Security and Software Engineering

CST Part 1a

Ross Anderson

Aims

- Introduce students to software engineering, and in particular to the problems of building
  - large 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 evolutionary 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
- You should 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’ (2^{nd} edition 2008), part 1 and chapters 25–26 (available online via my web page)
  - M Howard and D LeBlanc, ‘Writing Secure Code’, Microsoft press, 2003
  - N Leveson, ‘Safeware’ (1994; see also her ‘System Safety Engineering’ online)

Resources (2)

- Additional reading:
  - FP Brooks, ‘The Mythical Man Month’
  - J Reason, ‘The Human Contribution’
  - SW Thames RHA, ‘Report of the Inquiry into the London Ambulance Service’
  - S Maguire, ‘Writing Solid Code’, Microsoft 1993
  - H Thimbleby, ‘Improving safety in medical devices and systems’
  - MPP cases: O Campion-Awwad et al, ‘The National Programme for IT in the NHS – A Case History’
- And any application areas that interest you!

Outline

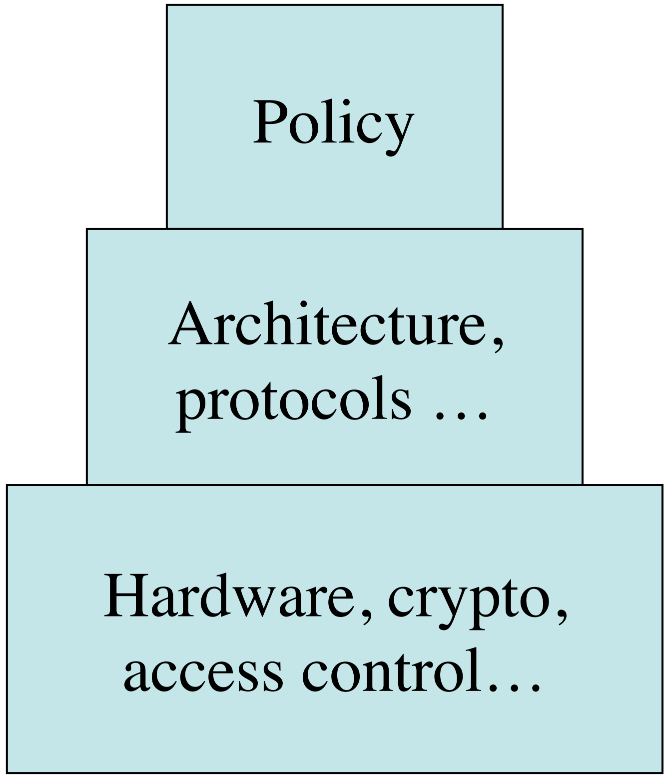
- Trying a different order from the booklet:
  - Security policy, safety case
  - Psychology
  - Protocols, software bugs of different types
  - The software crisis, the development lifecycle
  - Modern integrated development environments
  - Critical systems, combining safety and security
- No lecture May 1
- Guest lecture May 15 at 3pm in LT2, CL

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”

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 a technical term – mechanisms limiting the number of principals who can access information
- *Privacy* means control of your own secrets
- *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* is about restricting access to metadata. It has various flavours, from not being able to identify subjects to not being able to link their actions
- 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

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. an NSA man selling key material to the Chinese is trusted but not trustworthy (assuming his 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, plus conditions on the environment, which can lead to an accident in the event of failure
- Thus: failure + hazard = accident
- *Risk* is the probability of an accident, etc
- Thus: risk is hazard level combined with *danger* (probability hazard $\rightarrow$ accident) and *latency* (hazard exposure + duration); e.g. a *micromort*
- *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 security policy right in the early stages of the project
- Often it's iterative. Then the security requirements can get ignored, detached 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'
- Reinforce with material handling rules

Multilevel secure systems (MLS)

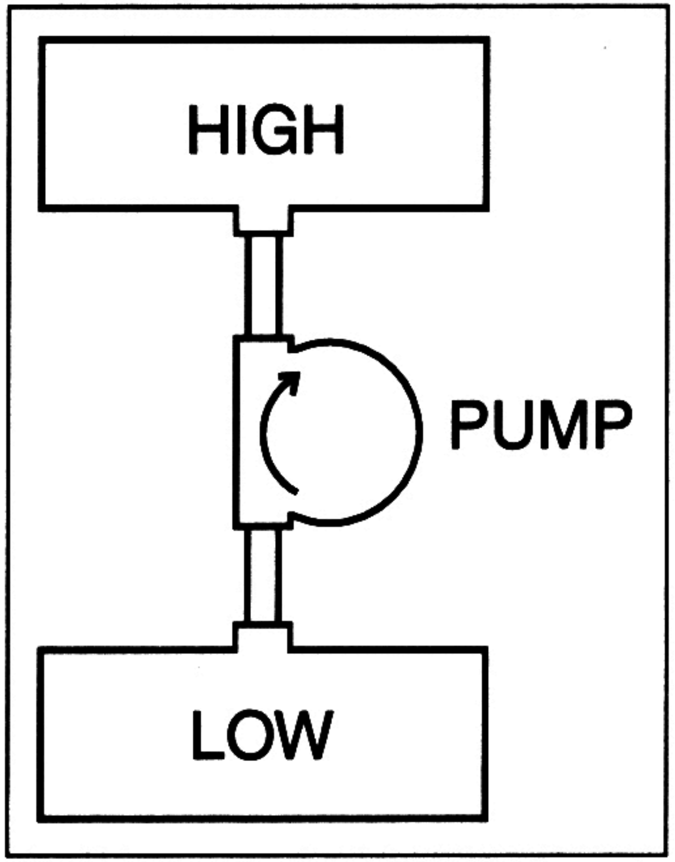
- Multilevel secure (MLS) systems are widely used in government
- They enforce standard handling rules for material at ‘Confidential’ ‘Secret’, ‘Top Secret’ etc.
- Resources have classifications; principals have clearances; clearance must equal or exceed classification; and information flows upwards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

Formalising the Policy

- Bell-LaPadula (1973):
  - *simple security policy*: no read up
  - **-policy*: no write down
- With these, one can prove that a system which 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) so it's verifiable
- 1970s idea: use a 'reference monitor', part of the operating system. Problem: this got complex, fast

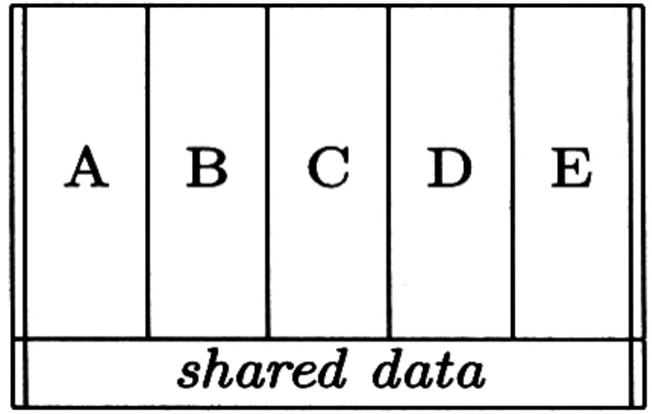
Typical MLS system

- Use architecture to get high assurance of the hey aspect of protection
- Example: stop classified data flowing from high to low using one-way flow
- Assurance then depends on a simple mechanism
- But keeping this small and verifiable 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: no reading from lower levels, or writing to higher ones
- Examples:
  - Medical device with 'calibrate' and 'operate' levels
  - grid control with safety as highest level, operational control at next level, then management, billing etc

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

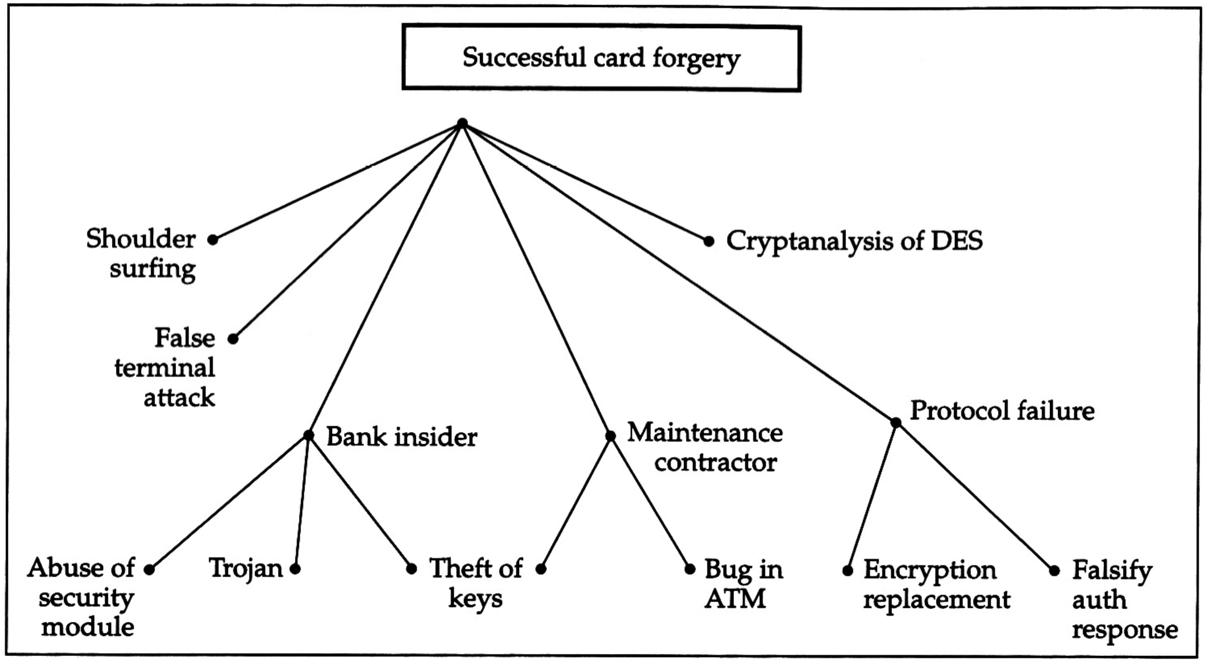
Other safety policies

- Industries have their own standards, cultures, often with architectural assumptions embedded in component design
- Over 180 regulations for cars alone – e.g. ABS mustn’t cause asymmetric braking
- Sometimes set top-down but in more mature industries safety standards tend to evolve
- Two basic ways to evolve them

Failure modes and effects analysis

- Understanding relationships between failures and outcomes can be bottom-up or top-down
- Bottom-up: ‘failure modes and effects analysis’ (FMEA) – developed by NASA
- Look at each component and list failure modes
- Figure out what you’ll do about each (cut the probability by overdesign? Redundancy?)
- Then use secondary mechanisms to deal with interactions

Fault tree analysis

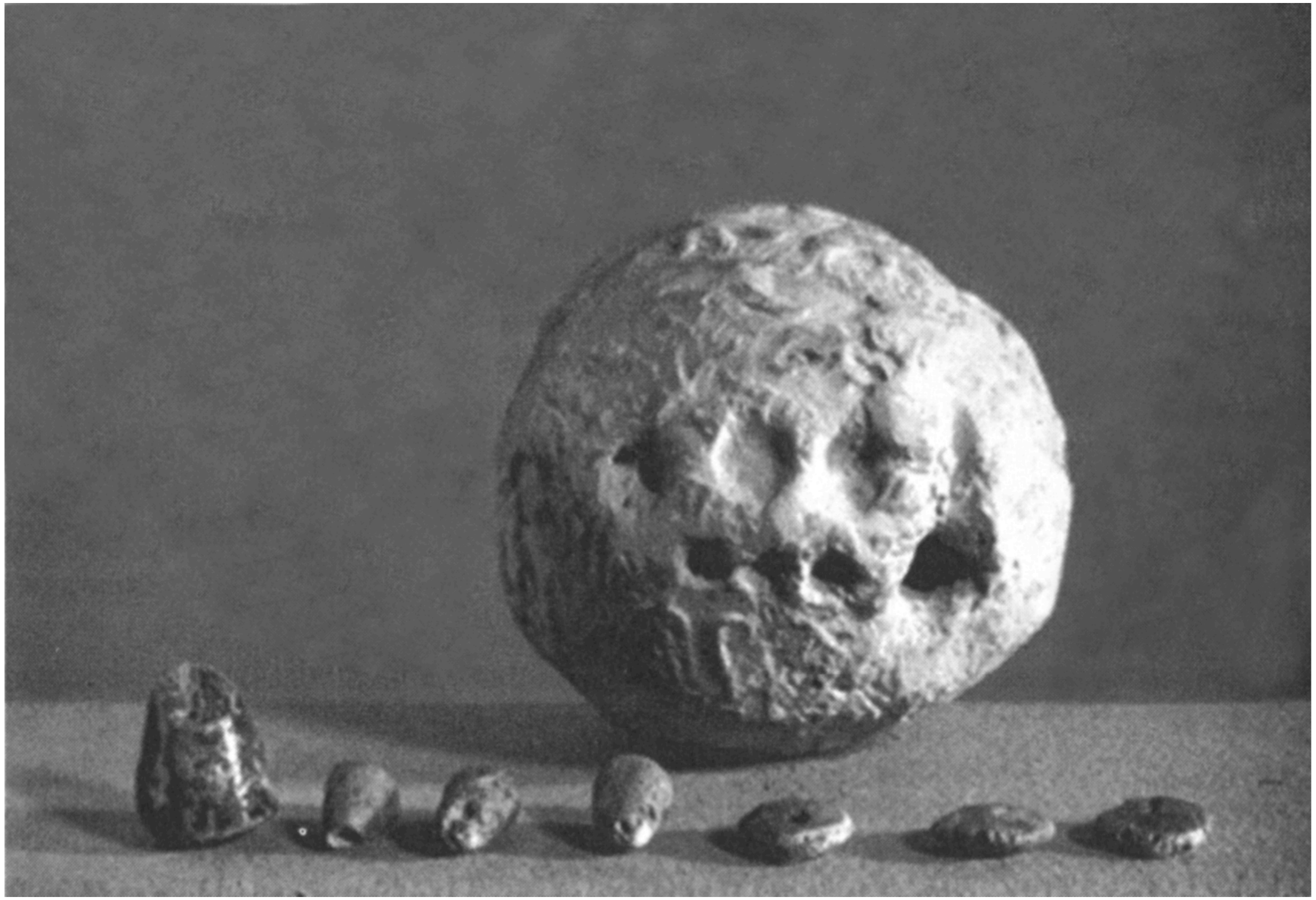


- Top-down – ‘fault tree analysis’ (in security, a threat tree)
- Work back from the bad outcome that we must avoid, to identify critical components

