

# Software and Security Engineering

Lecture 1

**Alastair R. Beresford**

arb33@cam.ac.uk

*With many thanks to Ross Anderson*

1

This course is usually lectured by Prof Ross Anderson and therefore much of the material is derived from an earlier version of the lecture course prepared by him.

# Aims

- Introduce software engineering with focus on:
  - Large systems
  - Safety-critical systems
  - Systems to withstand attack by capable opponents
- Illustrate what goes wrong
- Best practice to avoid failure

2

So far in Part 1A you have written small sample programs by yourself. Most software development in industry is at a significantly larger scale, involving teams of people and often involves safety or security requirements.

*All* significant pieces of software contain latent defects – bugs yet to be discovered. This affects both safety and security.

In this course we look at what has gone wrong in the past through case histories, and look at the development and management practices which have arisen in order to avoid failures in the future.

# Objectives

- **By the end of the course you should be able to:**
  - Write programs with tough assurance targets
  - Work effectively as part of a team
- **Understand**
  - Software development models
  - Development lifecycle
  - Understand bugs, vulnerabilities and hazards

3

In order to write programs which meet tough assurance targets and work effectively as a team member, you need to be able to apply appropriate programming best practice as described in this course. For example, how to use version control for source code, automated build systems, and suitable testing strategies.

An understanding of historical and current software development models such as waterfall, spiral and agile methods will allow you to select the right approach for a given project.

It is important to understand the terminology used (e.g. what is a *bug*, a *hazard*, or a *vulnerability*?) so you can correctly understand case histories and can communicate effectively with others.

An important aim is to prepare you for the Part IB group project. The techniques described here will also help with your Part II and Part III projects as well as other later courses such as Security and Concurrent and Distributed Systems.

# Books



4

This is a course which requires you to read around the subject. Text books are very helpful for this course, and here are three which offer useful, complementary perspectives.

R. Anderson, *Security Engineering* (2nd Edition, 2008). You may like to start by reading Part I and also Chapters 25 and Chapter 26. Available online: <https://www.cl.cam.ac.uk/~rja14/book.html>

M. Howard and D. LeBlanc, *Writing Secure Code* (2nd Edition, 2003).

N. Leveson, *Safeware: System Safety and Computers* (1994). See also her more recent book, *System Safety Engineering: Back To The Future* (2002), which is also available online: <http://sunnyday.mit.edu/book2.pdf>

## Make use of additional reading

F.P. Brooks, *The Mythical Man Month*

J. Reason, *The Human Contribution*

S.W. Thames, *Report of the Inquiry into the London Ambulance Service*

S. Maguire, *Writing Solid Code*

H. Thimbleby, *Improving safety in medical devices and systems*

O. Champion-Awwad et al, *The National Programme for IT in the NHS – A Case History*

<https://www.cl.cam.ac.uk/teaching/current/SWSecEng/materials.html>

5

In addition to the core text books, these are some of the additional text books, academic articles and white papers which you may find interesting.

The course materials page contains further links to related reading and an annotated slide deck:

<https://www.cl.cam.ac.uk/teaching/current/SWSecEng/materials.html>

You should not feel constrained to these materials. Read about any application areas which are interesting to you. This will help both with your general understanding and also provide you with useful perspectives and examples which you can use to inform your future career as well as supporting material to refer to in the forthcoming Tripos examinations.

Remember: wide-reading driven by curiosity will help a lot with this course.

# Use the ICAP framework to guide your learning

- Interactive
- Constructive
- Active
- Passive

*“Teachers open the door,  
But you must enter by yourself.  
Tell me and I forget.  
Teach me and I remember.  
Involve me and I learn.”  
– Benjamin Franklin*

**Or: reading is essential but insufficient**

6

*“interactive activities are most likely to be better than constructive activities, which in turn might be better than active activities, which are better than being passive.”*

Micheline Chi, Active-Constructive-Interactive: A Conceptual Framework for Differentiating Learning Activities, Topics in Cognitive Science, 1(1):73-105 2009.  
<https://onlinelibrary.wiley.com/doi/full/10.1111/j.1756-8765.2008.01005.x>

Examples of different types of learning:

- Interaction: discussing with peers, a supervision
- Construction: completing an example sheet, writing a summary in your own words
- Active: Taking notes of what the lecturer says, highlighting a passage
- Passive: Reading a book, listening to a lecture or video

Takeaway message: You need to read books and papers, but to really understand the material, you need to build artefacts, talk to others and critique the ideas.

