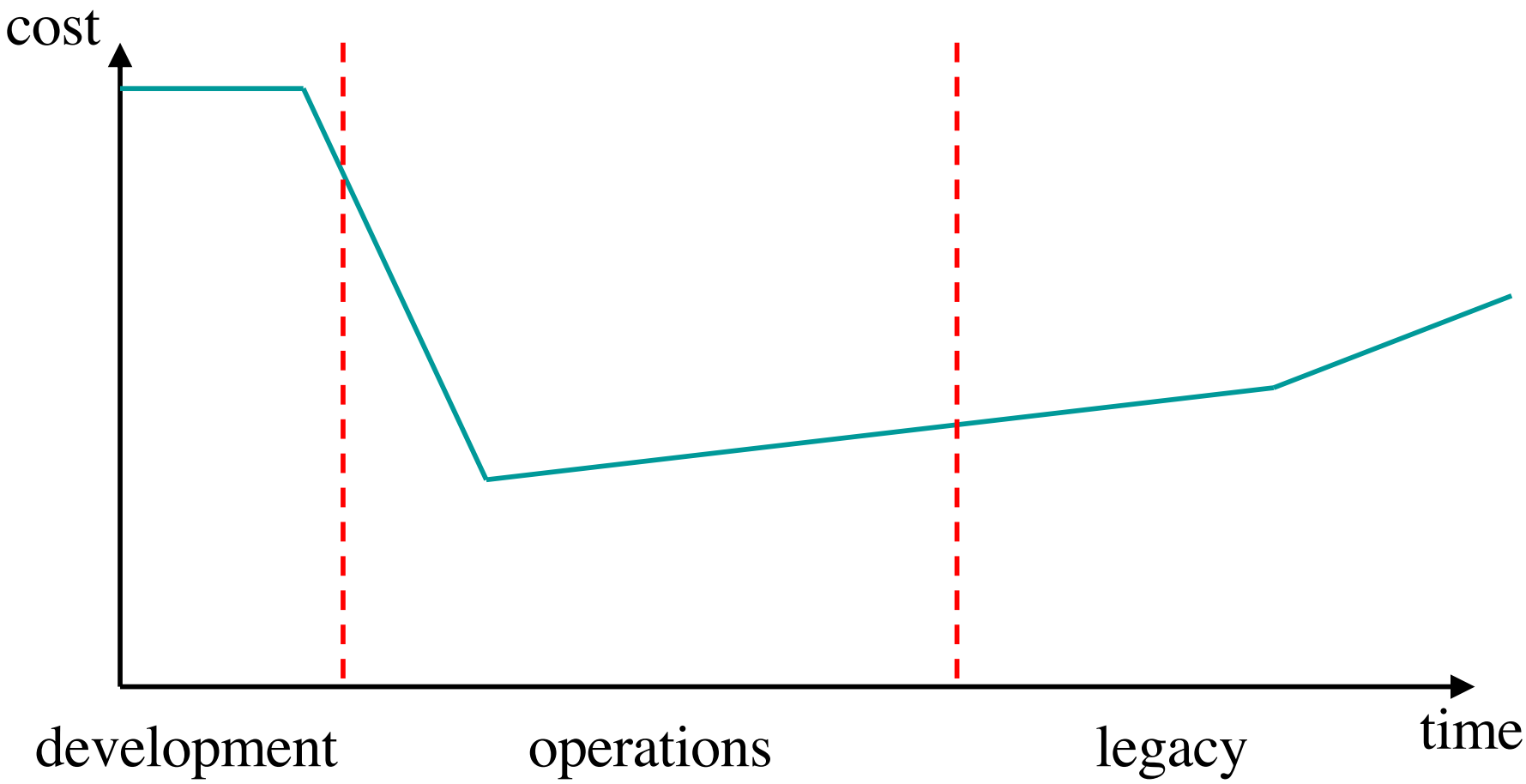
How is software different? (2)

- Large systems become qualitatively more complex, unlike big ships or long bridges
- The tractability of software leads customers to demand ‘flexibility’ and frequent changes
- This makes systems more complex to use over time as ‘features’ accumulate, and interactions have odd effects
- The structure can be hard to visualise or model
- The hard slog of debugging and testing piles up at the end, when the excitement’s past, the budget’s spent and the deadline’s looming

The software life cycle

- Software economics can be very messy
  - Consumers buy on sticker price, businesses on total cost of ownership
  - vendors try to lock customers in, so bargains are followed by ripoffs
- But let's consider the simplest case, of a company that develops and maintains software entirely for its own use

Cost of software: development 10%, maintenance 90%



How can you measure code cost?

- First IBM measures (60s)
  - 1.5 KLOC/person year (operating system)
  - 5 KLOC/person year (compiler)
  - 10 KLOC/person year (app)
- AT&T measures
  - 0.6 KLOC/person year (compiler)
  - 2.2 KLOC/person year (switch)
- Alternatives
  - Halstead (entropy of operators/operands)
  - McCabe (graph entropy of control structures)
  - Function point analysis

First-generation lessons learned

- There are huge variations in productivity between individuals
- The main systematic gains come from using an appropriate high-level language
- High level languages take away much of the accidental complexity, so the programmer can focus on the intrinsic complexity
- Extra effort getting the specification right usually pays for itself by reducing the time spent coding and testing

Development costs

- Barry Boehm, 1975

	Spec	Code	Test
C3I	46%	20%	34%
Space	34%	20%	46%
Scientific	44%	26%	30%
Business	44%	28%	28%

- So – the toolsmith should not focus just on code!

‘The Mythical Man-Month’

- Fred Brooks debunked interchangeability
- Imagine a project at 3 developers by 4 months
  - Suppose the design work takes an extra month. So we have 2 months to do 9 person-months work
  - If training someone takes a month, we must add 6 devs
  - But the work 3 devs did in 3 months can’t be done by 9 devs in one! Interaction costs maybe $O(n^2)$
- Hence Brooks’ law: adding manpower to a late project makes it later!

Software project economics

- Barry Boehm data, 1981: Project duration in person-months

$$PM = A.\ KLOC^{B}$$

  - A is code type, B expresses diseconomy of scale
- Cost-optimal time to first shipment

$$T = 2.5.\ (PM)^{1/3}$$

  - With more time, cost rises slowly
  - With less time, it rises sharply
- Yet some projects fail despite huge resources!

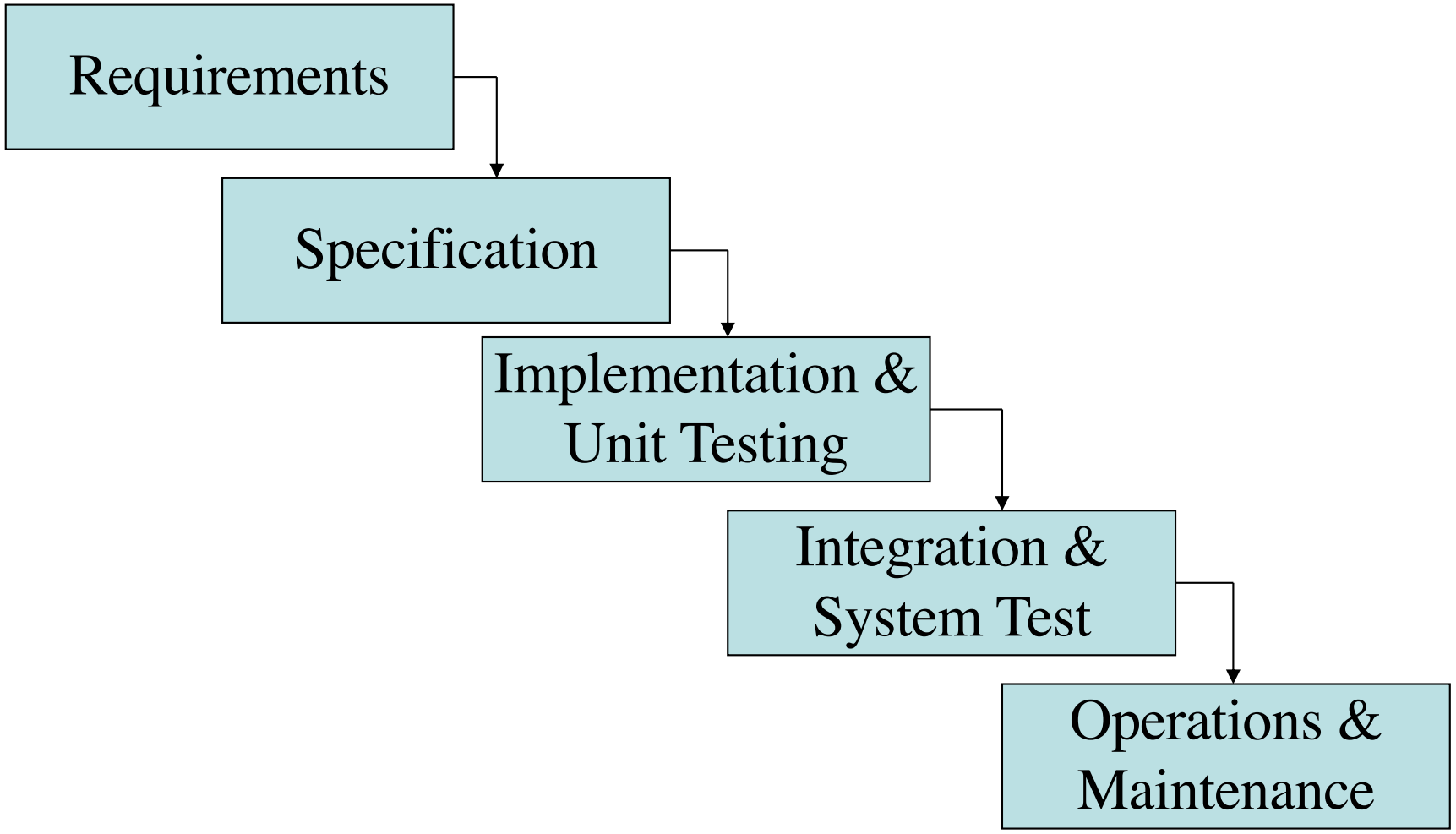
The software project ‘Tar Pit’