Example – nuclear safety

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

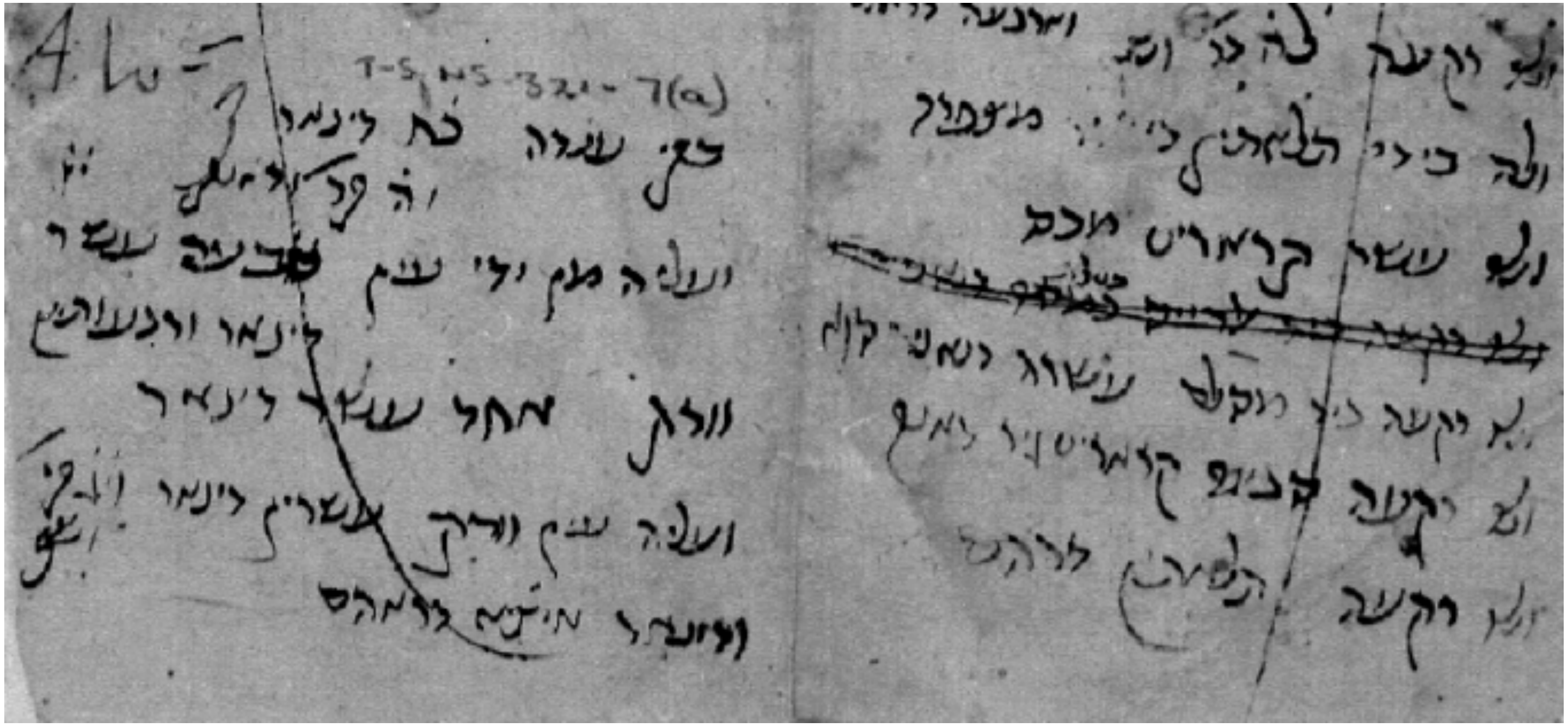
Bookkeeping, c. 3300 BC



Bookkeeping c. 1100 AD

- 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

From the Genizah Collection



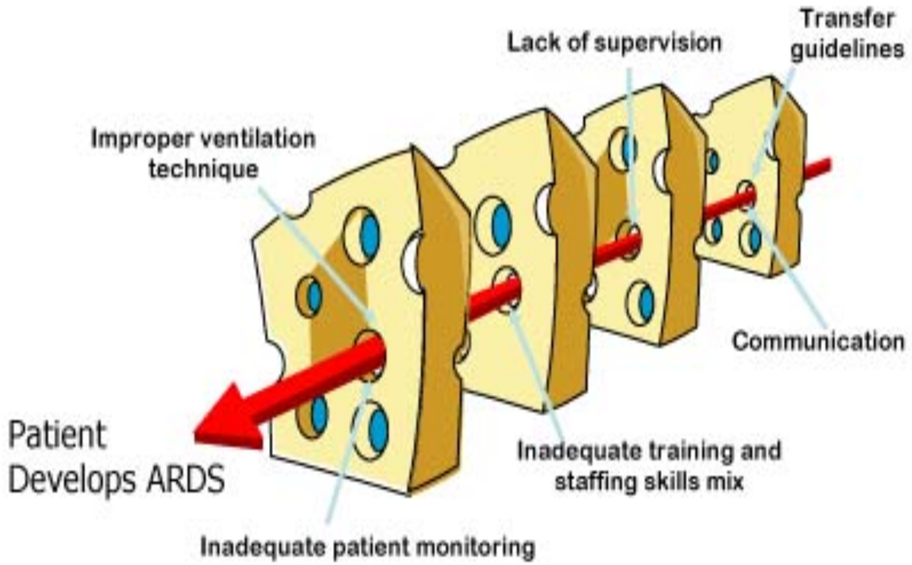
Separation of duties in practice

- 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?

Decoupling Policy, Mechanism

- Role-based Access Control adds extra indirection layer: ‘officer of the watch’, ‘branch accountant’, ‘charge nurse’
- Still need to devise a security policy!
- SELinux offers MLS with RBAC
- iPhones have something similar
- Red Hat uses it to separate services: a web server compromise doesn’t automatically get DNS

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!)

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...
- We'll now look at more detail

Predicting user behaviour

- It's so tempting to ignore 'user error'
- Banks routinely tell victims of fraud "Our systems are secure so it must be your fault"
- But regulators push back
- Most car crashes are user error; we still build cars with crumple zones
- Compare 1959, 2009 Chevrolets in video

Security and human behaviour

- Hierarchy of harms
  - Targeted attacks, such as spear phishing
  - Generic malware such as Zeus or Dridex
  - Bulk password compromise
  - Abuse of mechanisms provided as standard
- Each step down this hierarchy, the number of victims goes up an order or magnitude
- Let's start at the bottom where most harm is!

Abuse of standard mechanisms

- Just as a car crash is abuse of mechanism, so are most scams and abuses
- E.g., crook runs website offering flat to let, so you send off some money
- Or you get email telling you of a lottery win
- What can we do about cyber-bullying?
- Or doxxing?

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?

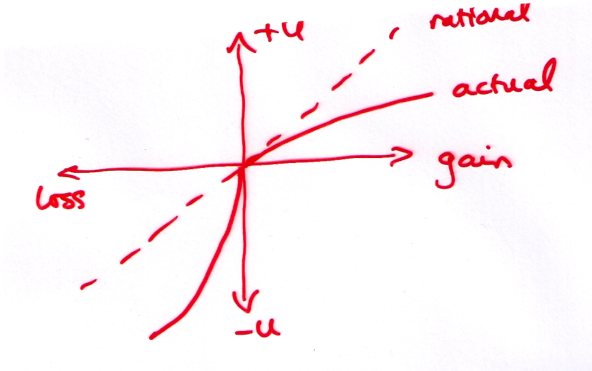
Phishing and social engineering

- Card thieves phone 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
- Think like a crook (and read Mitnick)!

Cognitive Factors

- Many errors arise from our highly adaptive mental processes
  - 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 have give way to skill
- But 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’

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!
- Kahneman and Tversky's 'prospect theory' explains such risk preferences systematically
- Risk misperception is exploited by cybercriminals (to remain inconspicuous) and terrorists (the opposite!)

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 lives will be saved"

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

  C: "400 will die"

  D: "with p =1/3, no-one will die, p=2/3, 600 will 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

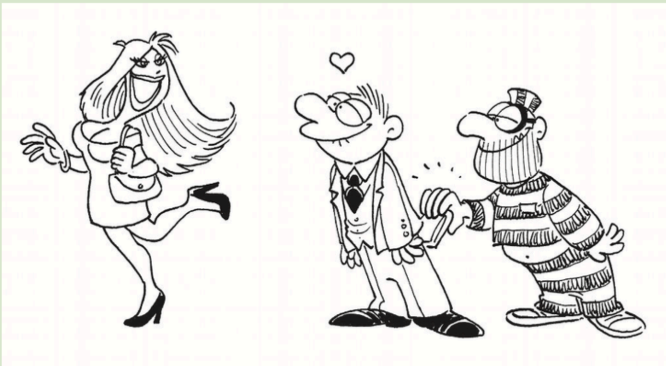
Social psychology

- Authority matters: Milgram showed over 60% of all subjects would torture a ‘student’
- The herd matters: Asch showed most people could deny obvious facts to please others
- Reciprocation is built-in: even monkeys do tit-for-tat! So give a gift, or appear to be in the mark’s in-group
- Cialdini’s “Influence – science and practice” (the marketer’s bible) discusses these, plus scarcity and need for consistency

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
- For more, see Modic and Lea’s taxonomy, and Stajano and Wilson on hustling

Scam mechanics



Distraction

While you are distracted by what retains your interest, hustlers can do anything to you and you won't notice.

Social Compliance

Society trains people not to question authority. Hustlers exploit this "suspension of suspiciousness" to manipulate you.

Scam mechanics II

Herd

Even suspicious marks will let their guard down when others next to them appear to share the same risks. Safety in numbers? Not if they're all against you.

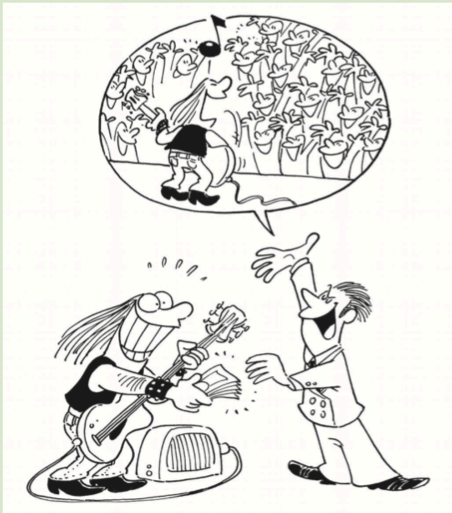
Dishonesty

Your larceny is what hooks you. Thereafter, anything illegal you do will be used against you by the fraudster.

Scam mechanics III

Kindness

People are fundamentally nice and willing to help. Hustlers shamelessly take advantage of it.



Need and Greed

Need and greed make you vulnerable. Once hustlers know what you want, they can easily manipulate you.

Scam mechanics IV

Time

When you are under time pressure to make an important choice, you use a different decision strategy. Hustlers steer you towards a strategy involving less reasoning.

- See Stajano and Wilson paper, or "The Real Hustle" videos

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!
- The kinds of security advice they're likely to follow depends on their main mental model

Affordances

- Why Johnny couldn't encrypt
- What actions do you make natural?
- Defaults really matter. Most people won't opt in, or opt out; they go with the flow
- So governments try to set socially optimal defaults (e.g. you must opt out of pensions)
- Why doesn't everyone set safer or more secure defaults online?

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?

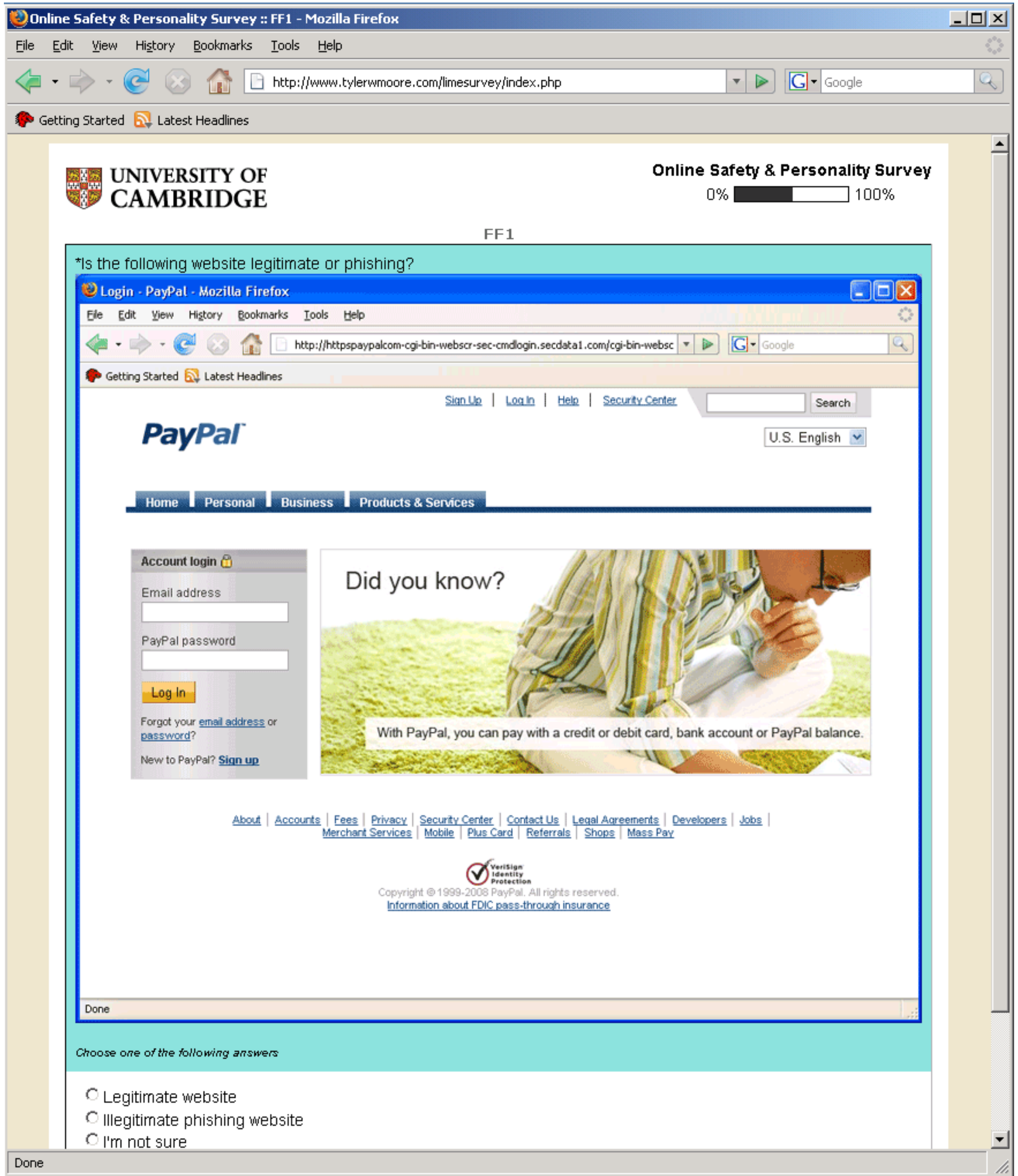
The compliance budget

- Violations of rules also matter: they're often an easier way of working, and sometimes necessary
- 'Blame and train' as an approach is suboptimal
- It's often rational to ignore warnings
- People will spend only so much time obeying rules, so choose the rules that matter
- The 'right' way of working should be easiest: look where people walk, and lay the path there

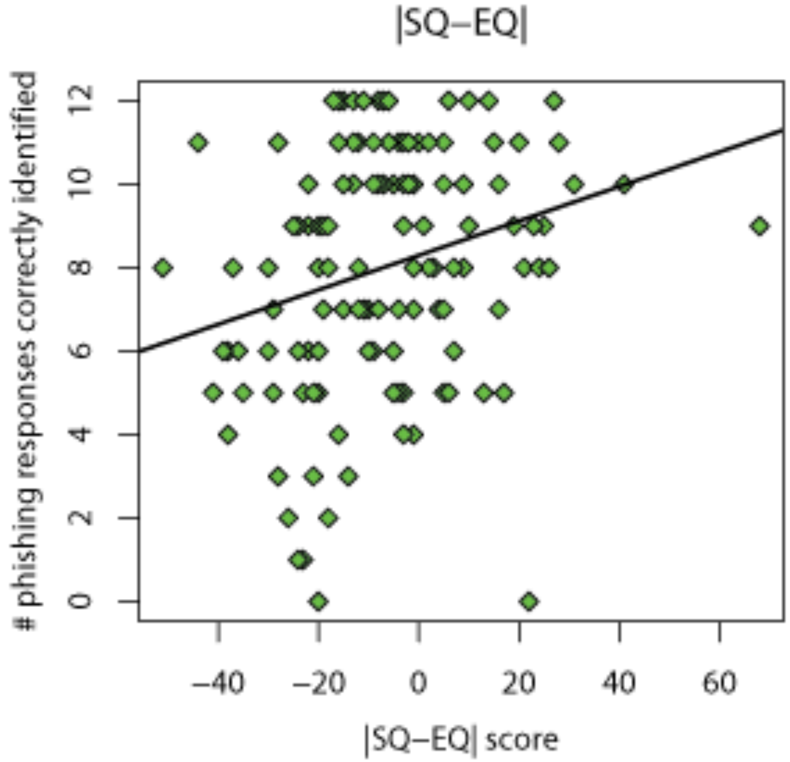
Where should the path be?

Differences between people

- Ability to perform certain tasks can very widely across subgroups of the population
- Risk thermostat – function of age, gender
- Also, banks tell customers 'parse URLs'
- Baron-Cohen: people can be sorted by SQ (systematizing) and EQ (empathising)
- Is this correlated with ability to detect phishing websites by understanding URLs?