# Using laptops in lectures can harm everyone's learning outcomes

## **The Pen Is Mightier Than the Keyboard: Advantages of Longhand Over Laptop Note Taking**



**Pam A. Mueller<sup>1</sup> and Daniel M. Oppenheimer<sup>2</sup>**

<sup>1</sup>Princeton University and <sup>2</sup>University of California, Los Angeles

Psychological Science  
1–10  
© The Author(s) 2014  
Reprints and permissions:  
sagepub.com/journalsPermissions.nav  
DOI: 10.1177/0956797614524581  
pss.sagepub.com

### **Abstract**

Taking notes on laptops rather than in longhand is increasingly common. Many researchers have suggested that laptop note taking is less effective than longhand note taking for learning. Prior studies have primarily focused on students' capacity for multitasking and distraction when using laptops. The present research suggests that even when laptops are used solely to take notes, they may still be impairing learning because their use results in shallower processing. In three studies, we found that students who took notes on laptops performed worse on conceptual questions than students who took notes longhand. We show that whereas taking more notes can be beneficial, laptop note takers' tendency to transcribe lectures verbatim rather than processing information and reframing it in their own words is detrimental to learning.

Many studies in recent years have found that using laptops in lectures adversely affects learning outcomes, not just for the user, but also for those sitting nearby. Studies have also shown that writing with a pen aids recall over the use of laptops, as seen here. If you normally use a laptop, try using a pen and paper (or tablet and pen) for this course.

If you really want to use a laptop, then try and summarise what I say (a constructive activity), don't transcribe the lecturers presentation verbatim (merely an active learning activity). If you want to browse the Internet, then please do that elsewhere, not in the lecture. (Lecture attendance in Cambridge is not compulsory.)

Paper reference: Pam Mueller and Daniel Oppenheimer. The Pen Is Mightier Than the Keyboard: Advantages of Longhand Over Laptop Note Taking, *Psychological Science*, 1(10), 2014. <https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/6/132/files/2010/11/Psychological-Science-2014-Mueller-0956797614524581-1u0h0yu.pdf>

Further reading: <https://cs.brown.edu/courses/cs019/2018/laptop-policy.html>

## Course Outline – key topics

- Security policy
- Safety case
- Security protocols
- User behaviour
- Bugs
- Software crisis
- Development lifecycle
- Critical systems
- *Testability*
- *Software-as-a-service*

8

The last two topics are given by Dr Andrew Rice and Dr Richard Sharp on the 8<sup>th</sup> and 10<sup>th</sup> May respectively.



# What is Security Engineering?

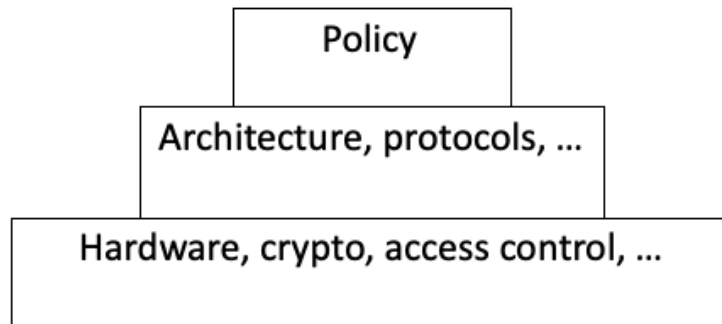
*Security engineering is about building systems to remain dependable in the face of malice, error and mischance.*

9

Security Engineering as a discipline focuses on the tools, processes and methods needed to design, implement and test complete systems, and to adapt existing systems as their environment evolves. Note that safety engineering has a similar high-level goals.

When considering the security or safety of the system, it is not sufficient to look at each component in isolation (although that is important). It is essential to look at how the whole system fits together. Security and safety is not composable.

# The Design Hierarchy



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

10

The traditional design hierarchy requires us to start by defining, at a high level, what we are trying to achieve. Then we explore the question of how to do so in terms of overall strategy and architecture. Finally we need to explore the detail: which hardware platform should we use, what cryptographic primitives are the right ones, and so on.