- You can pull any one of your legs out of the tar …
- Individual software problems all soluble but …

Structured design

- Only practical way forward is modularization
- Chop complex systems into simpler components
- Define clear APIs between them
- Sometimes task division seems straightforward (bank = tellers, ATMs, dealers, …)
- Sometimes it isn't, or it turns out to be deceptive
- Many methodologies have been developed to deal with this (Jackson, Yourdon, SSADM, UML…)

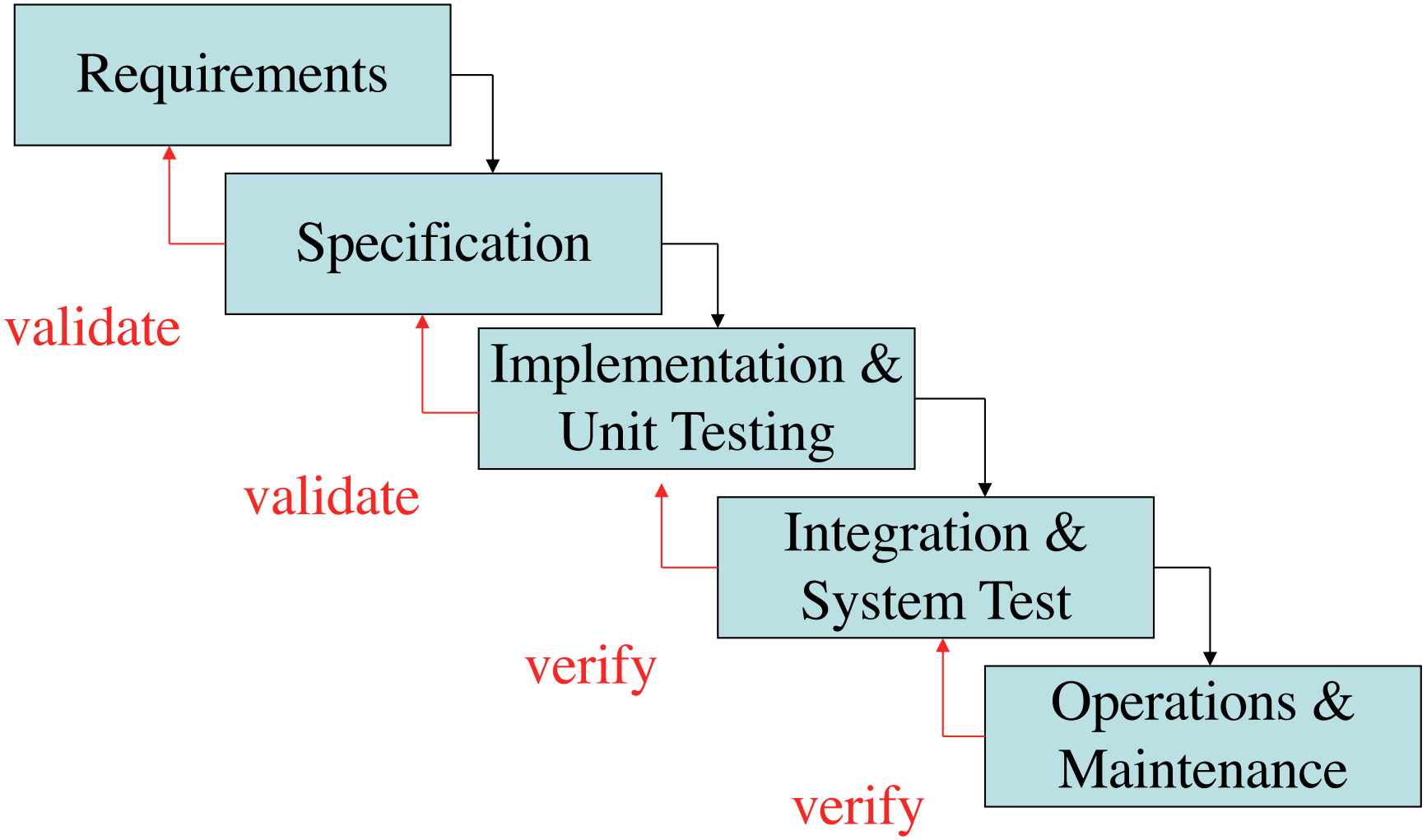
The waterfall model



The waterfall model (2)

- Requirements are written in the user's language
- The specification is written in system language
- There can be many more steps than this – system spec, functional spec, programming spec …
- The philosophy is progressive refinement
- Warning! When Winton Royce published this in 1970 he cautioned against naïve use
- But it become a US DoD standard, and UK too – not to mention sector safety standards (e.g.health)

The waterfall model (3)



The waterfall model (4)

- People often suggest adding an overall feedback loop from ops back to requirements
- However the essence of the waterfall model is that this isn't done
- It would erode much of the value that organisations get from top-down development
- Very often the waterfall model is used only for specific development phases, e.g. adding a feature
- But sometimes people use it for whole systems

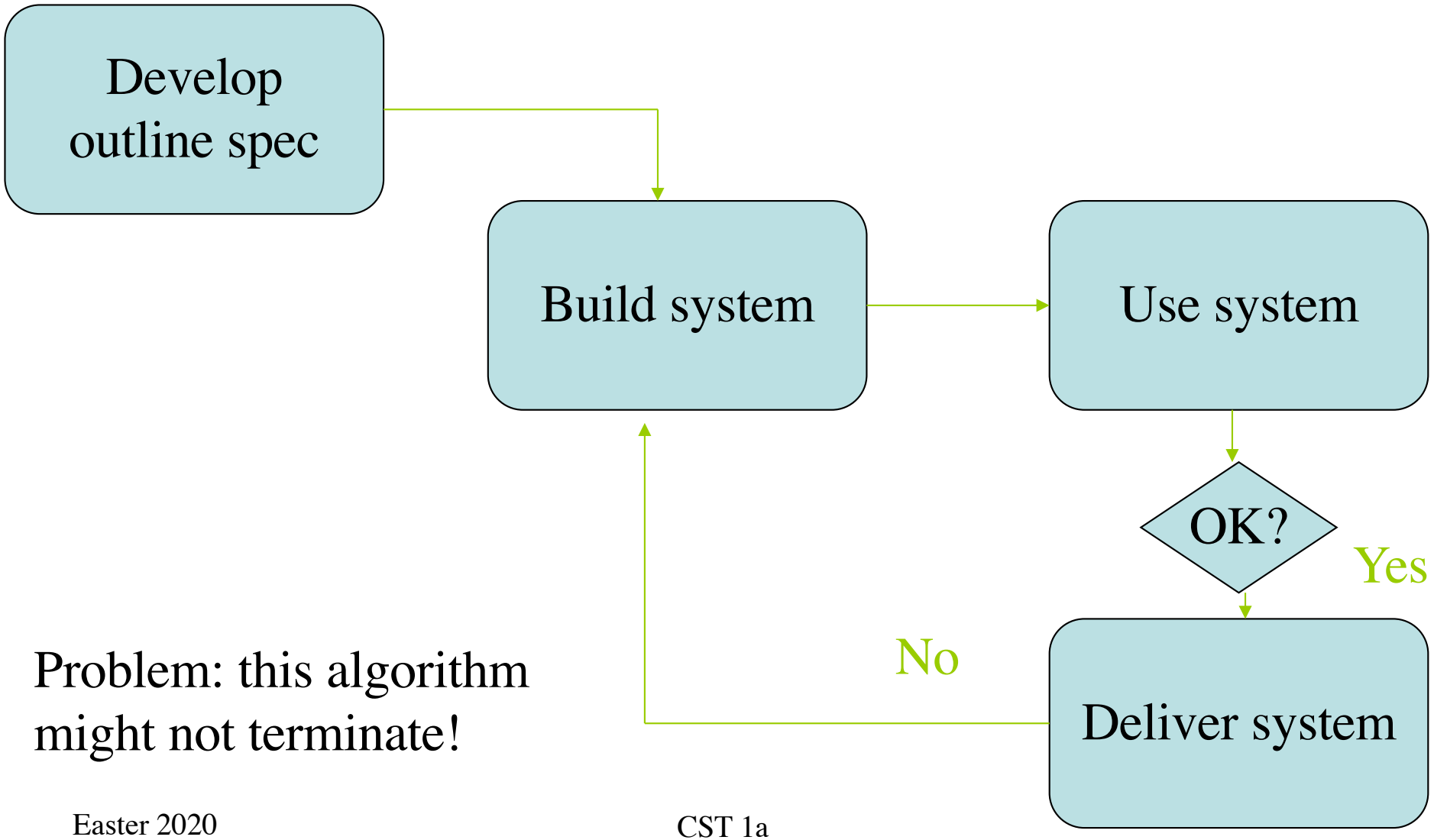
Waterfall – advantages

- Compels early clarification of system goals and is conducive to good design practice
- Enables the developer to charge for changes to the requirements
- It works well with many management tools, and technical tools
- Where it's viable it's usually the best approach
- The really critical factor is whether you can define the requirements in detail in advance. Sometimes you can (Y2K bugfix); sometimes you can't (HCI)