Online Safety & Personality Survey :: FF1 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.tylerwmoore.com/limesurvey/index.php
Google
Getting Started Latest Headlines
UNIVERSITY OF CAMBRIDGE
Online Safety & Personality Survey
0% 100%
FF1
*Is the following website legitimate or phishing?
Login - PayPal - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://httpspaypalcom-cgi-bin-webscr-sec-cmdlogin.secdata1.com/cgi-bin-websc
Google
Getting Started Latest Headlines
Sign Up | Log In | Help | Security Center
Search
PayPal
U.S. English
Home Personal Business Products & Services
Account login
Email address
PayPal password
Log In
Forgot your email address or password?
New to PayPal? Sign up
Did you know?
With PayPal, you can pay with a credit or debit card, bank account or PayPal balance.
About | Accounts | Fees | Privacy | Security Center | Contact Us | Legal Agreements | Developers | Jobs |
Merchant Services | Mobile | Plus Card | Referrals | Shops | Mass Pay
VeriSign Identity Protection
Copyright © 1999-2008 PayPal. All rights reserved.
Information about FDIC pass-through insurance
Done
Choose one of the following answers
Legitimate website
Illegitimate phishing website
I'm not sure
Done

Results



- Ability to detect phishing is correlated with SQ-EQ
- It is (independently) correlated with gender
- The ‘gender HCI’ issue applies to security too

Errors

- Read up the psychology that underlies errors!
- 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

Errors (2)

- Training and practice help – skill is more reliable than knowledge! 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

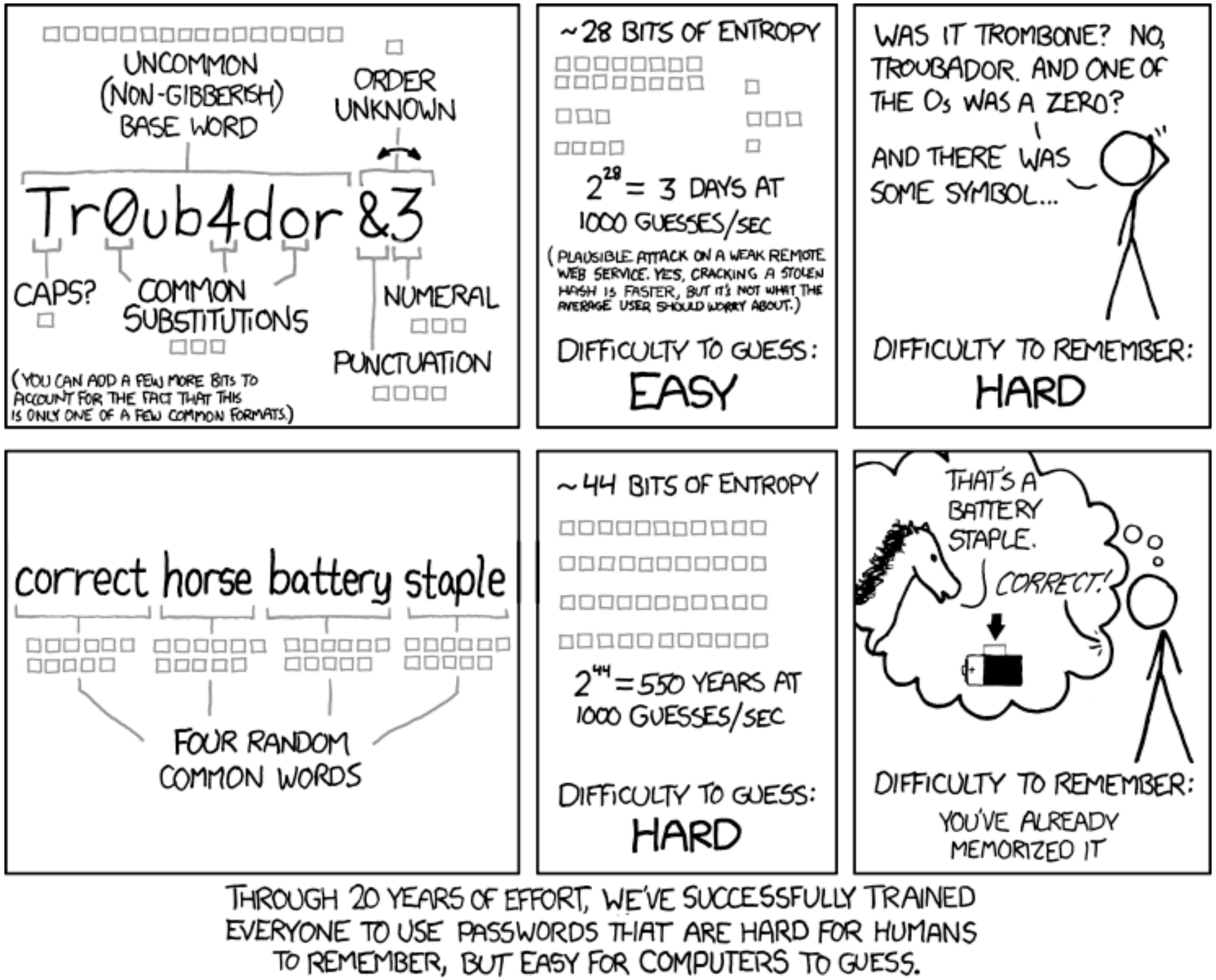
Passwords

- 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
- Everyone wants recovery questions too
- Many firms train customers in unsafe behaviour such as clicking on links
- Much ‘training’ amounts to victim blaming
- Two-factor authentication? Ubuntu hack

Incremental guessing

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

Matt Honan hack

- Gmail password reset sends a message to the backup email and prints part of it (was Matt's apple @me.com account)
- Apple password reset: billing address plus last 4 digits of credit card
- Amazon: provide any credit card number (add a new one; then you see last 4 digits of others)
- 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 then 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?

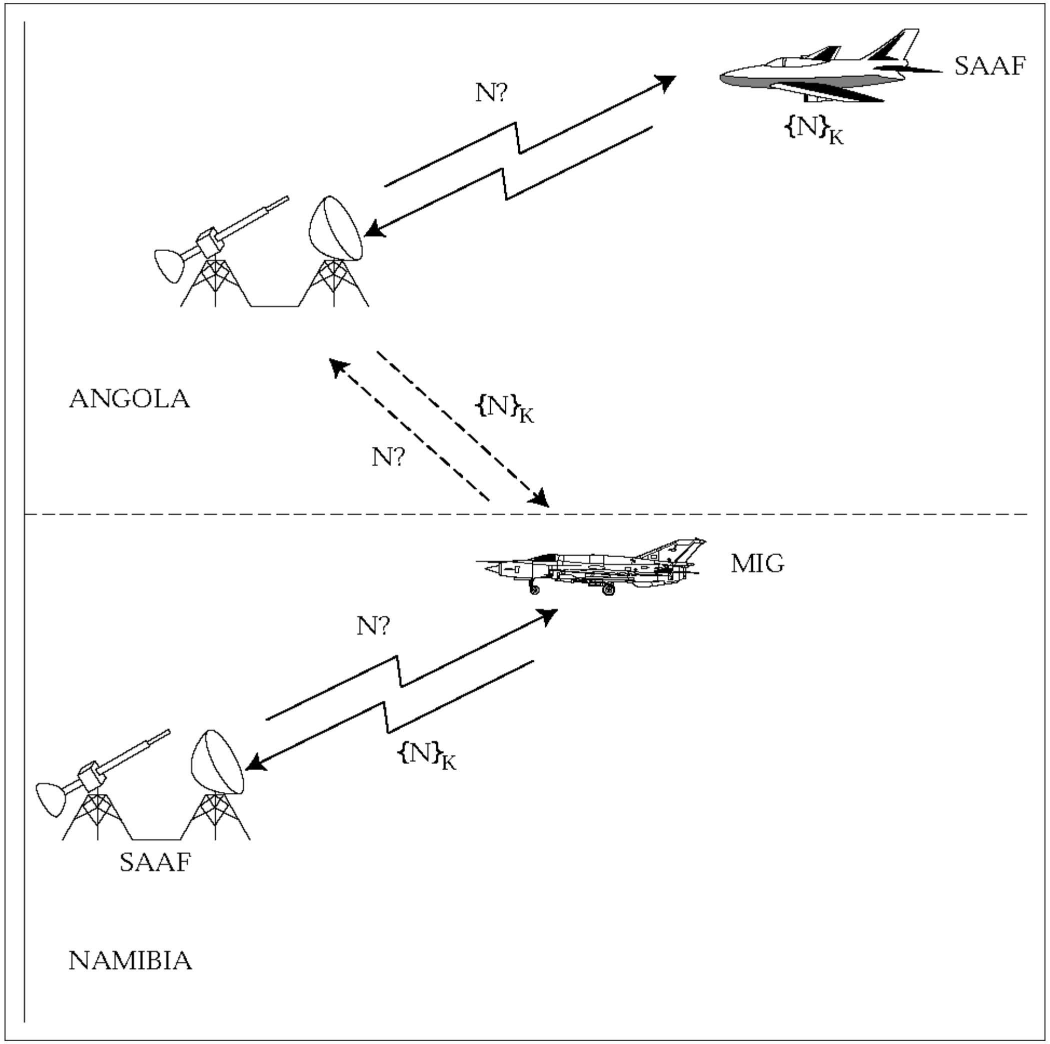
  $F \rightarrow B: N$

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

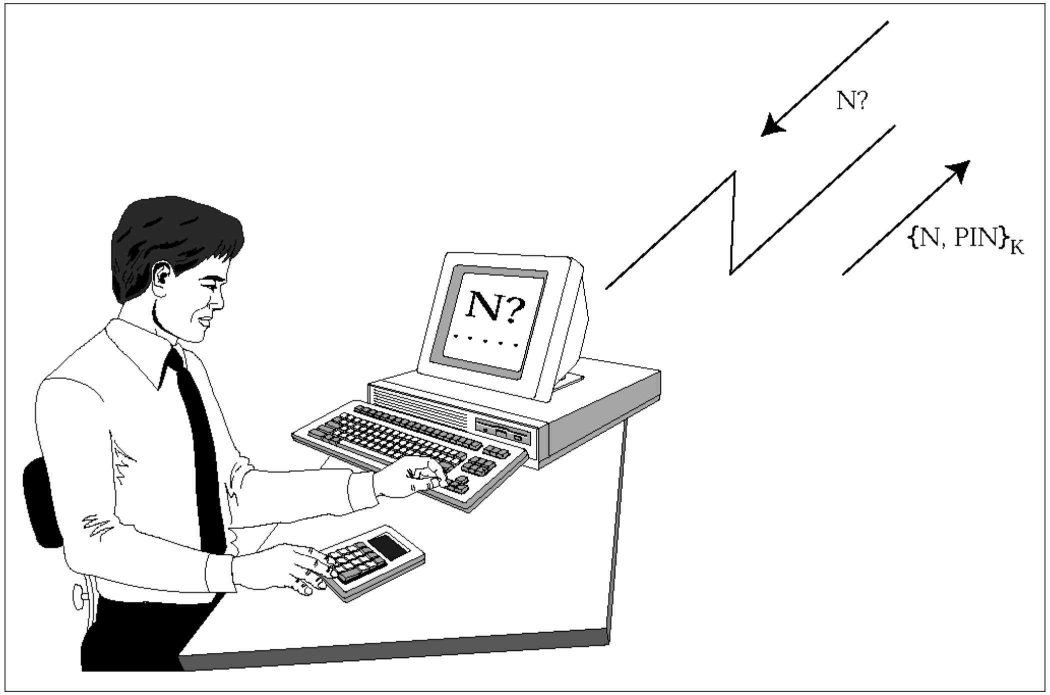
  $\mathit{F} \rightarrow B: \{N\}_K$

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

IFF (2)



Two-factor authentication



$S \rightarrow U$: N

$U \rightarrow P$: N, PIN

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

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)

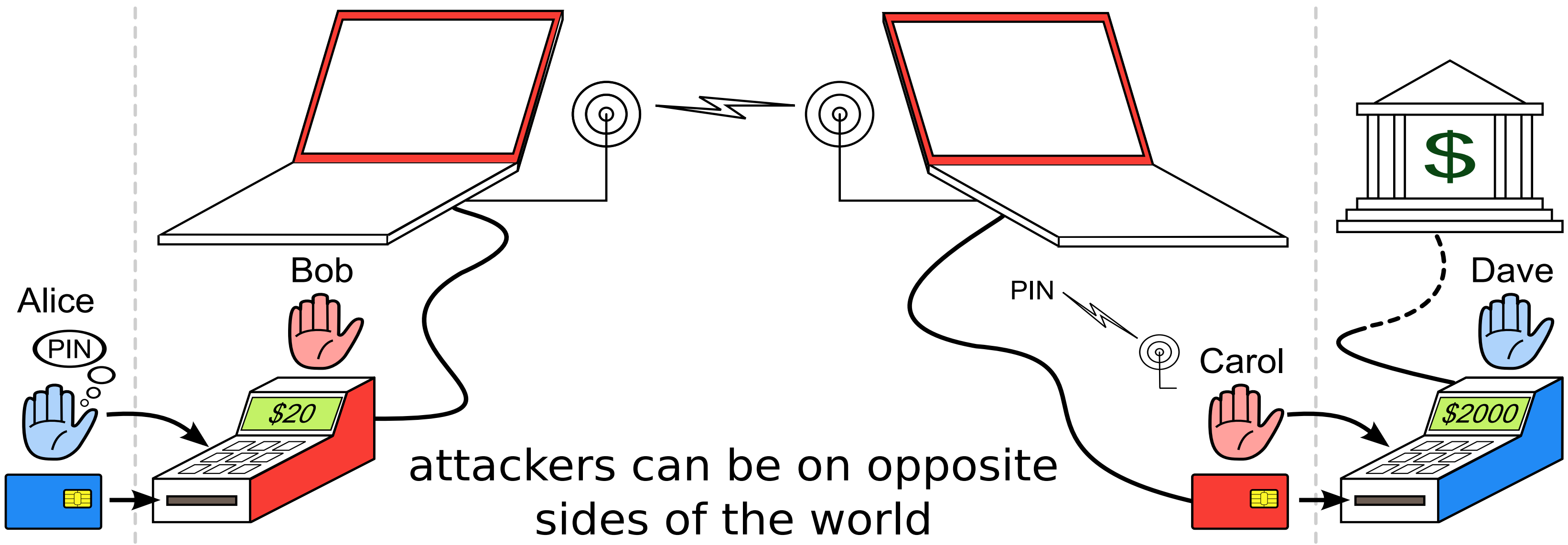
- $C \rightarrow M$: $\mathrm{sig}_B\{C, \mathrm{card\_data}\}$
- $M \rightarrow C$: N, date, Amt, PIN (if PIN used)
- $C \rightarrow M$: $\{N, \mathrm{date}, \mathrm{Amt}, \mathrm{trans\_data}\}_{KCB}$
- $M \rightarrow B$: $\{\{N, \mathrm{date}, \mathrm{Amt}, \mathrm{trans\_data}\}_{KCB}, \mathrm{trans\_data}\}_{KMB}$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

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 (2007)



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: if I have access to your card I can precompute an authenticator for Amt, d!

Public key crypto revision

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

Public key crypto revision (2)

- Anthony sends a box with a message 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 implementation:

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

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

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

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

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

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

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

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 → S: C, C#, NC

  S → C: S, S#, NS, CS

  C → S: $\{K0\}_S$

  C → S: crypto hash of K0, NC, NS, etc

  S → C: crypto hash of K0, NS, NC, etc
- This has been proved to be secure (Larry Paulson, 1999)
- So what could possibly go wrong?

Beyond the protocol abstraction

- You can't just use raw Diffie-Hellman as you saw it in Discrete Maths 1a!
- The function $y = g^x$ is a homomorphism; if $x3=x1+x2$ then $y3 = g^{x3} = g^{(x1+x2)} = g^{x1}g^{x2} = y1y2$
- So you can do fancy stuff like threshold decryption and signature (exercise: figure out how)
- You can also do fancy attacks!
- So real public key encryption (and signature) functions need careful design (see Security II)

Beyond the protocol abstraction (2)

- The same holds for shared-key encryption!
- Use a strong cipher: triple-DES or AES
- Choose the right mode of operation: do you need encryption? Authentication? Both? Do you need it randomized, or do you need the same answer each time you encrypt X?
- Again, for the subtle details see Security II
- Crypto API defaults are often weak!