For example, we might decide we are going to build autonomous cars which reduce the number of accidents by 50% when compared to human drivers. Given that high-level goal, we then explore the architecture which might allow us to get there. Only then are we ready to make detailed decisions on specific technologies.

## A system can be...

- equipment or a component (laptop, smartcard, ...)
- a collection of products, their operating systems, and some networking equipment
- The above plus applications
- The above plus internal staff
- The above plus external users

**Common failure: policy drawn too narrowly**

11

The definition of the system is often too narrowly defined.

For example, if a company produces a mobile app for a smartphone, it might assume that the app is the system. This is too narrow a definition. For example, what about the operating system running on the phone, or its hardware? Does the operating system get regular updates? What other apps are installed on the phone, and are they malicious? What about all the servers that the app communicates with?

We study cybercrime in the department. Most of the failures are not hi-tech, but rather use a system in bad and unintended ways. Cyberbullying via a messaging platform is one example: the platform is delivering the messages as requested, but overall the system is failing to protect users from harm.

## Electric bike should not propel bicycle when speed exceeds 15.5 mph



12

Here is an example where you need to think broadly about the definition of the “system”.

In the UK, in common with many countries, electric bikes can only be ridden without a license, tax or registration provided it has pedals and electric assistance does not occur when travelling over 15.5 mph (25 km/h). So how do cyclists get around this restriction?

The bicycle in this picture has electric assist. The electronics in the bike estimates the speed of the bike by counting revolutions of the rear wheel passing in front of a sensor. The “badassbox” works by suppressing the sensor reading in every other revolution, allowing you to travel twice as fast with electric assist. A by-product of this is that the speedometer on the bike no longer provides an accurate reading.

Further reading:

- Details on the badass box, <https://www.ebiketuning.com/badass-box-4-for-shimano.html>
- Rules around electric bikes in the UK, <https://www.gov.uk/electric-bike-rules>

# Security vs Dependability

**Dependability = Reliability + Security**

- **Malice is different from error**
- **Reliability and security are often strongly correlated**

13

Reliability and dependability sounds like they might mean the same thing. However, we demand greater precision in our use of terms.

Since malice is different from error, we wish to capture this. For example, a system might state a reliability guarantee, such as “Bob will be able to read this file”, while a security guarantee might state that “Foreign governments won’t be able to read this file”. Typically we want a dependable system with both reliability and security.

This example motivates the need to define terms carefully. Note that while we will be consistent in this course, terms may have different meanings (or different terms have the same meaning) in different communities. For example, the safety and security communities currently use different language.

# Subjects and principals

***Subject:*** a physical person

***Person:*** a subject or a legal person (firm)

***Principal:***

- A person
- Equipment
- A role, including complex roles

14

We adopt the same language as used by the legal profession and define a person as either a *subject* (physical person) or a legal person which can also include a limited company (e.g. Google) or a charity (University of Cambridge).

We use the term principal as a more general term to cover people, equipment and more general labels. The term role is often used to as a means of indirection between a principal and a person. For example, “the officer on watch”, or “Alice and Bob” or “Alice or any of her current direct reports”.

The definition of a principal can get quite complex. Sometimes we need to distinguish between “Bob’s smartcard representing Bob who’s standing in for Alice” from “Bob using Alice’s card in her absence”. For example, consider the case of a bank, whose policy states that all withdrawals over 10,000 GBP must be approved by any two bank managers out of the set of Alice, Bob or Charlie.

# Secrecy and privacy

*Secrecy*: mechanism to control which principals can access information

*Privacy*: control of your own secrets

*Confidentiality*: an obligation to protect someone else's secrets.

15

Secrecy often, but not always, implies a technical mechanism. This does not necessarily involve cryptography.

Privacy has many definitions which are wider than the one used in this course. For those who are interested in such things, you might wish to look up the right to be forgotten.

These three concepts are interrelated. For example, your medical privacy is protected by your doctors' obligation of confidentiality.

# Anonymity, integrity, authenticity

- *Anonymity*: restrict access to metadata
- *Integrity*: an object has not been altered since the last authorised modification
- *Authenticity* has two common meanings:
  - an object has integrity plus freshness
  - You are speaking to the right principal

16

Anonymity has various flavours, from not being able to identify subjects to not being able to link their actions. A simple example is k-anonymity where subjects are indistinguishable from k-1 others (subjects are said to be “k-anonymous”).