Waterfall – objections

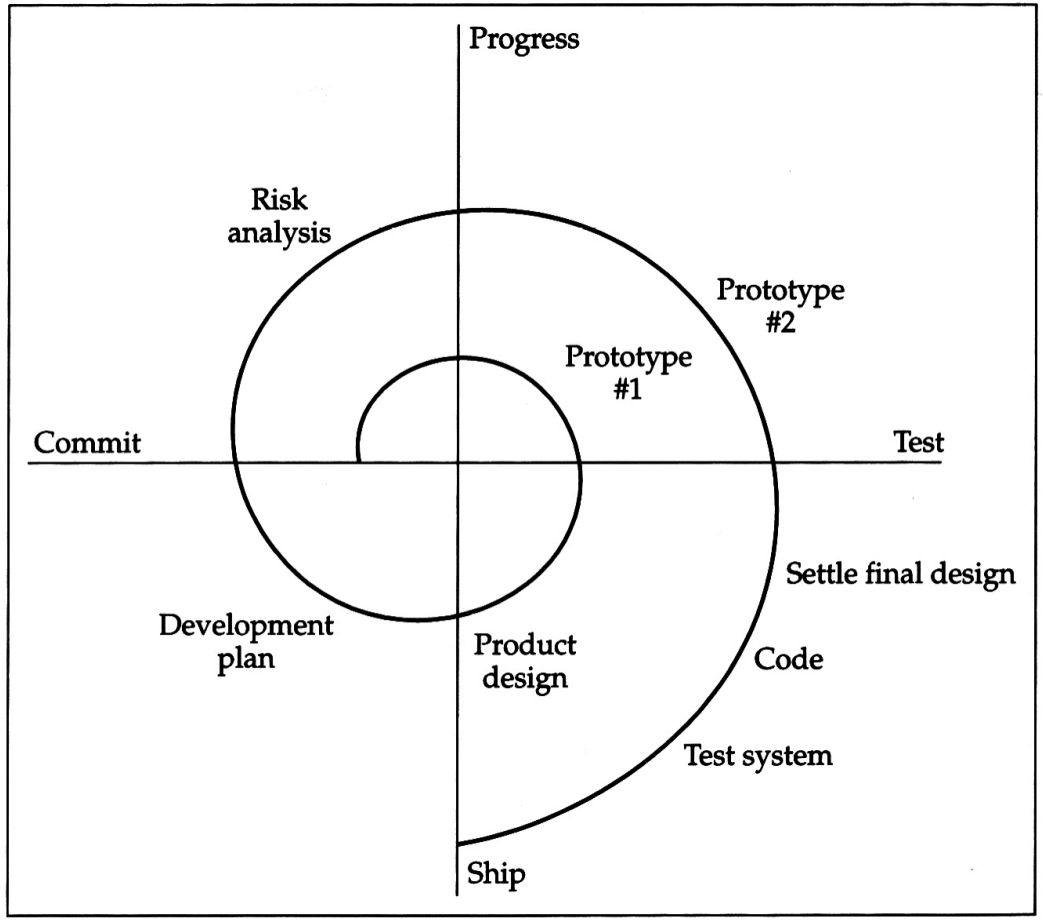
- Iteration can be critical in the development process:
  - requirements not yet understood
  - the technology is changing
  - the environment (legal, competitive) is changing
  - an existing product is getting regular small enhancements
- The attainable quality improvement may be unimportant over the system lifecycle
- It's used to loot naïve customers like government: when the system doesn't work it's the customer's fault as he signed off the specification

Iterative development



Problem: this algorithm might not terminate!

Spiral model



Spiral model (2)

- The essence is that you decide in advance on a fixed number of iterations
- E.g. engineering prototype, pre-production prototype, then product
- Each of these iterations is done top-down
- “Driven by risk management”, i.e. you put your energy into prototyping the bits you don’t understand yet

Evolutionary model

- By the 1990s, products like Windows and Office were so complex that they had to evolve (MS tried to rewrite Word from scratch twice and failed)
- The big change that made code evolution possible was the arrival of automatic regression testing
- Firms now have huge suites of test cases against which daily builds of the software are tested
- The development cycle is to add changes, check them in, and test them

Evolutionary model (2)

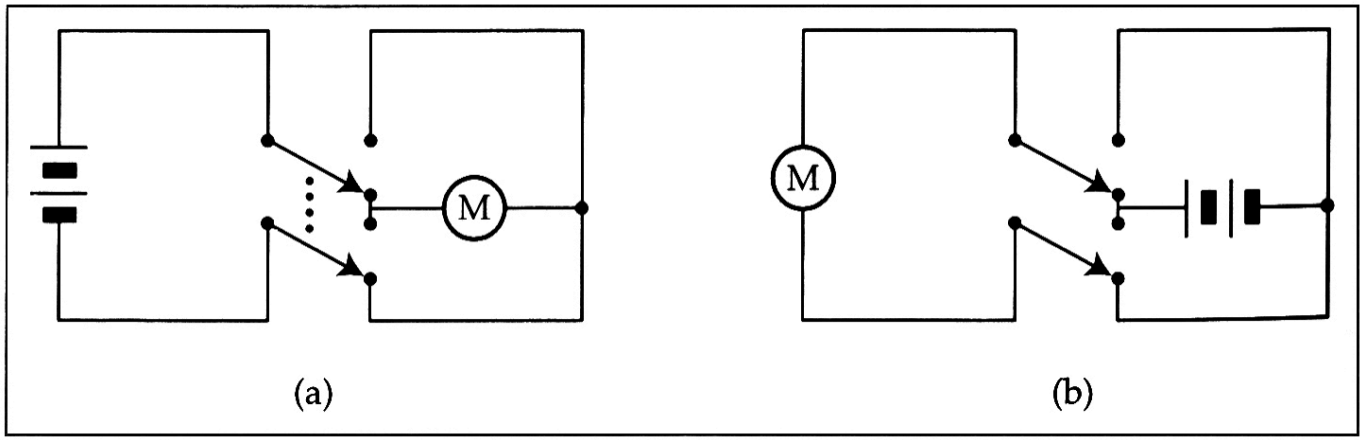
- A modern integrated development environment has several components
  - Code and documentation version control (git)
  - Code review (gerrit)
  - Automated build (make)
  - Continuous integration (Jenkins)
- This technology has had a huge effect on industry over the last 20 years as it evolved
- Think how you'll set up your group project!

Dependability

- Many systems must avoid a certain class of failures with high assurance
  - safety critical systems – failure could cause death, injury or property damage
  - security critical systems – failure could allow leakage of confidential data, fraud, …
  - real time systems – software must accomplish certain tasks on time
- Critical computer systems have much in common with mechanical systems (bridges, brakes, locks)
- Key insight: engineers study how things fail

Tacoma Narrows, Nov 7 1940

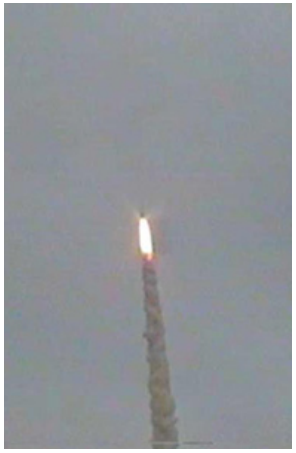
Hazard elimination



- Which motor reversing circuit above is the safe one?
- Some architecture and tool choices can eliminate whole classes of software hazards, e.g. using a strongly-typed language to limit syntax errors and memory leaks...
- But usually hazards involve more than just one program

Ariane 5, June 4 1996






- Ariane 5 accelerated faster than Ariane 4
- This caused an operand error in float-to-integer conversion
- The backup inertial navigation set dumped core
- The core was interpreted by the live set as flight data
- Full nozzle deflection $\rightarrow$ 20^{o} angle of attack $\rightarrow$ booster separation

Multi-factor failure

- Many safety-critical system failures involve multiple things going wrong at once
- It would be great to have no arithmetic or bounds errors but you have to be careful with exception handling
- Redundancy is also difficult to manage
- Criticality of timing tests the limits of simple verification techniques
- Testing can also be really hard