Beyond the protocol abstraction (3)

- 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)
- Again, later courses have many more details

Beyond the protocol abstraction (4)

- Governments may 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?

Beyond the protocol abstraction (5)

- ‘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 :

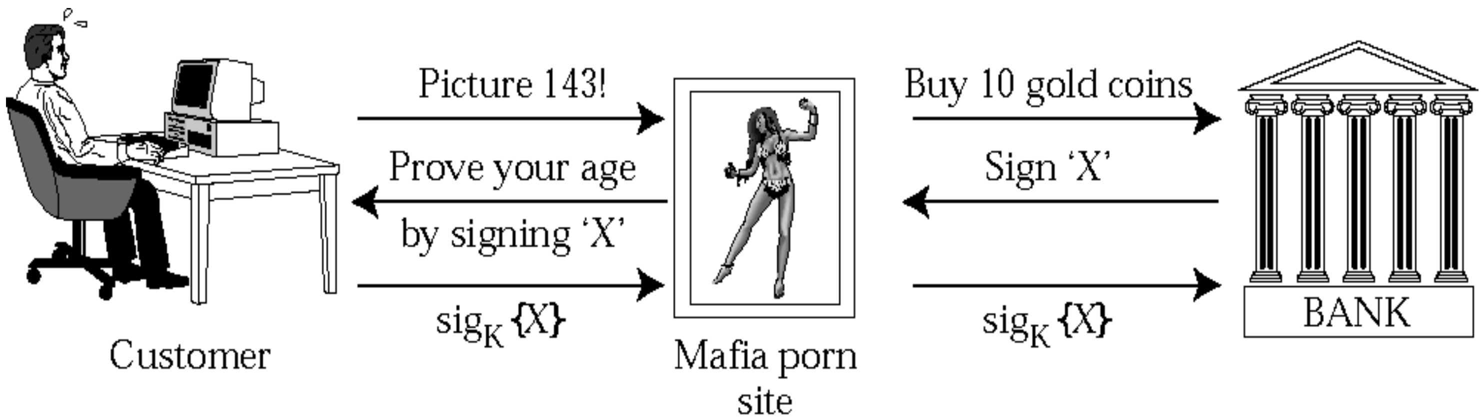
  $C \rightarrow M$: order

  $M \rightarrow C$: 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
- How else do they persist in systems that are very extensively tested?

Syntactic bugs

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

Apple's 'goto fail'

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
        OSStatus        err;
        ...

        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        ...

fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

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
- Race to get new certs: 50% in first month
- White House took ‘equity issue’ from NSA

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?”

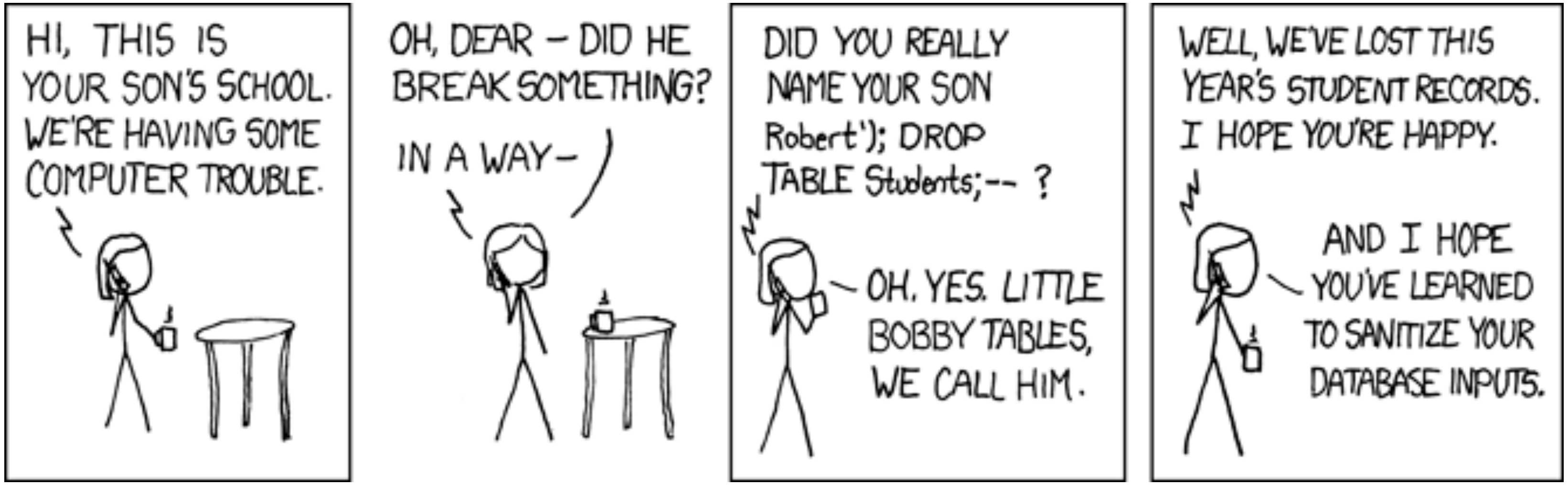
Code injection

- 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
  - See Howard and leBlanc on MS standards 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?)
- Coverity and other such tools (later)

The ‘Software Crisis’

- Software continues to lag far behind the hardware’s potential!
- Many large projects are late, over budget, dysfunctional, or abandoned (LAS, CAPSA, NPfIT, DWP, Addenbrookes …)
- Some failures cost lives (Therac 25) or billions (Ariane 5, NPfIT)
- Some expensive scares (Y2K, Pentium)
- Some combine the above (LAS)

The London Ambulance Service disaster

- Widely cited example of project failure because it was thoroughly documented (and the pattern has been frequently repeated)
- 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

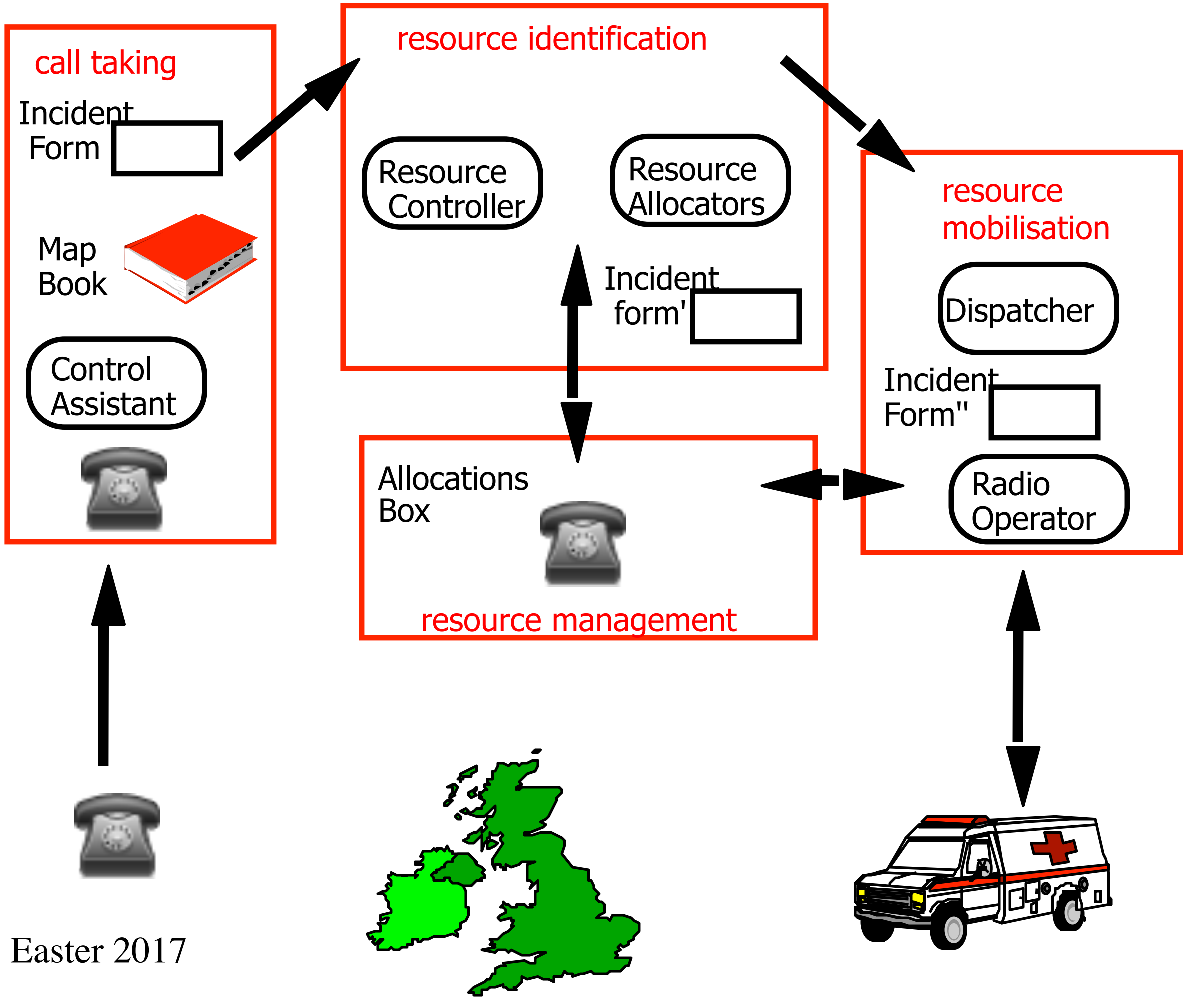
Original dispatch system

- 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

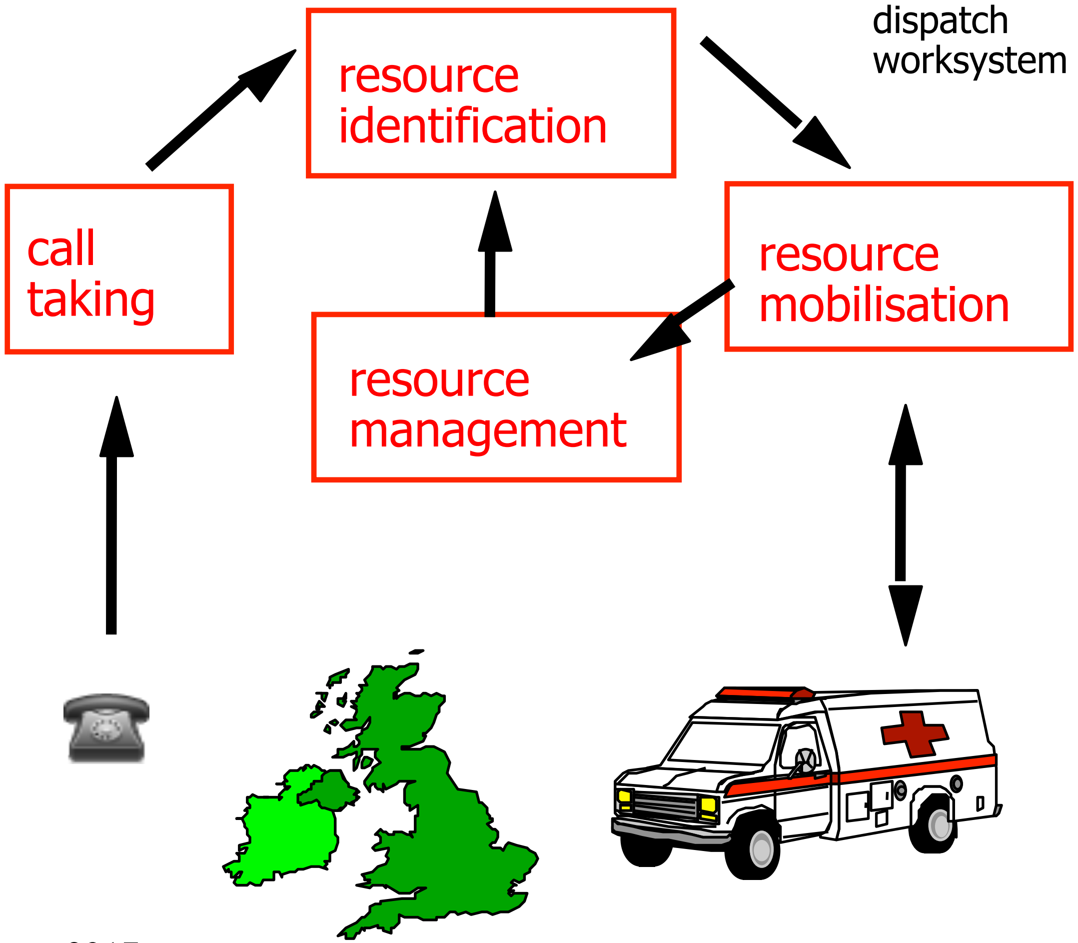
Project context

- 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

The manual implementation



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

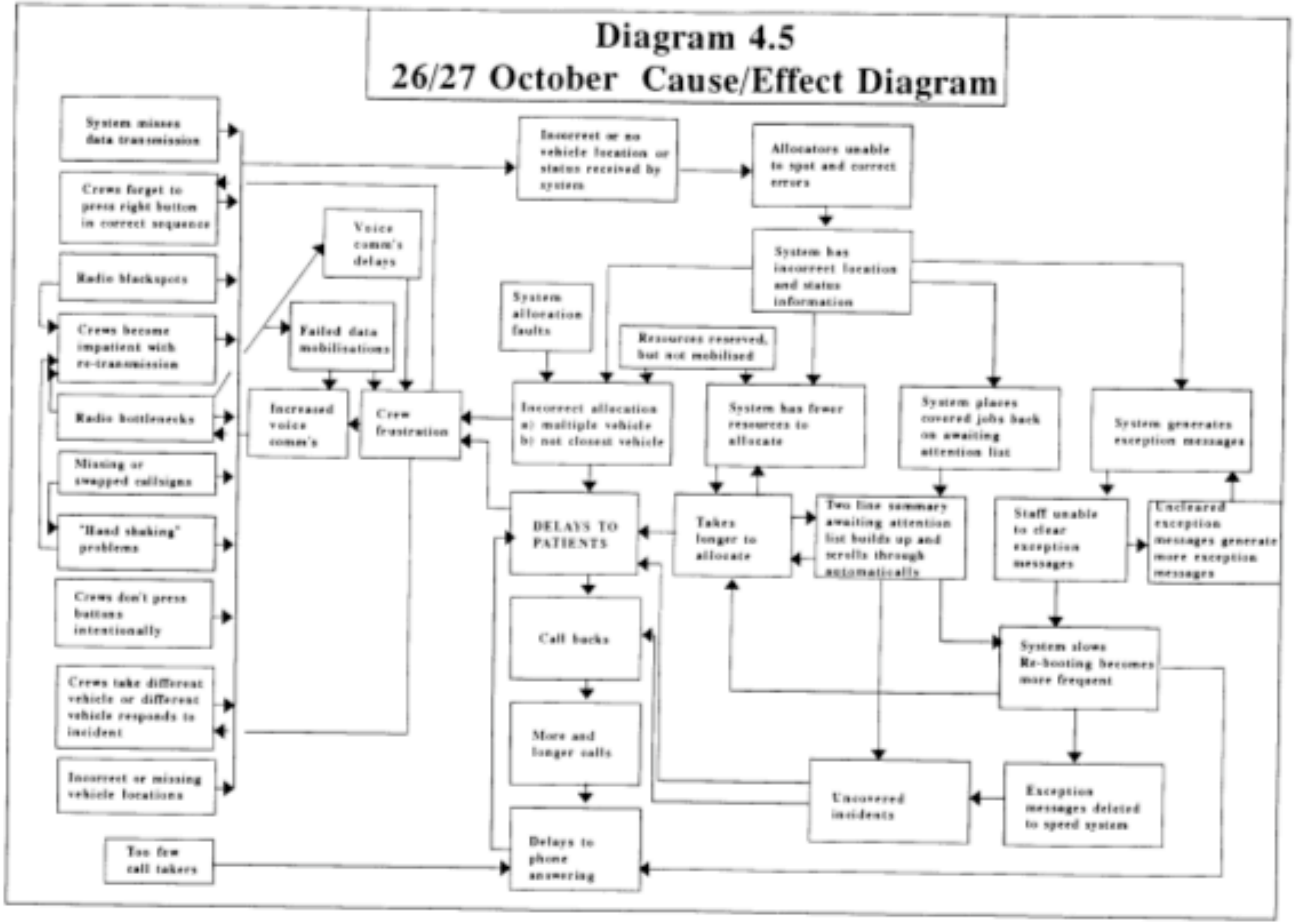


Diagram 4.5
26/27 October Cause/Effect Diagram
System misses data transmission
Crews forget to press right button in correct sequence
Radio blackspots
Crews become impatient with re-transmission
Radio bottlenecks
Missing or swapped callsigns
"Hand shaking" problems
Crews don't press buttons intentionally
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