A cheque has integrity (a signature) and freshness (a recent date) together giving it authenticity.



## Trust is hard; 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
5. ...

17

Trust is really hard. It can exist at different levels: human norms (you trust your doctor, and he has a warm manner, a nice office, etc). Trust can also be a trusted system. Are these the same thing? Yes and no. Yes, because the doctor can break a trusted system by malice or by accident (writes down his password which is visible to others); no in other cases.

We are going to use the second definition (the NSA definition) for this course. Under this definition, an employee of GCHQ selling cryptographic key material to a foreign power is *trusted* but not *trustworthy* (assuming of course that such a sale has not been authorized).

## Errors, failures, reliability, accidents

- **Error**: a design flaw or deviation from intended state
- **Failure**: nonperformance of the system when inside specified environmental conditions
- **Reliability**: probability of failure within a specified period of time
- **Accident**: an undesired, unplanned event resulting in a specified kind or level of loss

18

These are terminology from the safety community.

Failure is often expressed as Mean-Time-Before Failure (MTBF), or Mean-Time-To-Failure (MTTF). For example, a single-engine plane might have an MTBF of 240,000 hours overall. This isn't necessarily a meaningful summary on its own – other steps might be required to ensure this is the case. For example, it might have an MTBF of 5,000 hours if its not serviced correctly (e.g. oil change is forgotten).

## Hazards and risks

- **Hazard**: a set of conditions in a system or 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
  - Often combined with *unit of exposure*; e.g. a *micromort*
- Uncertainty is where the risk is not quantifiable
- Safety is simple: freedom from accidents

19

In a single-engined aircraft, a hazard might be the mountain you fly over at night since you will crash if the engine fails. There is less hazard due to engine failure when flying over the East Anglia during the day since its flat and you can probably land safely in a field somewhere.

David Speigehalter uses the "micromort" as a unit of risk, defined as a one-in-a-million chance of death. For example, taking the data from the Office of National Statistics for 2012, 499,331 people died in England and Wales out of a population of 56,567,000. Therefore the chance of death overall for each citizen is, on average, 24 micromorts per day. We can use this concept of micromorts per unit of exposure to assess the comparative risk of activities. For example, data can be used to estimate that scuba diving is 5 micromorts per dive; skydiving, 8 micromorts per jump; and skiing, 0.7 micromorts per day.

So what about terrorism? It has a tiny micromort! Yet we still care – because humans are not always rational.

It is worth highlighting that risk, or the probability of an accident, is different from the probability of failure or MTBF. A component can fail without it causing an event resulting in loss. In a twin-engine plane, one engine can fail and yet there is no accident.

## Security policy, profile, and target

- A *security policy* is a succinct statement of protection goals
- A *protection profile* is a detailed statement of protection goals
- A *security target* is a detailed statement of protection goals applied to a particular system

20

A security policy is typically less than a page of text written in plain language.

A protection profile is typically dozens of pages written in a semi-formal language.

A security target may run to hundreds of pages for both functionality and testing.

## 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?

21

[Ask the audience to talk to a neighbour and producing a list of problems with this policy]

Many things wrong with this policy. Examples include:

- \* The policy reduces to "need-to-know", but what does this mean?
- \* What's a "breach"?
- \* Reporting a breach is passive voice: who does the reporting of a breach? Over what time period?
- \* You need to trust the employees to adhere to policy. Do they feel trusted and empowered?
- \* There's nothing in this policy which you can implement (e.g. support employees, or turn into software): this policy is *security theatre*.

In the UK there's no general requirement to report a crime (with the exception of terrorism).

Edward Snowden is an example worth considering: he released lots of data because he felt that his duty as a soldier was to leak data since, in his opinion, what the NSA was doing was contrary to the constitution, and he had signed up to protect the constitution.

## Traditional government approach

- Start from the threat model: an insider who is disloyal or careless.
- Solution: limit the number of people you 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'

22

Examples of disloyal insiders include Burgess/MacLean, Aldrich Ames, Edward Snowden. Carelessness can include "loose talk", reading papers on train, being photographed outside Number 10 with papers in hand, malware on PC, and so on.

Another important concept is *vetting*, in which the background of employees working with sensitive information is investigated. Does the employee have any weaknesses which might be exploited by a third-party? For example, does the employee have unsustainable debts, a drinking or drug problem, can they be "bought off", etc.