Emergent properties

- As safety is a system property, it has to be dealt with holistically
- The same goes for security, and real-time performance too
- As we mentioned in lecture 1, a very common error is not getting the scope right
- As we discussed in lecture 3, designers often don't do enough work on human factors such as usability and training

The Therac accidents

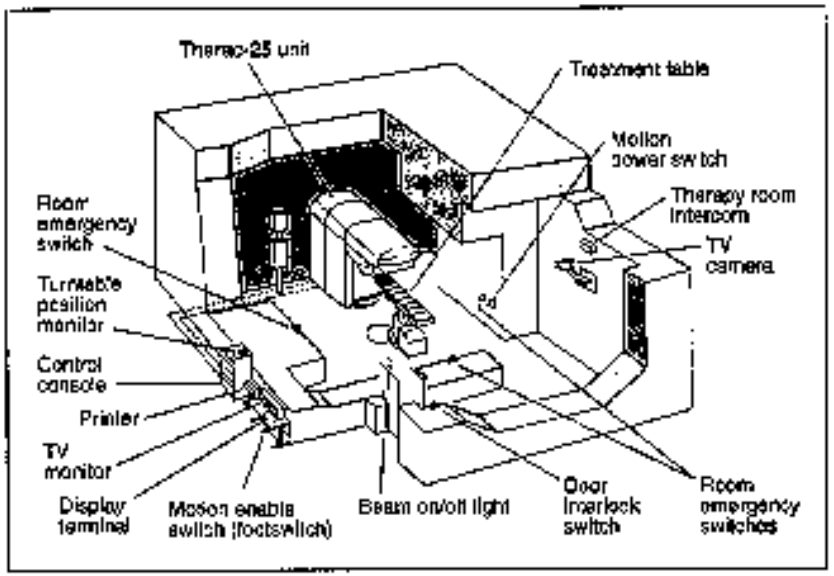
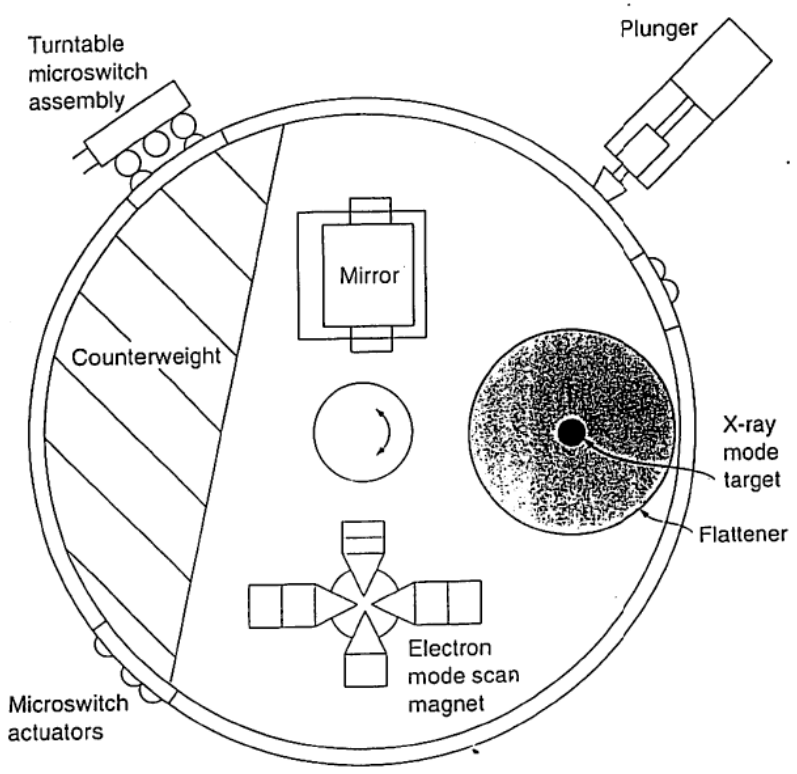


Figure 1. Typical Therac-25 facility.

- The Therac-25 was a radiotherapy machine sold by AECL
- Between 1985 and 1987 three people died in six accidents
- Example of a fatal coding error, compounded with usability problems and poor safety engineering

The Therac accidents (2)



- 25 MeV 'therapeutic accelerator' with two modes of operation
  - 25MeV focused electron beam on target to generate X-rays
  - 5-25MeV spread electron beam for skin treatment (with 1% of beam current)
- Safety requirement: don't fire 100% beam at human!

The Therac accidents (3)

- Previous model (Therac 20) had mechanical interlocks to prevent high-intensity beam use unless X-ray target in place
- The Therac-25 replaced these with software
- Fault tree analysis arbitrarily assigned probability of 10^{-11} to 'computer selects wrong energy' and 10^{-4} to software bugs
- Code was poorly written, unstructured and not really documented

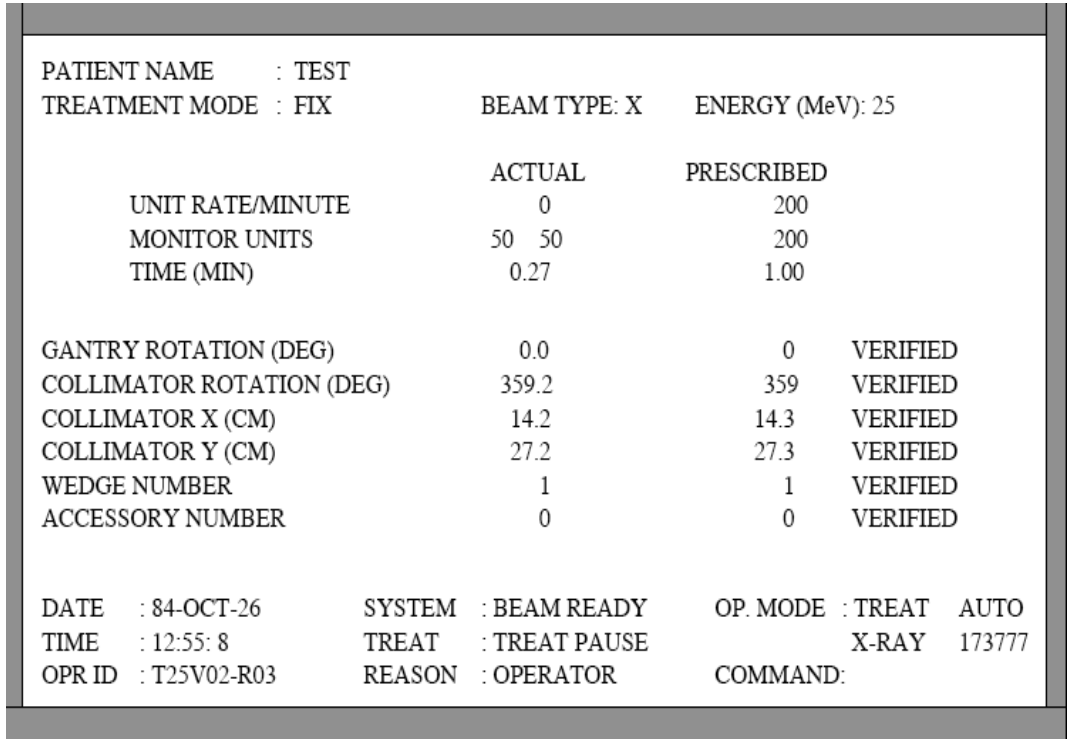
The Therac accidents (4)

- Marietta, GA, June 85: woman's shoulder burnt. Settled out of court. FDA not told
- Ontario, July 85: woman's hip burnt. AECL found microswitch error but could not reproduce fault; changed software anyway
- Yakima, WA, Dec 85: woman's hip burned. 'Could not be a malfunction'

The Therac accidents (5)

- East Texas Cancer Centre, Mar 86: man burned in neck and died five months later of complications
- Same place, three weeks later: another man burned on face and died three weeks later
- Hospital physicist managed to reproduce flaw: if parameters changed too quickly from x-ray to electron beam, the safety interlock failed
- Yakima, WA, Jan 87: man burned in chest and died – due to different bug now thought to have caused Ontario accident

The Therac accidents (6)



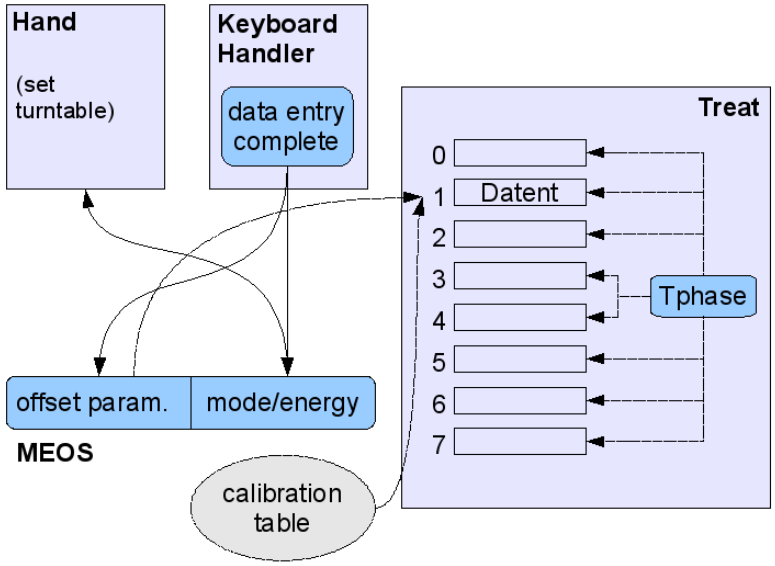
PATIENT NAME : TEST
TREATMENT MODE : FIX BEAM TYPE: X ENERGY (MeV): 25