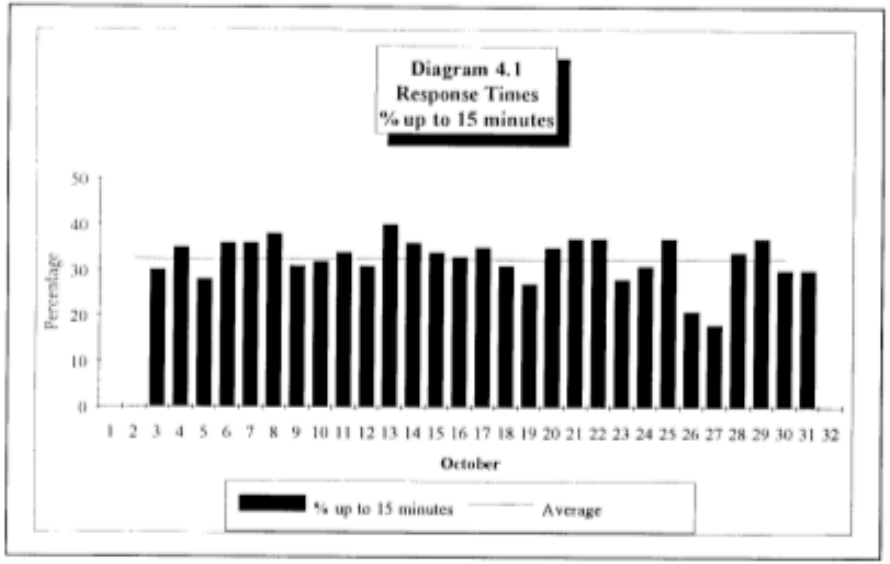


Diagram 4.1
Response Times
% up to 15 minutes
Percentage
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
October
% up to 15 minutes
Average

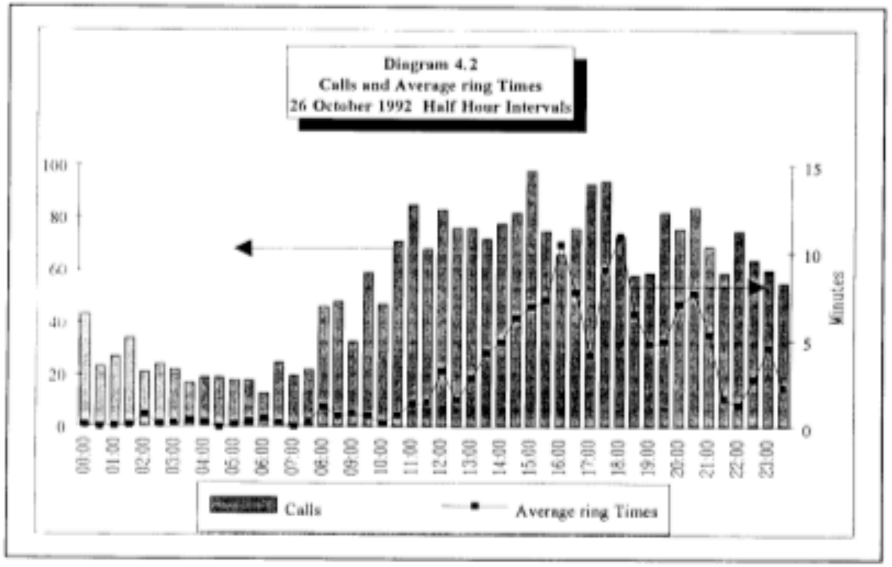


Diagram 4.2
Calls and Average ring Times
26 October 1992 Half Hour Intervals
100
80
60
40
20
0
15
10
5
0
Minutes
00:00 01:00 02:00 03:00 04:00 05:00 06:00 07:00 08:00 09:00 10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00 22:00 23:00
Calls
Average ring Times

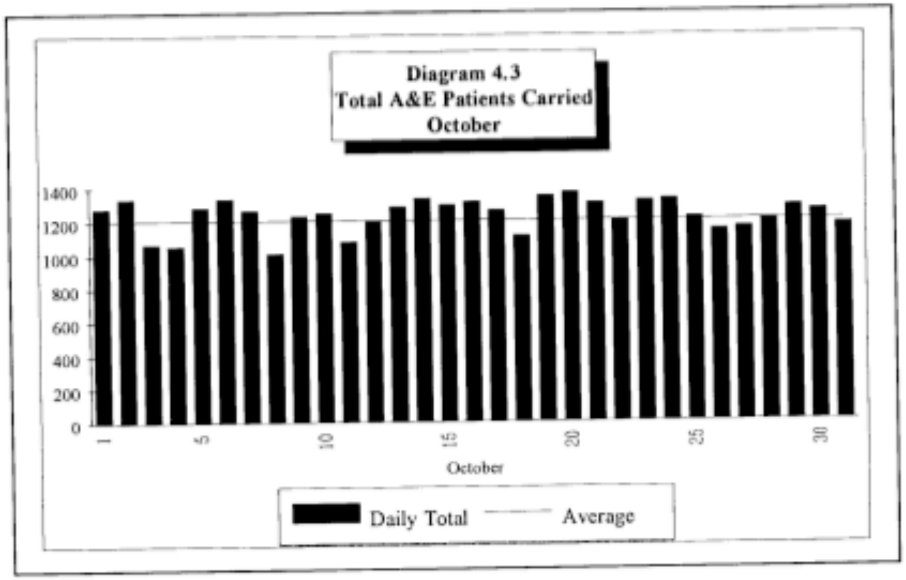


Diagram 4.3
Total A&E Patients Carried
October
1400
1200
1000
800
600
400
200
0
1
5
10
15
20
25
30
October
Daily Total
Average

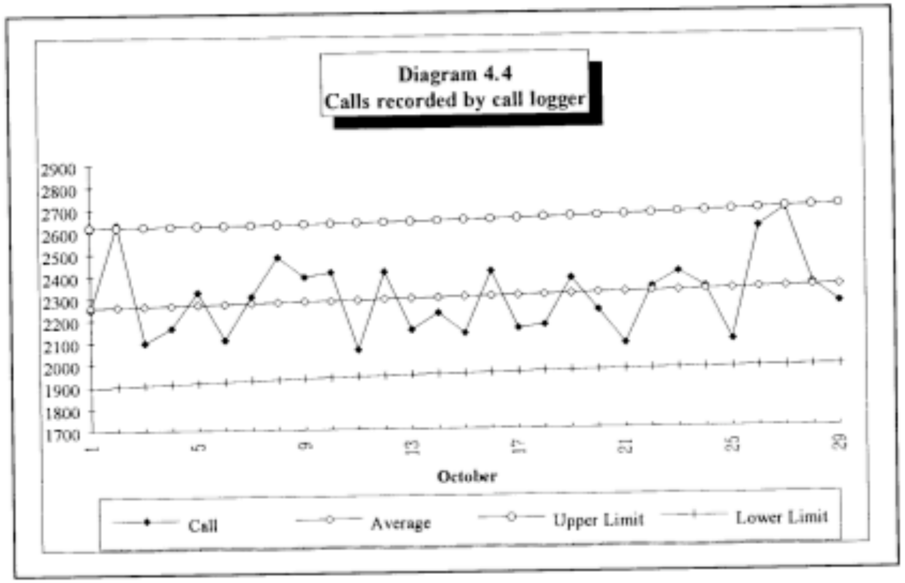


Diagram 4.4
Calls recorded by call logger
2900
2800
2700
2600
2500
2400
2300
2200
2100
2000
1900
1800
1700
1
5
9
13
17
21
25
29
October
Call
Average
Upper Limit
Lower Limit

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

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 – 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! Problems …
- General: big systems take 7 years not 3
- They hoped 'agile' development would fix it …
- Depended on real-time feed of tax data from HMRC, which in turn depended on firms
- Descended into chaos; NAO report

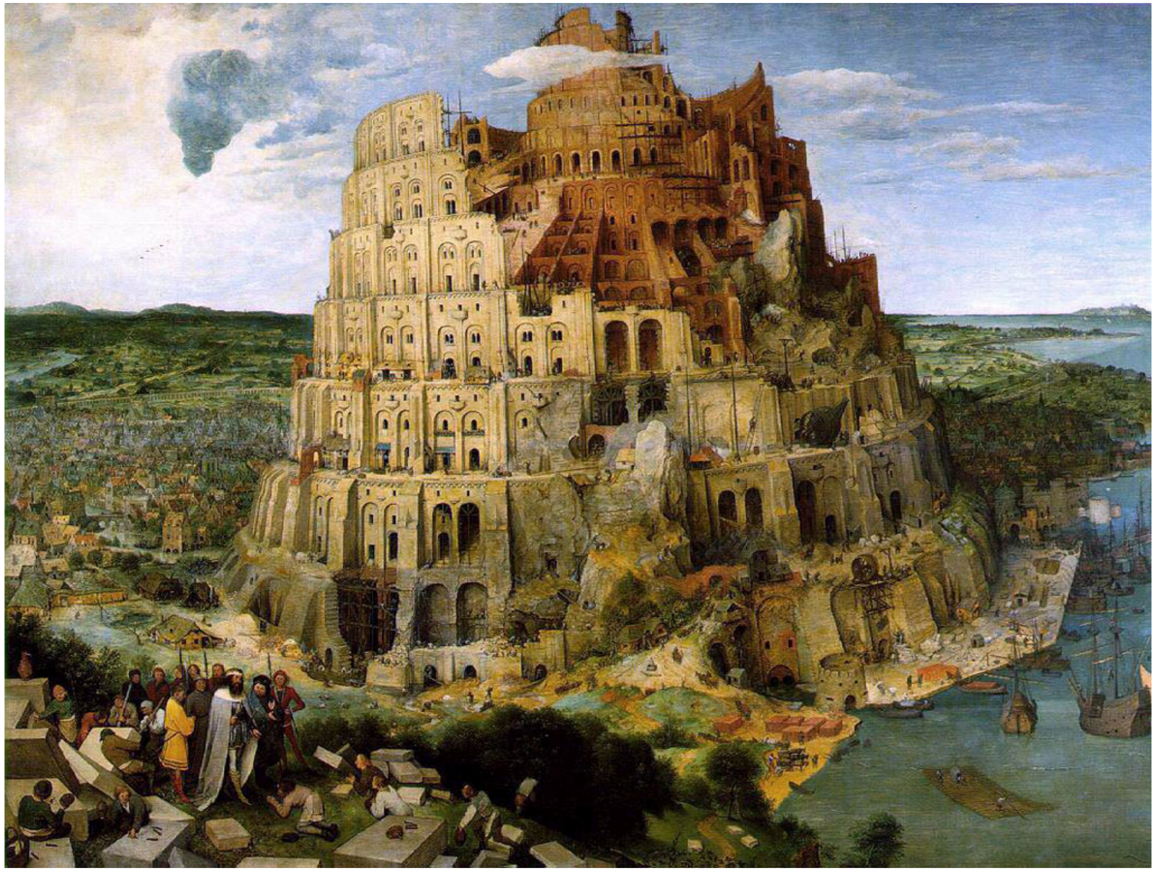
Next – ‘smart meters’

- 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
- March 2010: experts said we just can’t change 47m meters in 6 years. So excluded from spec
- Coalition government: wanted deployment by 2015 election! Planned to build central system Mar–Sep 2013 (then: Sep 2014 …)
- Spec still fluid, tech getting obsolete, despair …

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 $O(2^n)$
  - …
  - With complex socio-technical systems, we can't predict reactions to new functionality
- Most failures of really large systems due to wrong, changing, or contested requirements

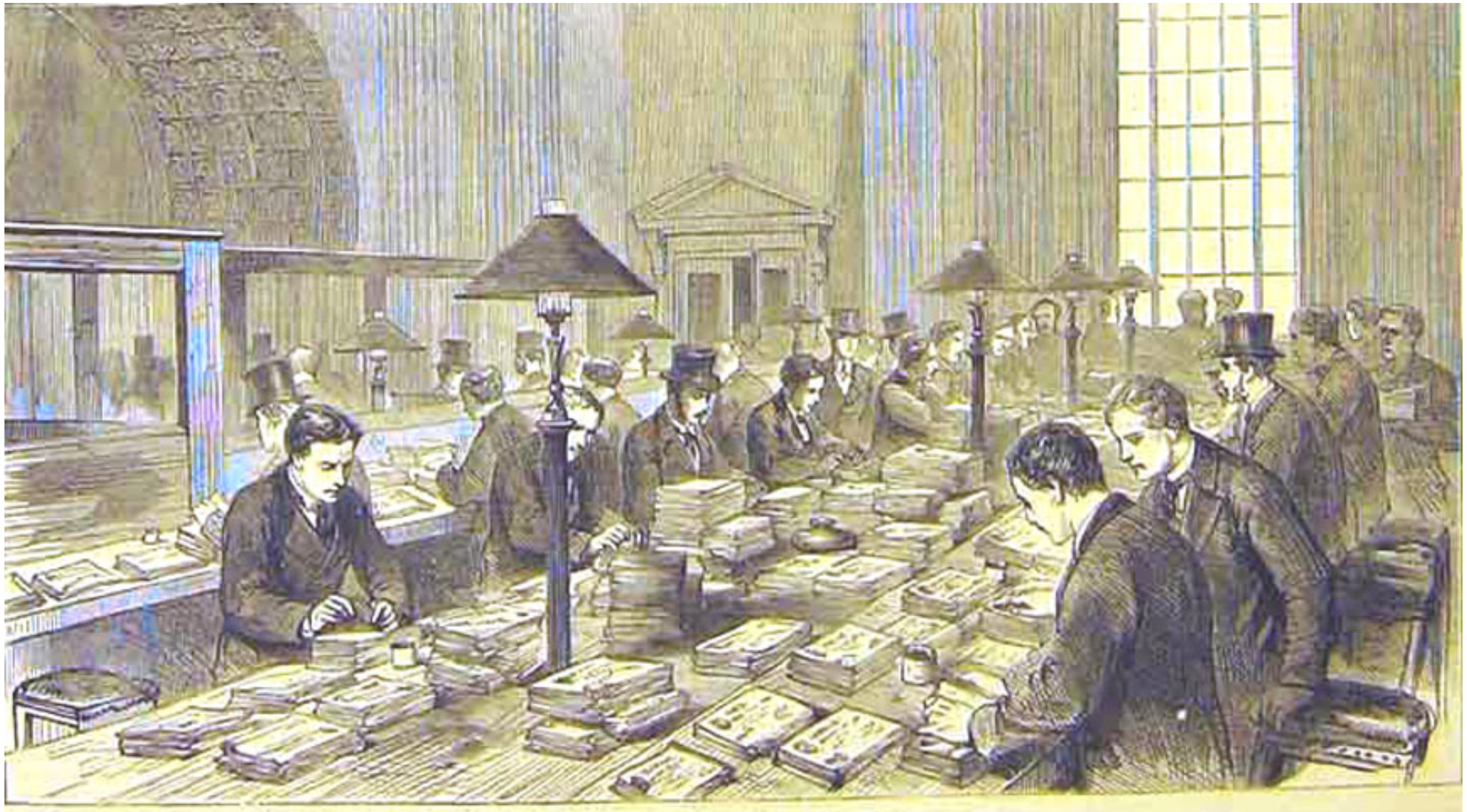
Project failure, c. 1500 BC



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, 1870 – Bank of England



THE ACCOUNTANTS' BANK NOTE OFFICE

Complexity 1876 – Dun, Barlow & Co

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 non-repeating task with continuous learning
  - pleasure of working with a tractable medium, 'pure thought stuff'
  - complete flexibility – you can base the output on the inputs in any way you can imagine
  - satisfaction of making stuff that's useful to others