# Multilevel Secure Systems (MLS)

- Classify all documents and data with a level, such as *official, secret, top secret; or high and low.*
- Principals have clearances; clearance must equal or exceed classification of any documents viewed.
- Enforce handling rules for material at each level.
- Information flows upwards only:
  - No read up
  - No write down

23

MLS are widely used by governments.

There are a variety of classification levels in use around the world. The UK Government uses Official, Secret, Top Secret ([https://en.wikipedia.org/wiki/Government\\_Security\\_Classifications\\_Policy](https://en.wikipedia.org/wiki/Government_Security_Classifications_Policy)). In this course, in common with much of the computer science literature, we will often use High and Low to describe a simple abstract representation of an MLS system.

Recall 'mandatory access control' from OS course.

Information flows are integrated with the operating systems as used by government employees. Clearly the OS needs to prevent employees with clearance to work with material up to secret level from accessing any files classified at top secret. No write down is also important to stop information leakage. For example, the OS will allow the user to cut-n-paste data from a confidential file to a secret one, but not vice-versa (or if you can, then the confidential file then becomes classified at secret).

# Bell-LaPadula formal model

- Bell-LaPadula (1973):
  - *simple security policy* (no read up)
  - *\*-policy* (no write down)
- With these two rules, one can prove that a system that starts in a secure state will remain in one
- Aim is to minimise the Trusted Computing Base

24

At first people thought that you only needed no read up, but then if you get malware running at high, it can leak data to low; so we also need the \*-property.

The Trusted Computing Base includes all the hardware and software required to enforce security policy.



# Covert channels cause havoc

- BLP lets malware move from Low to High, just not to signal down again.
- What if malware at High modulates shared resource (e.g. CPU usage) to signal to Low?
- How can you let message traffic pass from Low to High, if any acknowledgement of receipt could be delayed and used to signal?

**Moral: covert channel bandwidth is a complex.  
It's an emergent property of whole systems!**

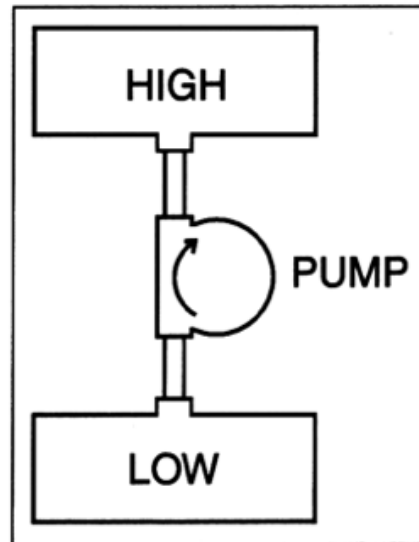
25

A covert channel occurs when the performance of a resource shared between Low and High allows information to flow which contravenes policy. An example shared resource might be a CPU shared between processes, some of which are running at High and some at Low. Then a High process can transfer data to a Low process by either using lots of CPU (to send a "one") or not using the CPU (signaling a "zero").

More information: [https://en.wikipedia.org/wiki/Covert\\_channel](https://en.wikipedia.org/wiki/Covert_channel)

## High assurance MLS system

- The pump simplifies the problem: replace the complex emergent property of the whole system with a simple property of a testable component
- Nevertheless, often harder than it looks!



# Multilateral Security

Stop lateral flow, examples:

- Intelligence, typically with compartments
- Medical records
- Competing clients of an accounting firm



27

MLS is good at stopping data from flowing from High to Low. In other settings, you want to stop lateral flows of information. Accounting firms use this to allow them to work for two or more firms who compete in the same sector. Accountants at one of the Big Four working on accounts for BP need to make sure that they don't talk to colleagues who are working on Shell's accounts.

# Biba formal model for integrity

- Biba (1975)
  - Simple integrity policy (no read down)
  - \*-integrity policy (no write up)
- Dual of the Bell-LaPadula model
- Examples:
  - Medical devices with *calibrate* and *operate* modes
  - Electricity grid controls with safety at the highest level, operational control as the next, and so on.

28

The Biba model is the inverse of the Bell-LaPadula model: instead of protecting confidentiality, it protects integrity. In this model, we do not want a process at High from being influenced by data from Low.