	ACTUAL	PRESCRIBED	
UNIT RATE/MINUTE	0	200	
MONITOR UNITS	50 50	200	
TIME (MIN)	0.27	1.00	
GANTRY ROTATION (DEG)	0.0	0	VERIFIED
COLLIMATOR ROTATION (DEG)	359.2	359	VERIFIED
COLLIMATOR X (CM)	14.2	14.3	VERIFIED
COLLIMATOR Y (CM)	27.2	27.3	VERIFIED
WEDGE NUMBER	1	1	VERIFIED
ACCESSORY NUMBER	0	0	VERIFIED

DATE : 84-OCT-26 SYSTEM : BEAM READY OP. MODE : TREAT AUTO
TIME : 12:55: 8 TREAT : TREAT PAUSE X-RAY 173777
OPR ID : T25V02-R03 REASON : OPERATOR COMMAND:

- East Texas deaths caused by editing ‘beam type’ too quickly
- This was due to poor software design

The Therac accidents (7)



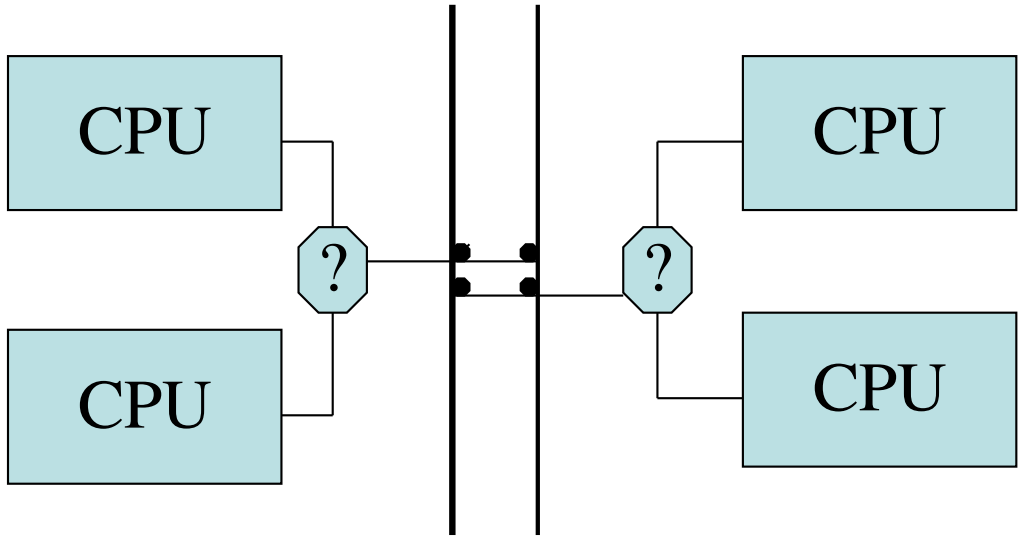
- Datent sets turntable and 'MEOS', which sets mode and energy level
- 'Data entry complete' can be set by datent, or keyboard handler
- If MEOS set (& datent exited), then MEOS could be edited again

The Therac accidents (8)

- AECL had ignored safety aspects of software
- Confused reliability with safety
- Lack of defensive design
- Inadequate reporting, followup and regulation – didn't explain Ontario accident at the time
- Unrealistic risk assessments
- Inadequate software engineering practices – spec an afterthought, complex architecture, dangerous coding, little testing, careless HCI design…
- AECL got out of the medical equipment business. But similar accidents are still happening! (NY Times article)

Redundancy

- Some vendors, like Stratus, developed redundant hardware for ‘non-stop processing’



Redundancy (2)

- Stratus users found that the software is then where things broke
- The ‘backup’ IN set in Ariane failed first!
- Next idea: multi-version programming
- But: errors are correlated, dominated by failure to understand requirements (Leveson)
- Implementations often give different answers
- With both types of errors, redundancy is hard!

Redundancy management – 737

Panama crash, June 6 1992

- Need to know which way up!
- New EFIS (each pilot), WW2 artificial horizon (top right)
- EFIS failed – loose wire
- Both EFIS fed off same IN set
- Pilots watched EFIS, not AH
- 47 fatalities
- And again: Korean Air cargo 747, Stansted Dec 22 1999

Kegworth crash, Jan 8 1989

- BMI London-Belfast, fan blade broke in port engine
- Crew shut down starboard engine and did emergency descent to East Midlands
- Opened throttle on final approach: no power
- 47 dead, 74 injured
- Initially blamed wiring technician! Later: cockpit design

Complex socio-technical systems

- Civil aviation is a relatively simple case for a number of reasons
  - It's been running since 1919, in modern form since 1945
  - Stable components: aircraft design, avionics design, pilot training, air traffic control …
  - Interfaces are stable too
  - Crew capabilities are well known
  - There are better incentives for learning than with medical devices!
- But institutional failures can still happen, as with the LAS

The Boeing 737 Max

- Two crashes, in Indonesia in 2018 and in Ethiopia in 2019, killing 346
- 737 Max fleet grounded, then production halted
- Boeing lost $18.7bn in lost sales / compensation by March 2020; market cap over $60bn down
- The world's biggest software failure yet in terms of lost lives and economic damage
- Boeing and the FAA made a lot of the mistakes we've seen already, and then some

MCAS

- The Boeing model 737 just evolved for 60 years to save costs of certification, pilot retraining
- It needed bigger engines to save fuel and compete with Airbus, so engines were moved forward to fit
- Test pilots discovered they couldn’t easily trim the plane at high speed
- The fix was the Maneuvering Characteristics Augmentation System (MCAS) – software added to an existing flight control computer

The fatal design error

- The flight control computer got input from two angle of attack (AoA) sensors, but the MCAS software used only one of them
- AoA sensors are regularly damaged by bird strikes, ground crew etc
- Uncommanded nose-down trim happened when the single AoA sensor failed and the pilot used electric trim (slightly flaky logic, like Therac)
- Pilots needed 40–50kg force to keep the nose up, struggled, and eventually lost

Aggravating factors

- In the safety analysis, ‘Unintended MCAS activation’ was rated ‘major’ (= maybe someone gets injured) rather than ‘catastrophic’ (= everyone gets killed)
- So no proper FMEA was done
- MCAS was removed from the pilot manual
- Boeing also failed to anticipate cockpit chaos
- Some years before, accountants had taken over company management from engineers…

Institutional factors

- Boeing had taken over much of the safety assurance from the FAA's own staff
- After a slightly similar 2009 accident in the Netherlands with the previous model 737, they got Uncle Sam to arm-twist the Dutch investigators
- They hoped the Indonesia crash was pilot error
- The FAA realized by then that there was a problem but let the US fleet continue flying
- Such arrangements are called 'Regulatory capture'

Pulling it together

- First, understand and prioritise hazards (see the video on the 737 Max for what can go wrong here)
- Develop safety case: hazards, risks, and strategy per hazard (avoidance, constraint)
- Who will manage what?
- Trace constraints to code, and identify critical components / variables to developers
- Develop test plans, certification, training, etc
- Figure out how this fits with your development methodology

Pulling it together (2)

- If you possibly can, tie down the critical properties (safety, security, performance) early
- 'Shift left'
  - In a waterfall development, get them in the spec
  - In a spiral model, sort them at prototype stage
  - In an evolutionary model, get them into code (DevOps becomes DevSecOps)
- At least, that's how you do it in an ideal world!
- Often reality is more challenging

Pulling it together (3)

- Managing an emergent property – safety, security, real-time performance – can be intrinsically hard
- Although some failures happen during the 'techie' phases of design and implementation, most happen before or after
- The soft spots are requirements engineering, certification, and then operations / maintenance
- These are interdisciplinary, involving systems people, domain experts and users, cognitive factors, politics and marketing
- We'll have more on certification later

Tools and methods

- Homo sapiens invents and uses tools when some parameter of a task exceeds our native capacity
  - Heavy object: raise with lever
  - Tough object: cut with axe
  - …
- Software engineering tools are designed to deal with complexity

Tools and methods (2)

- There are two types of complexity:
  - **Incidental complexity** dominated programming in the early days, e.g. keeping track of stuff in machine-code programs. Solution: high-level languages
  - **Intrinsic complexity** is the main problem today, e.g. complex system (such as a bank) with a big team. 'Solution': structured development, project management tools, …
- We can aim to eliminate the incidental complexity, but the intrinsic complexity must be managed

Incidental complexity (1)

- The greatest single improvement was the invention of high-level languages like FORTRAN
  - 2000 loc/year goes much farther than assembler
  - Code easier to understand and maintain
  - Appropriate abstraction: data structures, functions, objects rather than bits, registers, branches
  - Structure lets many errors be found at compile time
  - Code may be portable; at least, the machine-specific details can be contained
- Performance gain: 5–10 times. As coding = 1/6 cost, better languages give diminishing returns

Incidental complexity (2)

- Thus most advances since early HLLs focus on helping programmers structure and maintain code
- Don't use 'goto' (Dijkstra 68), structured programming, pascal (Wirth 71); info hiding plus proper control structures
- OO: Simula (Nygaard, Dahl, 60s), Smalltalk (Xerox 70s), C++, Java … covered elsewhere
- Don't forget the object of all this is to manage complexity!