How is software different? (2)

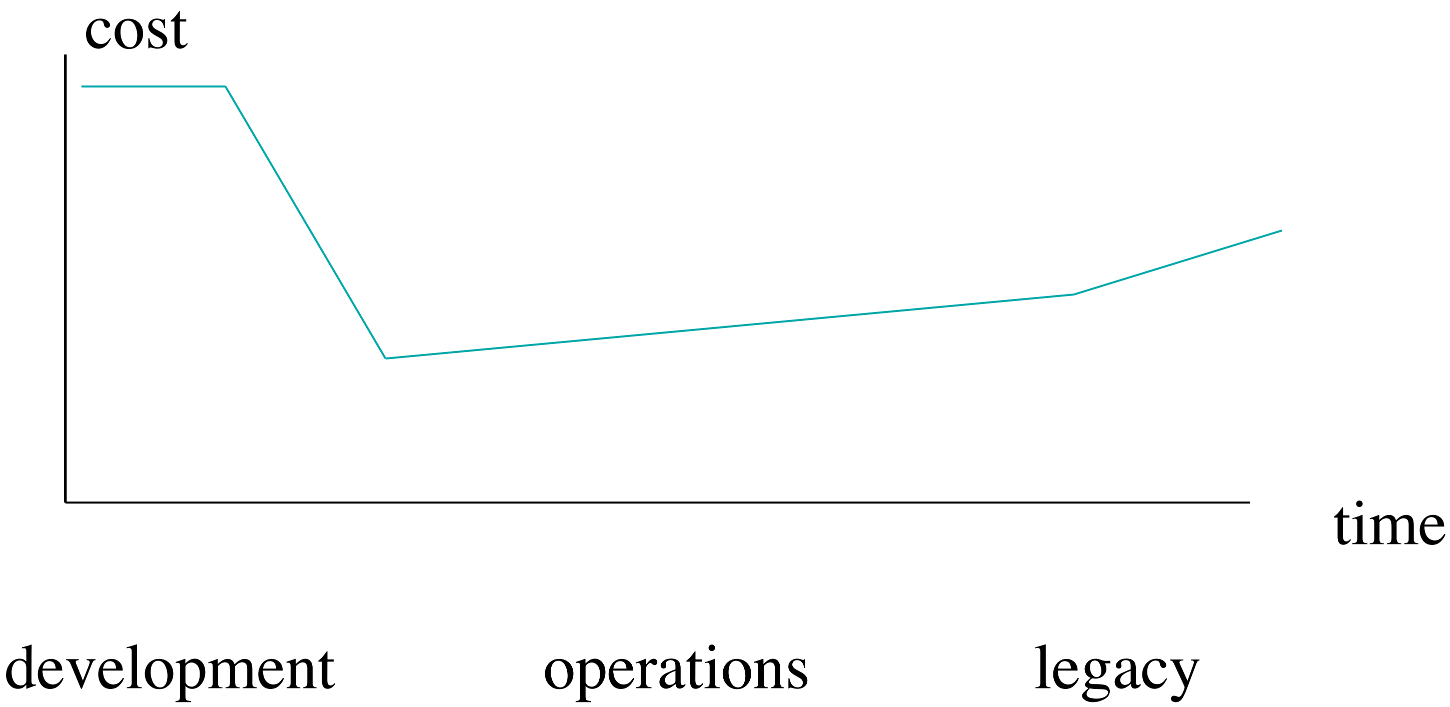
- Large systems become qualitatively more complex, unlike big ships or long bridges
- The tractability of software leads customers to demand 'flexibility' and frequent changes
- This makes systems more complex to use over time as 'features' accumulate, and interactions have odd effects
- The structure 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 nasty
  - Consumers buy on sticker price, businesses on total cost of ownership
  - vendors use lock-in tactics
  - complex outsourcing
- First let's consider the simple (1950s) case of a company that develops and maintains software entirely for its own use

Cost of software

- Initial development cost (10%)
- Continuing maintenance cost (90%)



What does code cost?

- First IBM measures (60s)
  - 1.5 KLOC/developer year (operating system)
  - 5 KLOC/dev yr (compiler)
  - 10 KLOC/dev yr (app)
- AT&T measures
  - 0.6 KLOC/dev yr (compiler)
  - 2.2 KLOC/dev yr (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
- It's also worth putting extra effort into getting the specification right, as it more than pays for itself by reducing the time spent on 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 x 4 months
  - Suppose the design work takes an extra month. So we have 2 months to do 9 dev mth 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 engineering economics

- Boehm, 1981 (empirical studies after Brooks)
  – Cost-optimum schedule time to first shipment $T=2.5(\text{dev-months})^{1/3}$
  – With more time, cost rises slowly
  – With less time, it rises sharply
  – Hardly any projects succeed in less than 3/4 T
- Similar and supporting studies – see van Vliet, chapter 7, on software cost metrics
- Yet some projects fail despite huge resources!

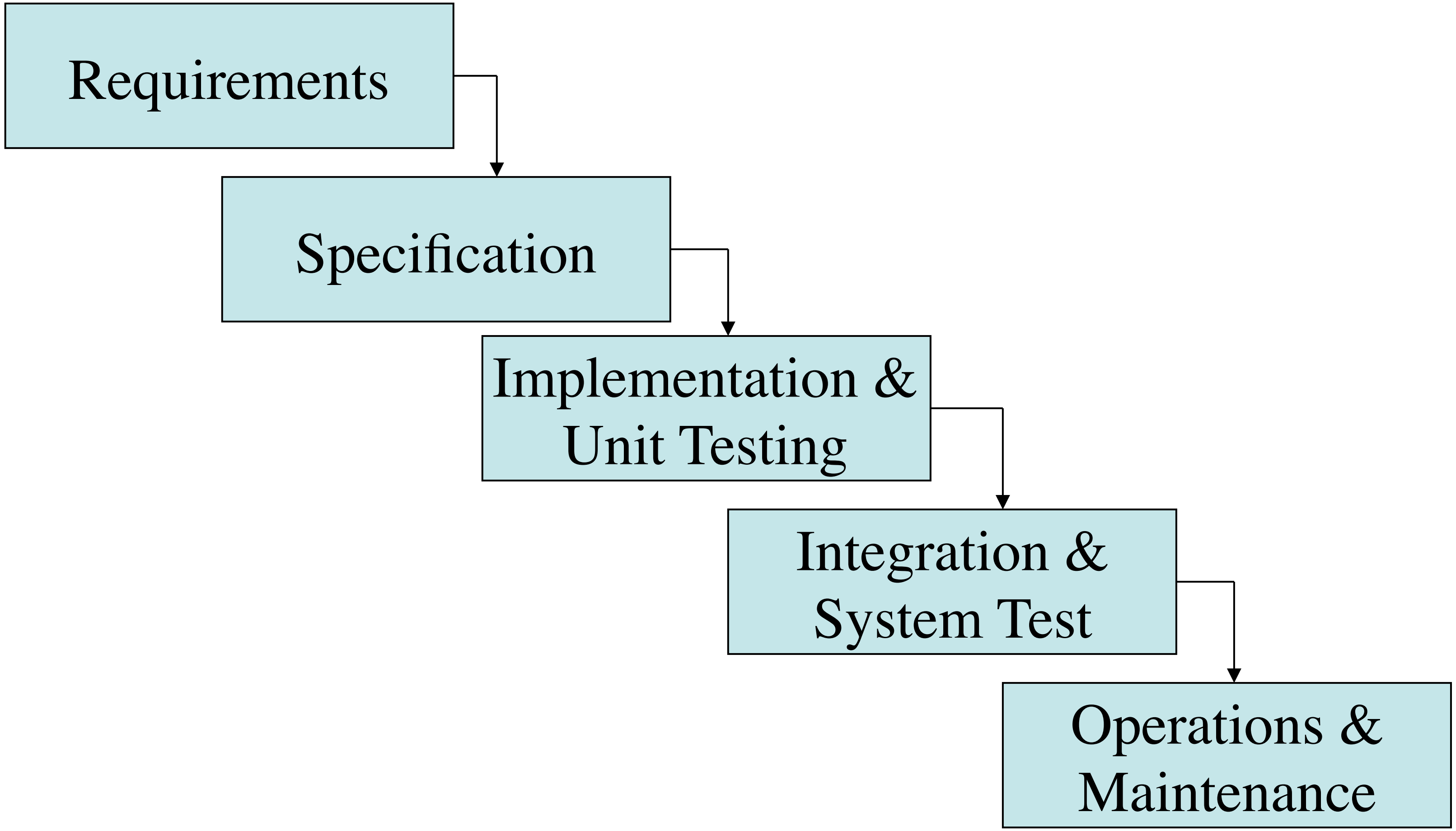
The software project ‘Tar Pit’

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

Structured design

- People realised fairly quickly only practical way to build large complex programs is to chop them up into modules
- Sometimes task division seems straightforward (bank = tellers, ATMs, dealers, …)
- Sometimes it isn't
- Sometimes it just seems to be!
- Quite a number of methodologies have been developed (SSDM, Jackson, Yourdon, UML…)

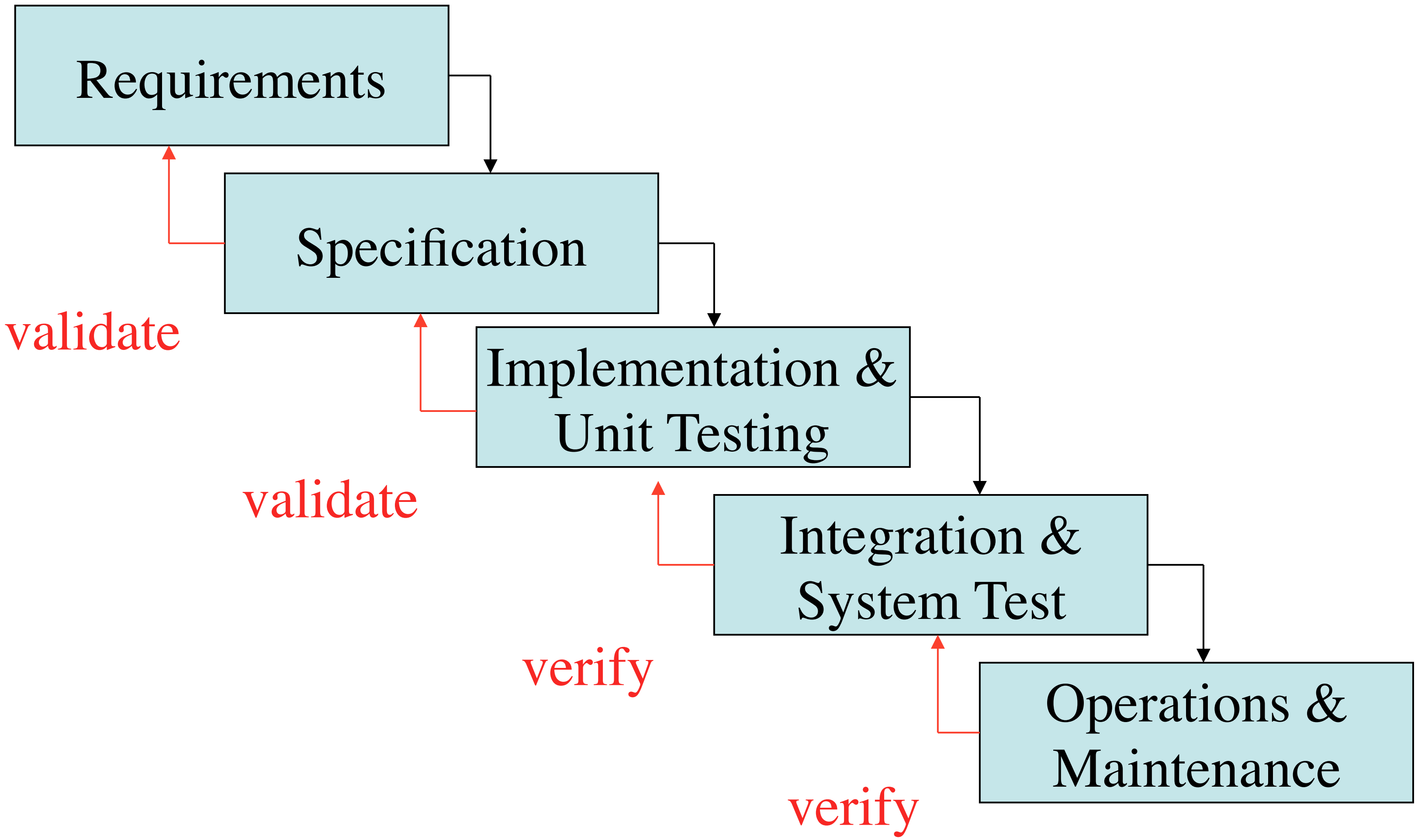
The waterfall model



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 of what the user wants
- Warning – when Winton Royce published this in 1970 he cautioned against naïve use
- But it become a US DoD standard …

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

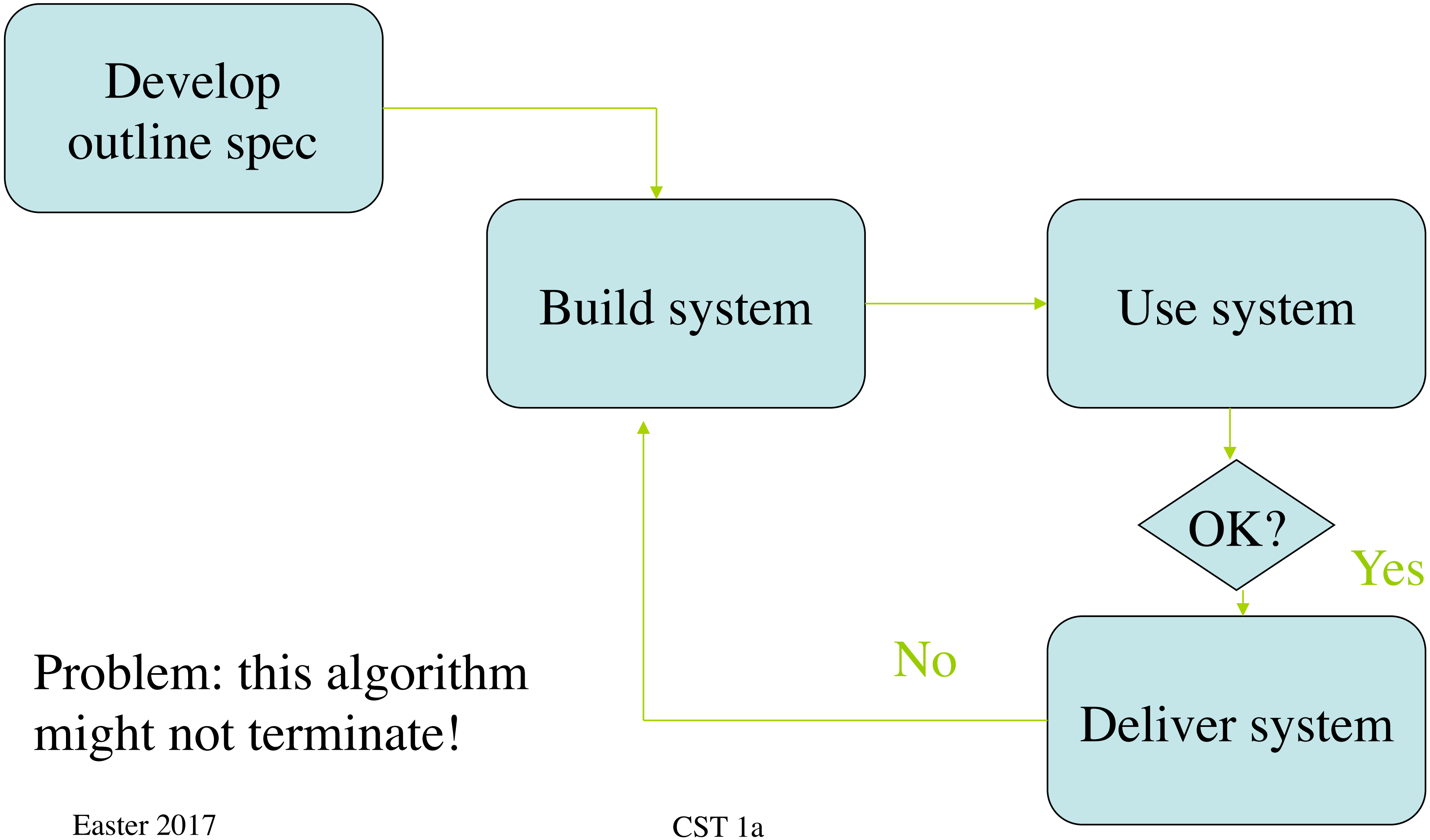
Waterfall – advantages

- 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

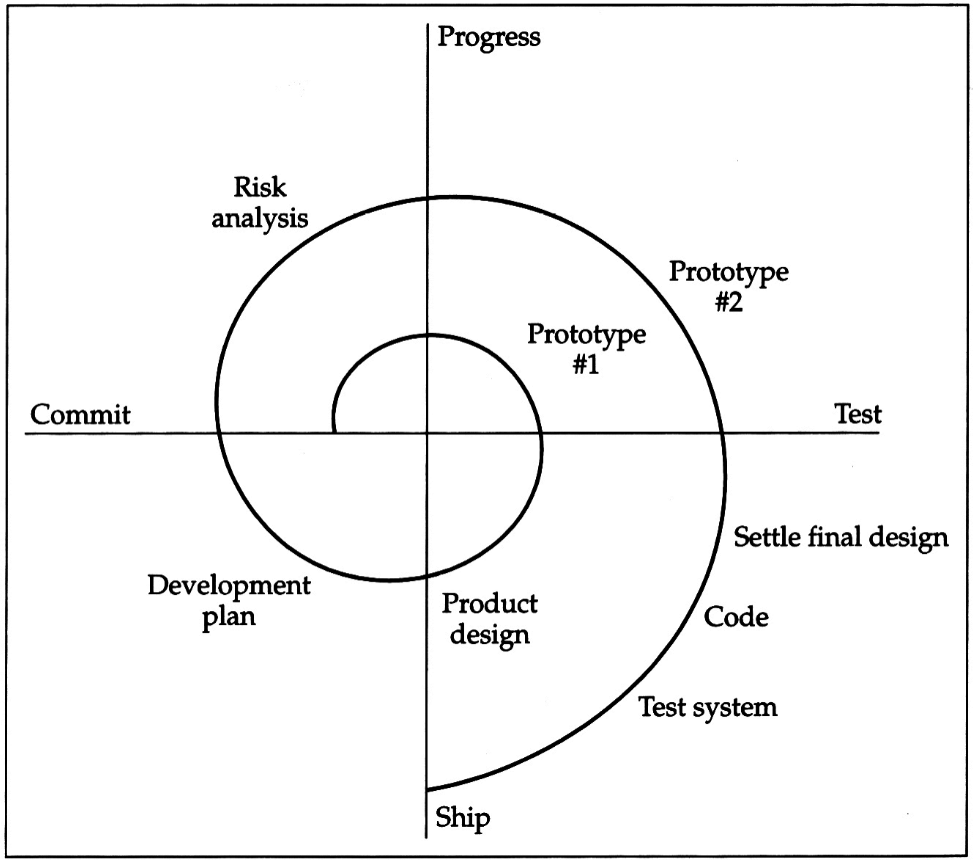
- Iteration can be critical in the development process:
  - requirements not yet understood by developers
  - or not yet understood by the customer
  - the technology is changing
  - the environment (legal, competitive) is changing
- The attainable quality improvement may be unimportant over the system 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)

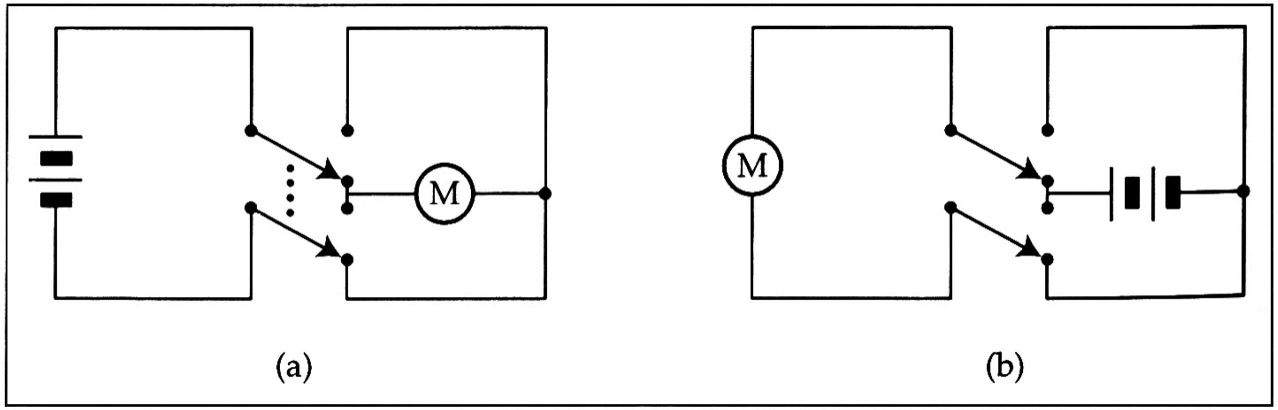
- A modern integrated development environment has several components
  - Code and documentation version control (git)
  - Code review (gerrit)
  - Automated build (make)
  - Continuous integration (Jenkins)
- The guest lecture will discuss the effect this tech has had on the industry
- Think how you'll set up your group project!

Assurance of critical software

- 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

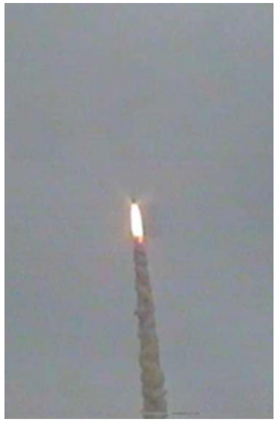
Tacoma Narrows, Nov 7 1940

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 software

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 → 20^{o} angle of attack → booster separation

Multi-factor failure

- Many safety-critical systems are also real-time systems used in monitoring or control
- It would be great to have no arithmetic or bounds errors in the first place
- Exception handling is often tricky (and it would be great to have no core dumps ever)
- But criticality of timing makes many simple verification techniques inadequate
- Testing can also be really hard

Emergent properties

- In general safety is a system property and has to be dealt with holistically
- The same goes for security, and real-time performance too
- A very common error is not getting the scope right
- For example, designers don't consider human factors such as usability and training

The Therac accidents

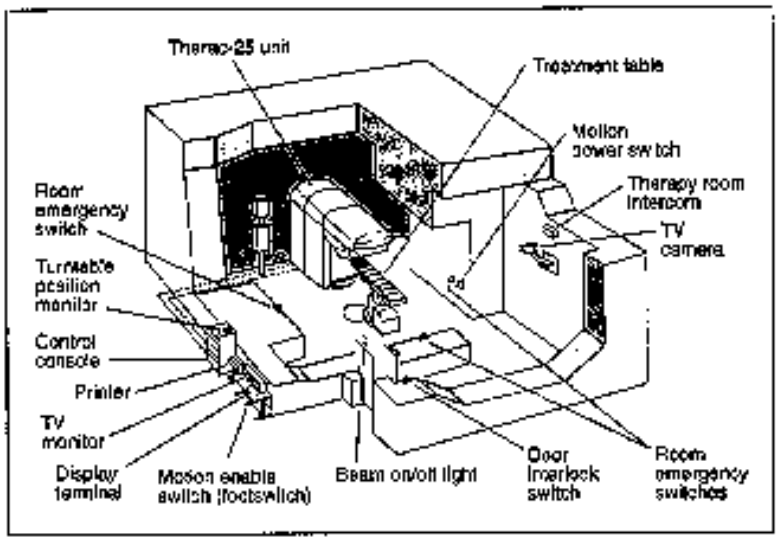
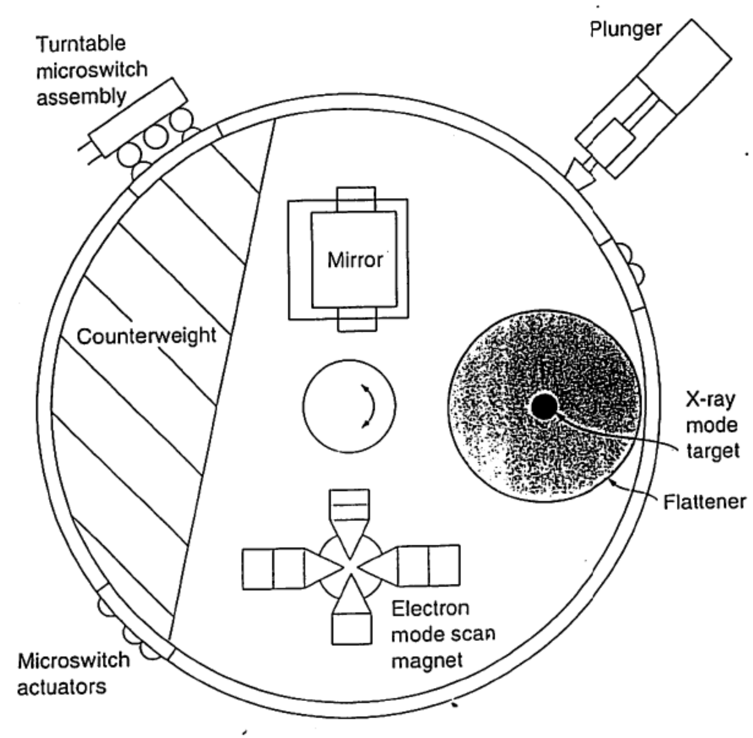


Figure 1. Typical Therac-25 facility.

- 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 models (Therac 6 and 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'
- 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)

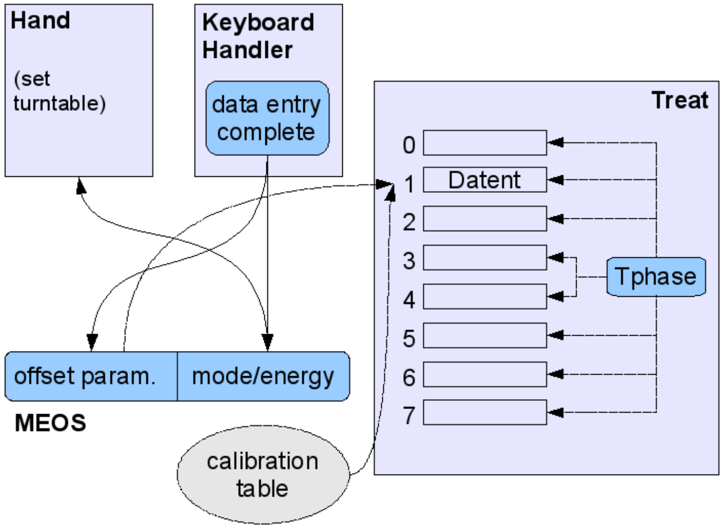
- East Texas Cancer Centre, Mar 86: man burned in neck and died five months later of complications
- Same place, three weeks later: another man burned on face and died three weeks later
- Hospital physicist managed to reproduce flaw: if parameters changed too quickly from x-ray to electron beam, the 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)
- Poor medical device safety usability still costs many lives

Easter 2017

CST 1a

1	2	3
4	5	6
7	8	9
△•	0	▽

Abbott Gemstar

7	8	9
4	5	6
1	2	3
△•	0	▽

Abbott AimPlus

1	2△	3
4	5	6
7	8	9
•	0▽	

CME BodyGuard 545

1	2	3	4	5△	•
6	7	8	9	0▽	

CME BodyGuard 545

1△	2	3▽
4	5	6
7	8	9
	0	•

Graseby 500

1	2	3	4
5	6	7	8
9	0	•	C

Graseby Omnifuse

1	2	3	
4	5	6	•
7	8	9	0

SK Medical SK-500III

7	8	9
4	5	6
1	2	3
0	•	

SK Medical SK-600III

1	2	3	4
5	6	7	8
C	9	0	△
		•	▽

Upreal UPR-900

1	2	3	•
4	5	6	0
7	8	9	

Upreal CTN-TCI-V

7	8	9	0
4	5	6	C
1	2	3	•

BBraun Vista Basic

7	8	9	0
4	5	6	•
1	2	3	C

DRE SP1500 Plus

1	2	3	4	
5	6	7	8	C
	•	0	9	

DRE Avanti Plus

0	1	2	3
•	4	5	6
	7	8	9

Sigma Spectrum

7	8	9
4	5	6
1	2	3
0	•	

Sigma 6000 Plus

1	2	3
4	5	6
7	8	9
	0	•

Sigma 8000 Plus


P Bag Vol: 250ml
Volume Left 246.9
Infused 3.1
Press OK to start
START OK
STOP NO
BOLUS
ON OFF
Info
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
- Biggest killer nowadays: infusion pumps
- Regulators are incompetent / captured
- Nurses get blamed for fatalities
- Avionics are safer, as incentives are better
- Read Harold Thimbleby's paper!

Software safety myths (1)

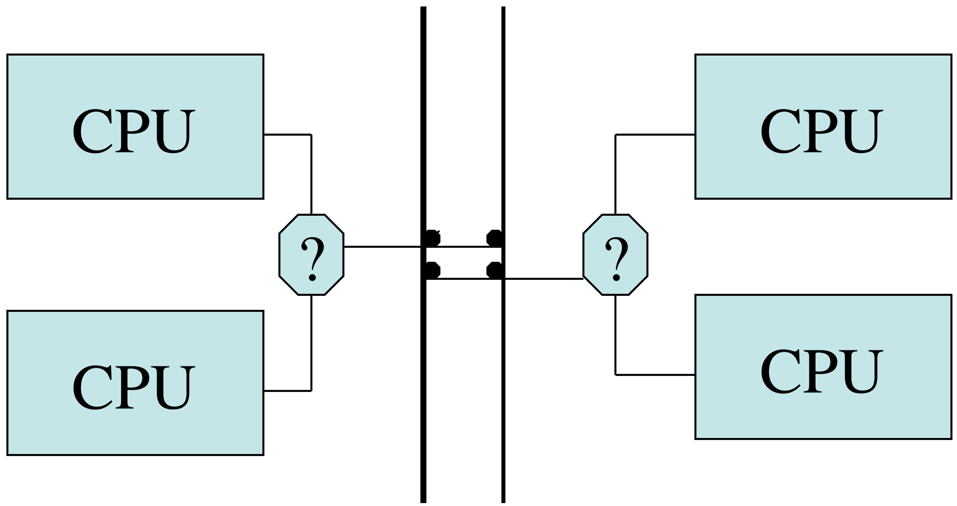
- 'Computers are cheaper than analogue devices'
  - Shuttle software cost $\$10^8$ pa to maintain
- 'Software is easy to change'
  - Exactly! But it's hard to change safely
- 'Computers are more reliable'
  - Shuttle software had 16 potentially fatal bugs found since 1980 – and half of them had flown
- 'Increasing reliability increases safety'
  - They're correlated but not completely

Software safety myths (2)

- ‘Reuse increases safety’
  - Not in Ariane, Patriot and Therac, it didn’t
- ‘Formal verification can remove all errors’
  - Not even for 100-line programs
- ‘Testing can make software arbitrarily reliable’
  - For MTBF of 10^9 hours you must test $>10^9$ hours
- ‘Automation can reduce risk’
  - But thus often takes an extended period of evolution
  - What about architectural hacks such as redundancy?

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

- Aviation is actually an easy case as it's a mature evolved system!
- Stable components: aircraft design, avionics design, pilot training, air traffic control …
- Interfaces are stable too
- Crew capabilities are well known
- The whole system has good incentives for learning – much better than with medical devices!