Incidental complexity (3)

- Early batch systems were very tedious for developers … e.g. our school computer in 1972
- Time-sharing systems allowed online test – debug – fix – recompile – test – …
- This still needed plenty scaffolding and carefully thought out debugging plan
- Integrated programming environments such as TSS, Turbo Pascal,…
- Some of these started to support tools to deal with managing large projects – ‘CASE’

Formal methods

- Pioneers such as Turing talked of proving programs correct
- Pioneered by Floyd (67), Hoare (71), … now many variants (Z for specifications, HOL for hardware, various theorem provers…)
- Can find subtle bugs, especially in conceptually difficult tasks
- Two basic approaches (academic v industrial)
  - Find all the bugs in a small program
  - Find many of the bugs in a large one

Static analysis tools are a useful result of formal methods

DOI:10.1145/1646353.1646374

How Coverity built a bug-finding tool, and a business, around the unlimited supply of bugs in software systems.

BY AL BESSEY, KEN BLOCK, BEN CHELF, ANDY CHOU, BRYAN FULTON, SETH HALLEM, CHARLES HENRI-GROS, ASYA KAMSKY, SCOTT MCPEAK, AND DAWSON ENGLER

A Few Billion Lines of Code Later

Using Static Analysis to Find Bugs in the Real World

like all static bug finders, leveraged the fact that programming rules often map clearly to source code; thus static inspection can find many of their violations. For example, to check the rule "acquired locks must be released," a checker would look for relevant operations (such as `lock()` and `unlock()`) and inspect the code path after flagging rule disobedience (such as `lock()` with no `unlock()` and double locking).

For those who keep track of such things, checkers in the research system typically traverse program paths (flow-sensitive) in a forward direction, going across function calls (inter-procedural) while keeping track of call-site-specific information (context-sensitive) and toward the end of the effort had some of the support needed to detect when a path was infeasible (path-sensitive).

A glance through the literature reveals many ways to go about static bug finding.[1,2,4,7,8,11] For us, the central religion was results: If it worked, it was good, and if not, not. The ideal: check millions of lines of code with little manual setup and find the maximum number of serious true errors with the minimum number of false reports. As much as possible, we avoided using annotations or specifications to reduce

Individual / group productivity

- ‘Chief programmer teams’ (IBM, 70–72): capitalise on wide productivity variance
- Team of chief programmer, apprentice, toolsmith, librarian, admin assistant etc, to get maximum productivity from your staff
- Can be effective during implementation
- But each team can only do so much
- Why not just fire the less productive programmers? Or only hire after a trial contract?

Capability maturity model

- Watts Humphrey, 1989: it's best to keep teams together, as productivity grows over time
- Nurture the capability for repeatable, manageable performance, not outcomes that depend on individual heroics
- CMM developed at Software Engineering Institute at Carnegie Mellon University (also runs CERT)
- It identifies five levels of increasing maturity in a team or organisation, and a guide for moving up

Capability maturity model (2)

1. Initial (chaotic, ad hoc) – the starting point for use of a new process
2. Repeatable – the process is able to be used repeatedly, with roughly repeatable outcomes
3. Defined – the process is defined/confirmed as a standard business process
4. Managed – the process is managed according to the metrics described in the Defined stage
5. Optimized – process management includes deliberate process optimization/improvement

Trends in development methods

- Over the past 20 years, emphasis has shifted from requirements to testing to people
- 1990s: put a lot of effort into the spec
- 2000s: the major effort is in an incremental build system, with an automatic regression test environment
- Can be simple, or an expensive “lab car”
- Foundation for the next step

Agile development – beginnings

- ‘Extreme Programming’ (Beck, 99): aimed at small teams working on iterative development with automated tests and short build cycle
- ‘Solve your worst problem. Repeat’
- Focus on development episode: write tests first, then the code. ‘The tests are the documentation’
- Programmers work in pairs, at one keyboard and screen
- That didn’t survive, but episodes did, and people added the ‘scrum’

Agile development – now

- Start with a sound technical foundation: languages, build environment, testing
- Agree processes: tickets, daily scrum, weekly lunch, customer interaction...
- Break the development into short sprints
- Figure out what else is needed (e.g., updates to security policy or safety case) and ‘move left’
- As infrastructure becomes a service, move site reliability engineering left too

Testing

- Testing is often neglected in academia, but it's typically about half the effort, and half the cost
- Bill G: "are we in the business of writing software, or test harnesses?"
- Happens at many levels
  - Design validation, UX prototyping
  - Module test after coding
  - System test after daily build
  - Beta test / field trial
  - Subsequent litigation
- Cost per bug rises dramatically down this list!

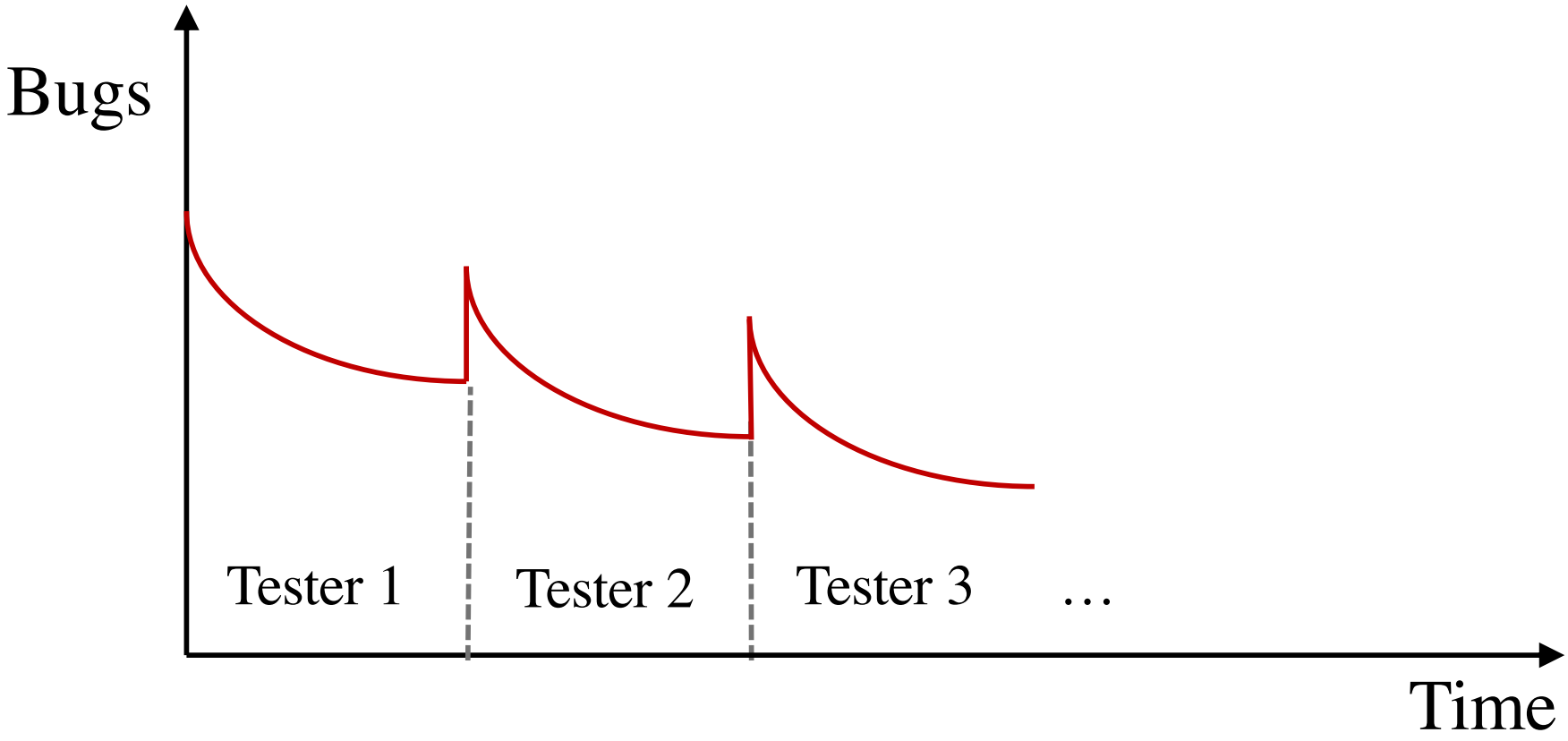
Testing (2)

- The big advance: design for testability, automated regression tests, continuous integration
- Regression tests check that new versions of your software give same answers as older versions
  - Before regression testing, 20% of bug fixes used to reintroduce failures in already tested behaviour
  - Customers more upset by failure of a familiar feature than at a new feature that's a bit flaky
  - So test the inputs your users will actually generate!
- Add fuzzing too: test lots of random inputs

Testing (3)

- Reliability growth models help us assess MTBF, bugs remaining, economics of testing
- Failure rate due to one bug is $e^{-k/T}$; with many bugs these sum to k/T
- So for 10^9 hours mtbf, must test $>10^9$ hours
- New testers bring new bugs to light, as their test focus is different
- Incentives matter: hostile testers used by military, NASA etc; most large software and service firms use bug bounty programmes

More testers find more bugs



Think about diversity & inclusion

Check your photo

1. Background and lighting

Your photo must have:

- a plain light-coloured background – without texture or pattern
- balanced light – no shadows on your face or behind you
- no objects behind you

2. Your appearance

Make sure:

- the photo is a good likeness taken in the last month
- your whole face is visible with your eyes open
- you have a plain expression – no smile and mouth closed
- there are no reflections or glare (if you have to wear glasses)
- you're not wearing headwear (unless for religious or medical reasons)

3. Photo quality and format

Your photo must:

- be in colour, with no effects or filters
- not be blurred or have 'red eye'
- be unedited – you can't 'correct' your passport photo

Our automated check suggests

- we can't find the outline of your head

Your photo

"Today, I simply wanted to renew my passport online. After numerous attempts and changing my clothes several times, this example illustrates why I regularly present on Artificial Intelligence/Machine Learning bias, equality, diversity and inclusion"
@CatHallam1

The spec still matters!

- Classic study of failure of 17 large demanding systems by Curtis, Krasner and Iscoe
- Causes of failure:
  1. Thin spread of application domain knowledge
  2. Fluctuating and conflicting requirements
  3. Breakdown of communication, coordination
- They were very often linked, and the typical progression to disaster was $1 \rightarrow 2 \rightarrow 3$
- For large upgrades this is still a big deal

Maintaining the spec is hard work

- Thin spread of application domain knowledge
  - How many people understand everything about running a hospital / building an airliner?
  - Some fields try hard to be open, e.g. aviation
  - But many details are jealously guarded turf
  - Comms complexity for N people can be N^2 or 2^N!
  - So you get mistakes with new products / big upgrades
- The spec may change in midstream anyway
  - Competing products, new standards, new tech
  - Changing environment (takeover, election, …)
- Don't let the spec fragment! Someone must own it

Safety case maintenance

- Big issue with medical devices – post-market surveillance (being worked on)
- Vendors prefer to front-load certification, whose costs deter new market entrants, and dislike recalls, which are expensive
- Similar patterns with cars, aircraft…
- The move to autonomy is causing safety and security to become entangled

Vulnerability lifecycle

- An engineer introduces a bug
- Someone discovers it: now a ‘zero day’
- Disclose responsibly; or at once; or exploit
- Primary exploit window till patch shipped
- But many devices aren’t patched (orphan products like old phones)
- What do we do about Mirai?

Coordinated disclosure

- Bad old days: firms tried to deny existence of bugs, and threatened people who disclosed them – to save costs of fixing
- Reaction: hackers disclosed bugs anyway leading to instant exploits
- Consensus arose in 2000s: vulnerabilities should be disclosed after a time delay
- ‘Responsible’ or ‘coordinated’ disclosure
- Can use CERTs, regulators as channel

How do you know when you're done?

- The Cathedral
  - safety: dozens of sectoral regulators (planes: FAA/CAA, medical: FDA/MHRA…)
  - security is messier because it's adversarial but has sectoral standards too (PCI for payments…)
- Or the Bazaar
  - patch cycle, fed by
  - breach reporting, coordinated disclosure

Focus on outcomes, or process?

- Outcomes
  - Metrics easier for regular losses (risk)
  - But people worry more about rare but publicized bad things (recall bias)
  - Rare catastrophes are harder still (uncertainty)
  - So are attacks! (we fear hostile intent)
  - Product liability (more in Economics, Law and Ethics in 1b)

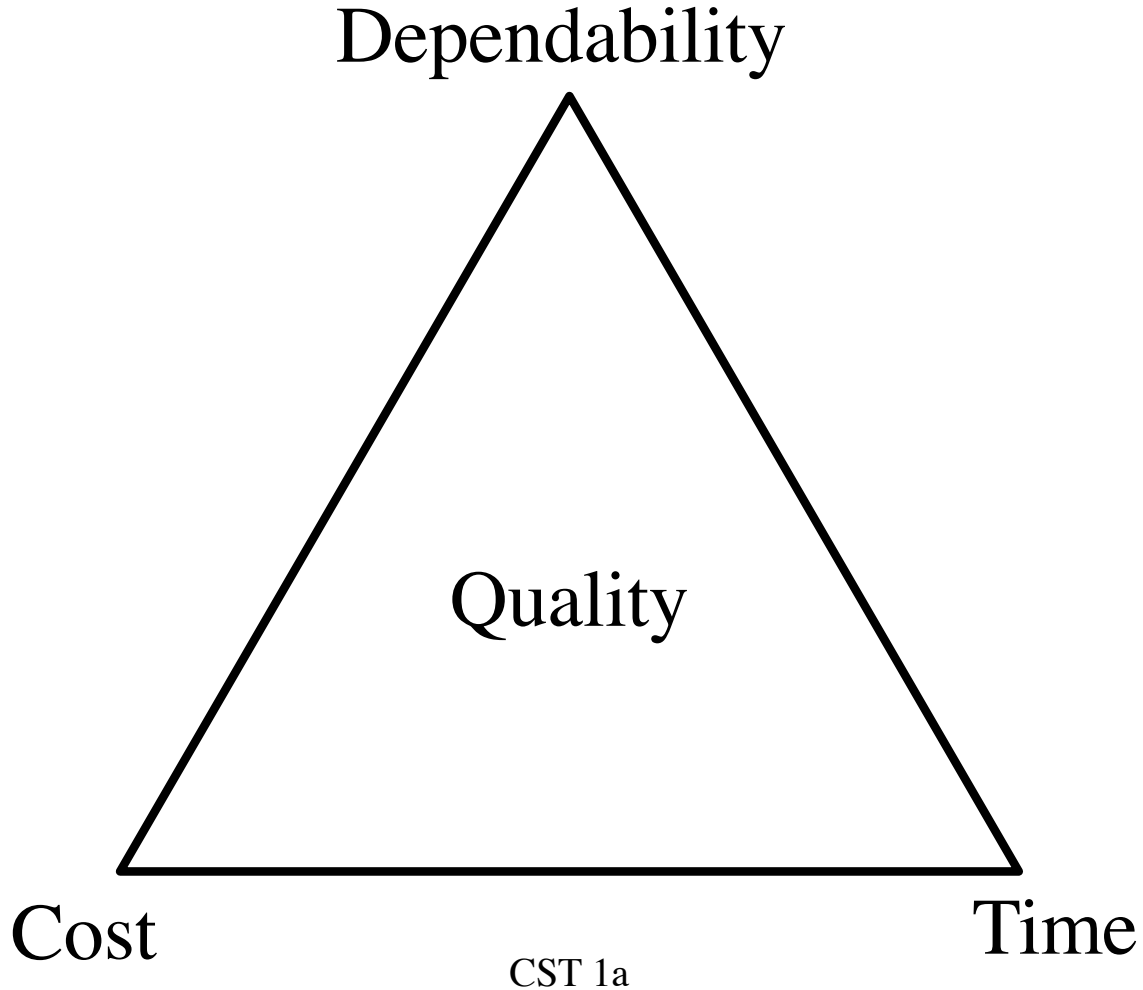
Focus on outcomes, or process?