Pulling it together

- First, understand and prioritise hazards. E.g. the motor industry uses:
  1. Uncontrollable: outcomes can be extremely severe and not influenced by human actions
  2. Difficult to control: very severe outcomes, influenced only under favourable circumstances
  3. Debilitating: usually controllable, outcome art worst severe
  4. Distracting; normal response limits outcome to minor
  5. Nuisance: affects customer satisfaction but not normally safety

Pulling it together (2)

- Develop safety case: hazards, risks, and strategy per hazard (avoidance, constraint)
- Who will manage what? Trace hazards to hardware, software, procedures
- Trace constraints to code, and identify critical components / variables to developers
- Develop safety test plans, procedures, certification, training, etc
- Figure out how all this fits with your development methodology (waterfall, spiral, evolutionary …)

Pulling it together (3)

- Managing a critical property – safety, security, real-time performance – is 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

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!
- The Panix lesson
- Expect everything to go to monthly updates
- This will stress test a lot of regulators!

Tools

- Homo sapiens 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 (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 (but do see 'Objects have failed' on the course page)
- Don't forget the object of all this is to manage complexity!

Incidental complexity (3)

- Early batch systems were very tedious for developer … 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
- Floyd (67), Hoare (71), … now a wide range:
  – Z for specifications
  – HOL for hardware
  – BAN for crypto protocols
- These are not infallible (a kind of multiversion programming)
- Can find a lot of bugs, especially in small, difficult tasks

Static analysis tools

- One outcome of the formal-methods community is modern static analysis tools
- Tools like Coverity don't expect to find all bugs, just many of them
- Problem: when you buy it, you find 10,000 more bugs and your ship date slips
- Similar problems when you upgrade it!
- But attitudes to software are changing...

Programming philosophies

- ‘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?

Programming philosophies (2)

- 'Egoless programming' (Weinberg, 71) – code should be owned by the team, not by any individual. In direct opposition to chief programmer team
  - But: groupthink entrenches bad stuff more deeply
- 'Literate programming' (Knuth et al) – code should be a work of art, aimed not just at machine but also future developers
  - But: creeping elegance is often a symptom of a project slipping out of control

Capability maturity model

- Humphrey, 1989: it's important 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 CMU with DoD money
- 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 style

- Over the past 20 years, emphasis shift 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

- See guest lecture for this!
- Start with a sound technical foundation: languages, build environment, testing
- Agree processes: daily scrum, weekly lunch, customer interaction...
- Break the development into short sprints
- Figure out what else is needed (e.g., security policy, safety case, real-time constraints...)

The specification still matters!

- Study of failure of 17 large demanding systems, Curtis Krasner and Iscoe 1988
- 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→ 2 → 3

But specification is hard

- Thin spread of application domain knowledge
  - How many people understand everything about running a phone service / bank / hospital?
  - Many aspects are jealously guarded secrets
  - Some fields try hard to be open, e.g. aviation
  - Or with luck you might find a real 'guru'
  - But you can expect specification mistakes
- The spec may change in midstream anyway
  - Competing products, new standards, fashion
  - Changing environment (takeover, election, …)
  - New customers (e.g. overseas) with new needs

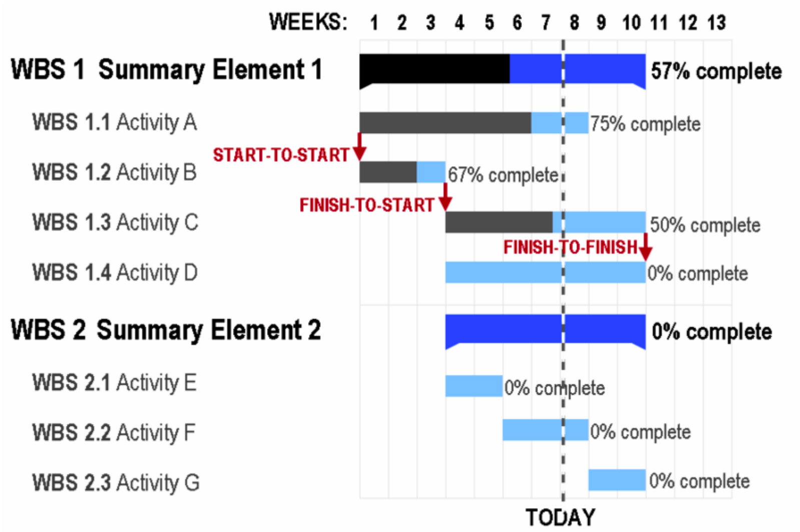
How the spec can kill you...

- Spec-driven development of large systems leads to comms problems – N people means N(N-1)/2 channels and 2^N subgroups
- Big firms do hierarchy; but if info flows via 'least common manager', bandwidth will be inadequate
- So you proliferate committees, staff departments
- This causes politicking, blame shifting
- Management attempts to gain control result in restricting many interfaces, e.g. to the customer

Project management

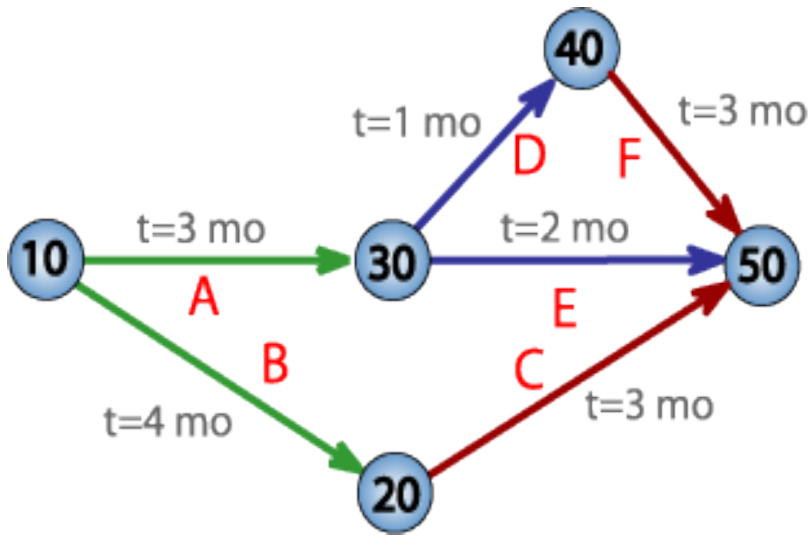
- A manager's job is to
  - Plan
  - Motivate
  - Control
- The skills involved are interpersonal, not techie; but managers must retain respect of techie staff
- Growing software managers a perpetual problem! 'Managing programmers is like herding cats'
- Nonetheless there are some tools that can help

Activity Charts



- ‘Gantt’ chart (after inventor) shows tasks and milestones
- Problem: can be hard to visualise dependencies

Critical path analysis



- Project Evaluation and Review Technique (PERT): draw as a graph with dependencies
- Give critical path (here, two) and shows slack
- Can help maintain 'hustle' in a project
- Also helps warn of approaching trouble

Keeping people motivated

- People can work less hard in groups than on their own projects – 'free rider' or 'social loafing' effect
- Competition doesn't invariably fix it: people who don't think they'll win stop trying
- 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, leadership, and 'team building' (shared food / drink / exercise; scrumming)

Testing

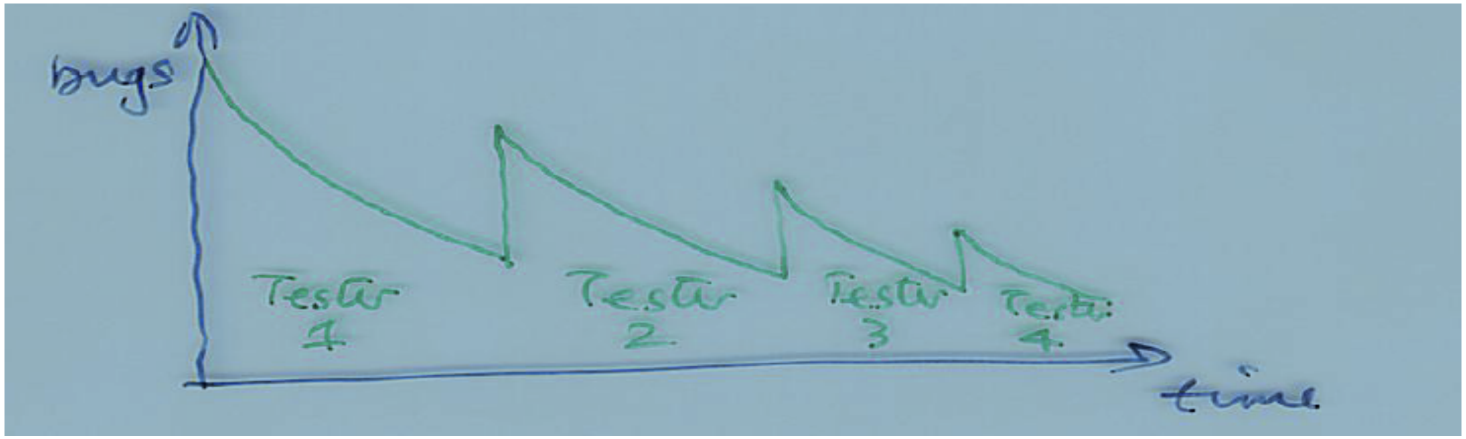
- Testing is often neglected in academia, but it's typically about half the effort, and half the cost
- Bill G: "are we in the business of writing software, or test harnesses?"
- Happens at many levels
  - Design validation, UX prototyping
  - Module test after coding
  - System test after daily build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

Testing (2)

- Huge advance: design for testability, continuous integration, and automated regression tests
- Tests check that new versions of the software give same answers as old versions
  - Customers more upset by failure of a familiar feature than at a new feature which doesn't work right
  - Without regression testing, 20% of bug fixes reintroduce failures in already tested behaviour
  - Test the inputs that your users actually generate!
  - In hard-core agile philosophy, the tests are the spec

Testing (3)

- Reliability growth models help us assess mtbf, number of bugs remaining, economics of further 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
- But: changing testers brings new bugs to light



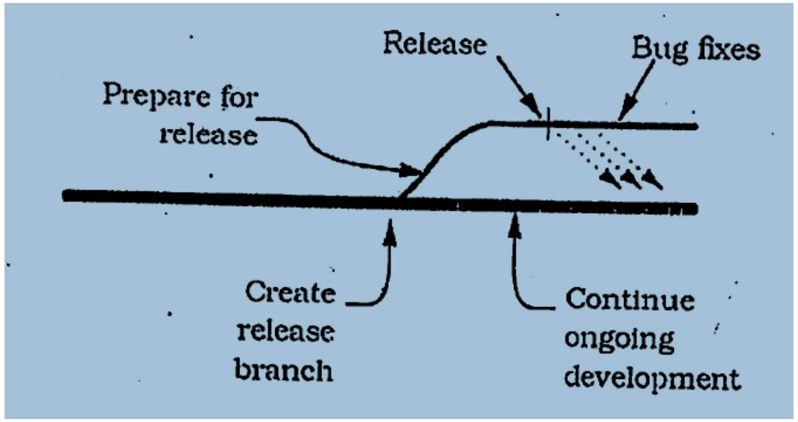
Testing (4)

- The critical problem is to exercise the conditions under which the system will actually be used
- Many failures result from unforeseen input / environment conditions (e.g. Patriot)
- Random testing – fuzzing – now good practice
- Incentives still matter: commercial developers look for friendly certifiers while military, NASA, DoE arrange hostile review
- So: to whom do you have to prove what?

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

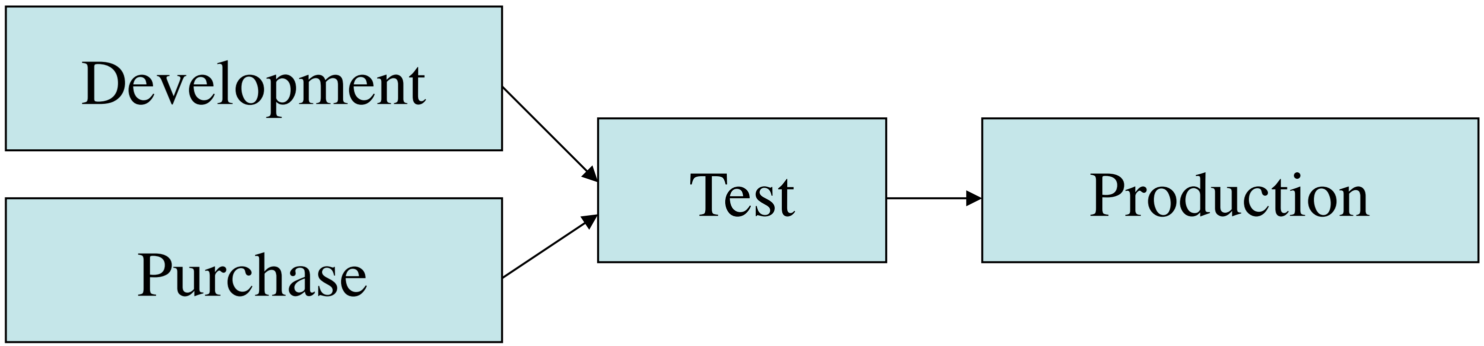
Release management



- Getting from development code to production release
- Main focus is stability – work on recently-evolved code, test with lots of hardware versions, etc
- Add all the extras like copy protection, rights management
- Critical decision: patch old versions, or force upgrades?

Change control

- Change control and configuration management are critical yet often poor
- The objective is to manage the testing and deployment of software you've written, or bought, or got fixes for
- Someone must assess the risk and take responsibility for live running, and manage backup, recovery, rollback etc



Responsible disclosure

- Old approach: try to deny existence of bugs for as long as you can
- Reaction: hackers boast about them
- Consensus in 2000s: vulnerabilities should be disclosed after a time delay
- Immediate disclosure: then instant exploit
- No disclosure: then vendors won't fix
- Can use CERTs, regulators as channel

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 (non-IT products, orphan products like old phones)
- What do we do about Mirai?

Shared infrastructure

- We share a lot of code through open source operating systems, libraries and tools
- Huge benefits but also interaction issues!
- Can you cope with an emergency bug fix (like Heartbleed)?
- How do you feed your fixes back to others?
- Do you encourage responsible disclosure?
- Are you aware of different license terms?

Agency issues

- Recall the lessons from LAS! Employees often optimize their own utility, not the project's
- Bureaucracies are machines for avoiding blame!
- Risk reduction becomes compliance
- Tort law reinforces herding: negligence judged 'by the standards of the industry'
- So firms do the checklists, use fashionable tools, hire the big consultants…

How do you know when you're done?

- Security: the Cathedral
  - the Common Criteria
  - protection profiles and CLEFs
  - rating maintenance, accreditation...
- Or the Bazaar
  - patch cycle
  - responsible disclosure
  - breach reporting

Safety

- Mostly the Cathedral rather than the Bazaar
- Aircraft: the FAA and the CAA
- Cars: UNECE standards, independent lab testing (mandatory in Europe)
- Medical devices: FDA in USA, national regulators in Europe (but being harmonised)
- Regulation and liability
- Regulatory capture

Focus on outcomes, or process?

- Outcomes
  - Metrics easier for regular losses (risk)
  - But rare catastrophes are harder (uncertainty)
  - How reassuring are fatality statistics? E.g. Train Protection Systems, Tesla
  - Accidents are random, but not security exploits!
  - Product liability for death or injury is strict (more in Economics, Law and Ethics in 1b)

Focus on outcomes, or process?

- Process
  - Necessary to adapt as environment changes
  - Security development lifecycle is established
  - Safety rating maintenance
  - And blame avoidance is what bureaucracies do
  - Public sector is really keen on ‘compliance’
  - But leaves a gap of residual risk / uncertainty

Incentives

- The world offers hostile review, which we tackle in stages (dogfood, alpha, beta, ops)
- Some applications use hostile reviewers deliberately (higher assurance levels of CC, manned spaceflight, nuclear weapons)
- Standard contract in Bangalore: you have to fix bugs for 90 days after sale
- Businesses avoid risk (regulatory games!)

A big question

- At present cars get pre-market testing
- Tesla has started monthly updates, like for phones and laptops; the others will follow
- But cars last 200k miles, and improving
- We don't know how to patch old software!
- So how will today's Land Rover get patches in 2037? In 2047?
- 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
- Security engineering is going the same way, and the two are merging as we put CPUs everywhere
- 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 privacy, as well as underlying properties like security, are often emergent

Conclusions (2)

- Complex systems are usually socio-technical, so people come into play as users, and also as members of development and other teams
- Institutions matter. About 30% of big commercial projects fail – has been stable for years! (better tools let people climb higher up the complexity mountain before they fall off)
- In future, the confluence of security and safety may make maintenance the complexity limit, even more than at present!