- Process
  - Safety regulators are mostly strong on process
  - Safety / security development lifecycle
  - Public sector is keen on ‘compliance’ (blame avoidance is what bureaucracies do)
  - But standards must adapt as environment changes: ‘always fighting the last war’
  - Still a gap of residual risk / uncertainty

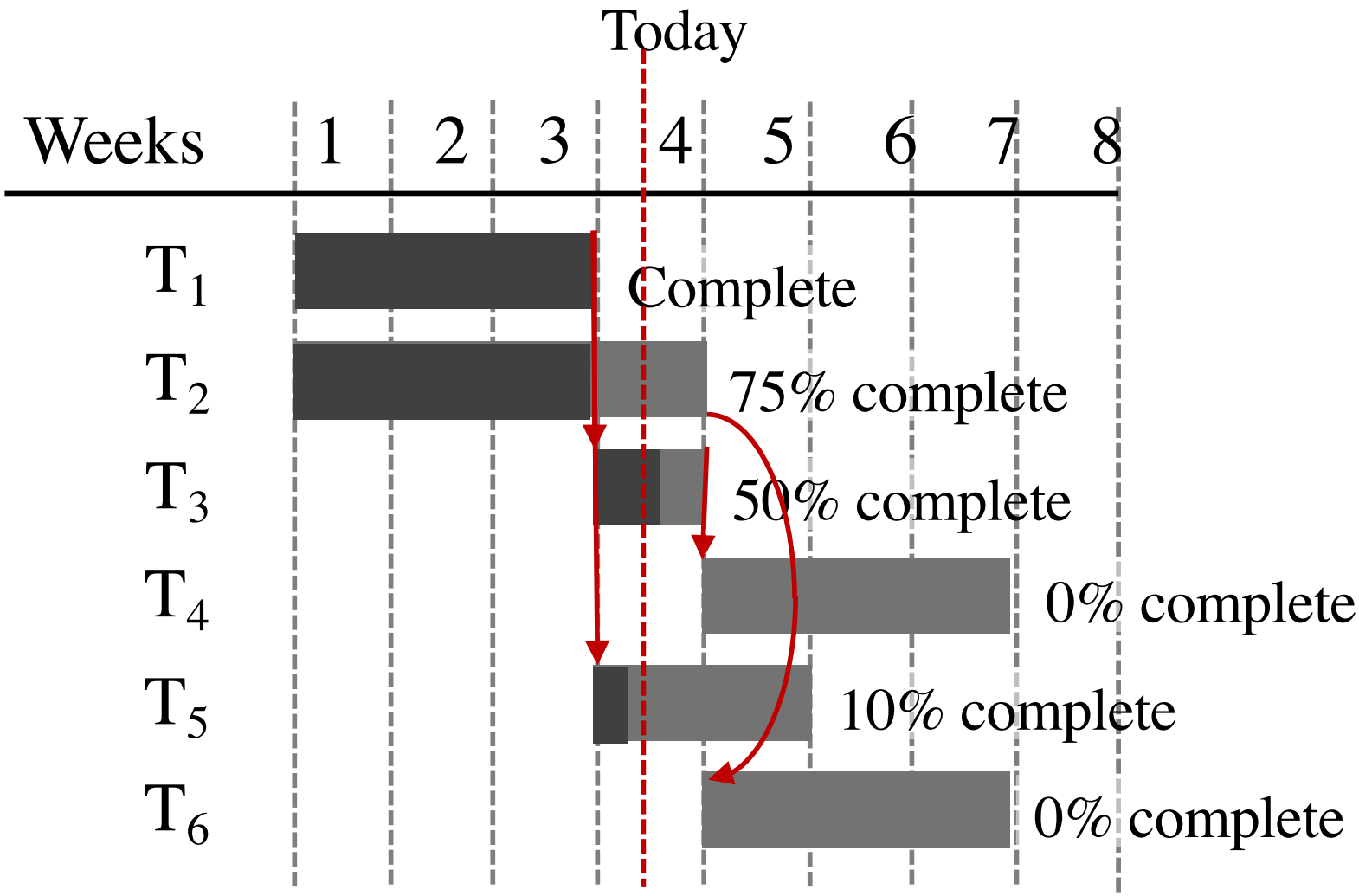
Project management

- A manager's job is to
  - Plan
  - Motivate
  - Control
- The skills involved are interpersonal, not techie; but managers must retain respect of techie staff
- Growing software managers a perpetual problem! 'Managing programmers is like herding cats'
- Nonetheless there are some tools that can help

Project management trilemma
– right, quick or cheap (choose any two)

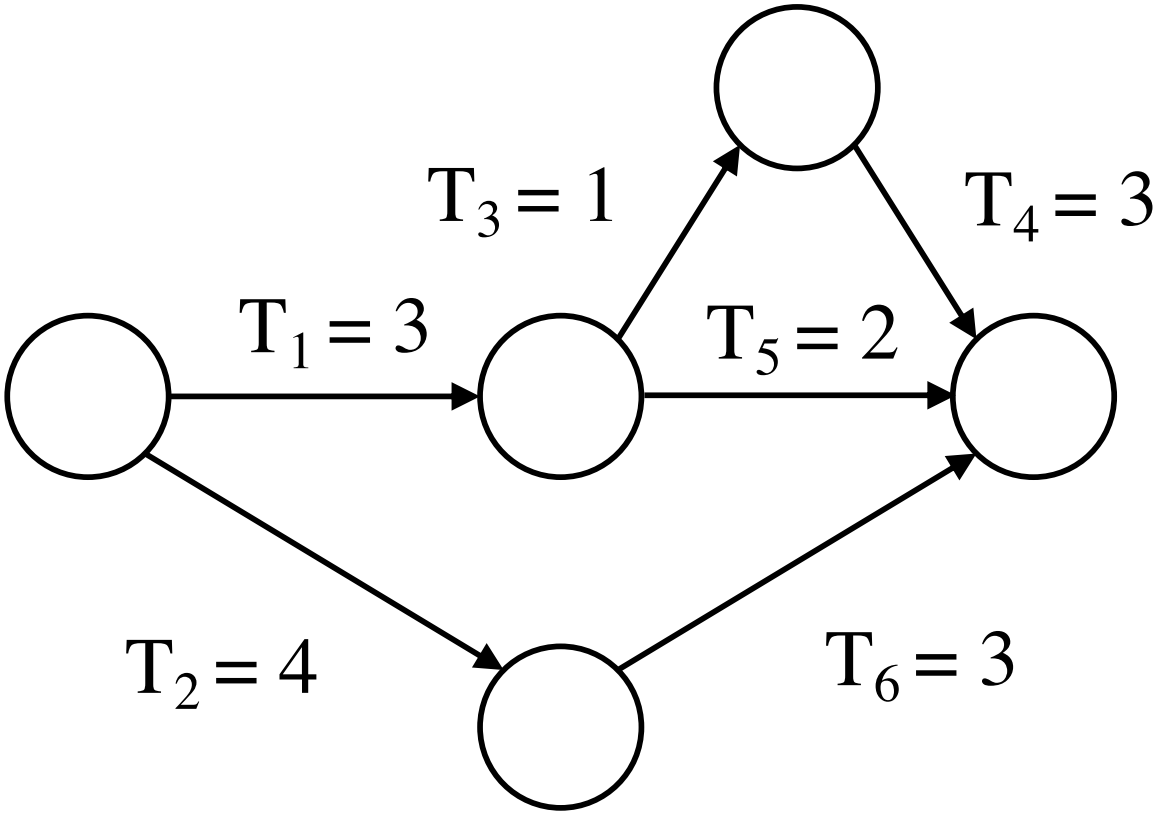


Gantt charts: tasks and milestones



Can be hard to visualise dependencies in large charts

PERT charts: show critical paths



Which paths are critical?

Keeping people motivated

- People can work less hard in groups than on their own projects – ‘free rider’ or ‘social loafing’ effect
- Dan Rothwell’s ‘three C’s of motivation’:
  - Collaboration – everyone has a specific task
  - Content – everyone’s task clearly matters
  - Choice – everyone has a say in what they do
- Many other factors: acknowledgement, attribution, equity, discrimination, leadership, and ‘team building’ (shared food / drink / exercise; scrumming)

Documentation

- Think: how will you deal with management documents (budgets, PERT charts, staff schedules)
- And engineering documents (requirements, hazard analyses, specifications, test plans, code)?
- CS tells us it's hard to keep stuff in synch!
- Possible partial solutions:
  - High tech: integrated development environment
  - Bureaucratic: plans and controls department
  - Social consensus: style, comments, formatting

Change control and operations: important and can be overlooked

- Change control and config are critical; often poor
- Objective: manage testing and deployment
- Someone must assess risk and be responsible for:
  - Live running
  - Updates, patches
  - Manage backup, recovery, rollback
  - …
- DevOps integrates development and operations
- DevSecOps integrates monitoring, incident response

Shared infrastructure

- We share a lot of code through open source operating systems, libraries and tools
- Huge benefits but also interaction costs!
- How do you coordinate disclosure?
- How do you negotiate fixes with others who rely on your code / platform?
- Are you aware of different license terms?
- How will you cope with an emergency bug fix (like Heartbleed)?

The emerging challenge

- With the “Internet of Things”, safety now includes security
- Things like cars, medical devices and grid equipment have 10-year certification cycles
- Put software everywhere, and attacks scale!
- Expect many more devices to go to monthly updates like phones and laptops
- This will stress test a lot of regulators!

My big question (see 36C3 talk)

- Tesla has started monthly updates, like for laptops; other car vendors will follow
- That costs real money. So legacy vendors wanted to stop support after 6 years. But cars last 15+
- And: embedded carbon cost ~ lifetime fuel burn!
- Result: new EU Directive 2019/771
- So how will today's car software get patches in 2030? In 2040? In 2050?
- What new tools and new ideas do we need?

Conclusions

- Software engineering is about managing complexity. That's why it's hard. That's our trade
- We can cut incidental complexity using tools, but the intrinsic complexity remains
- Top-down approaches can sometimes help, but really large systems evolve
- Safety and security are often emergent properties
- Complex systems are usually socio-technical; people come into play as users, and also as members of development and other teams

Conclusions (2)

- Scaling is hard! Large firms behave differently from small dev teams. Once many teams work on a project, coordination scales poorly
- Architecture, tools, methods, culture and incentives can help
- In future, the confluence of security and safety may make maintenance the complexity limit, even more than at present!