For example, a nuclear power station will have safety as at the top level: and any control of the power station for the safety of the power grid itself will be ignored (a blackout is better than a nuclear meltdown).

In practice, safety systems need more than simply High and Low. Compartmentalization is often a good way forward, and to do this we can apply the multi-lateral security model.

# Software and Security Engineering

Lecture 2

**Alastair R. Beresford**

arb33@cam.ac.uk

*With many thanks to Ross Anderson*

ANDY GREENBERG SECURITY 07.21.15 06:00 AM

# HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT



[Watch first 3 minutes of the video. Ask the audience to write down all the aspects of the car which the remote attackers could control.]

Further reading and video: <https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/>

# Architecture matters



- Lots of legacy protocols trust all network nodes
- Chrysler Jeep recall
- Defence in depth: separate subnets, capable firewalls,

31

There are a wide variety of legacy protocols in existence: DNP3 in control systems, CAN bus in cars, and so on. Many had either no stated security policy at all, or a security policy which does not take remote networking into account (e.g. the Internet).

As we just saw in the video, the Jeep Cherokee could be controlled remotely over the mobile network. This is a poor architecture: you don't want to allow unfettered remote control of the CAN bus, but equally well you need to for other things to function. How do you fix? Can we apply some of the ideas from models we saw in the last lecture (e.g. multilevel and multilateral security)? Defence in depth is also important in order to ensure the failure of one component does not lead to an accident.

# Swiss Cheese Model

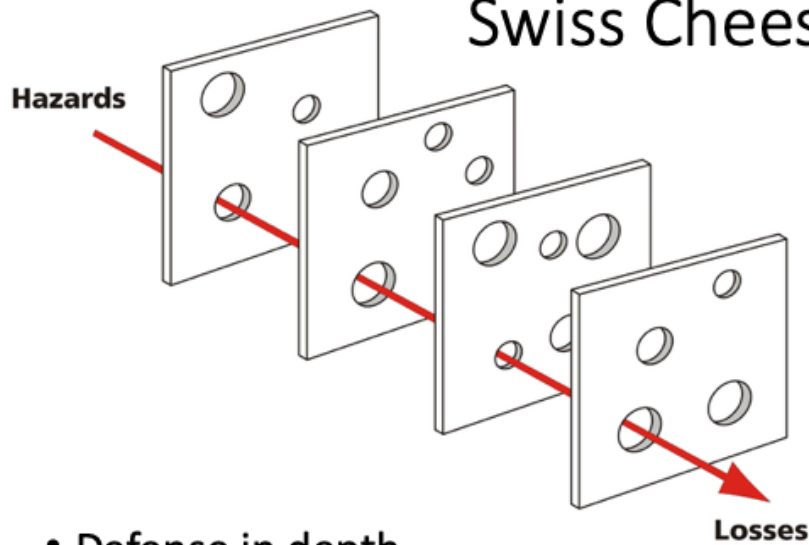


Diagram by  
Davidmack  
CC-BY-SA 3.0

- Defense in depth
- Layers could include hardware, software, policy, human factors, etc.

32

The Swiss Cheese model of accident causation is a model used in risk analysis and risk management. The aim is to ensure *defense in depth*: what might be open at one layer is closed at another. Defense in depth works provided that there is at least one layer which does not have a flaw which allows a hazard to turn into an accident. An accident occurs when the weakness in every layer lines up – something we wish to avoid.

[https://en.wikipedia.org/wiki/Swiss\\_cheese\\_model](https://en.wikipedia.org/wiki/Swiss_cheese_model)



# Safety policies

- Industries have their own standards, cultures, often with architectural assumptions embedded in component design
- Plethora of safety legislation
- Sometimes brand new standards, but in more mature industries safety standards tend to evolve
- Two basic ways to evolve:
  - *failure modes and effects analysis*
  - *fault tree analysis*

33

Safety is sectoral: those working on car safety don't talk to the aircraft industry.

Many industries are much more tightly regulated than the computer industry. For example, there are over 180 regulations for cars. Example: "ABS failure mustn't cause asymmetric braking".

Understanding and improving our understanding of the relationships between failures and outcomes can be bottom-up (failure modes and effects analysis) or top-down (fault tree analysis). Both approaches are useful.

## Failure modes and effects analysis (bottom-up)

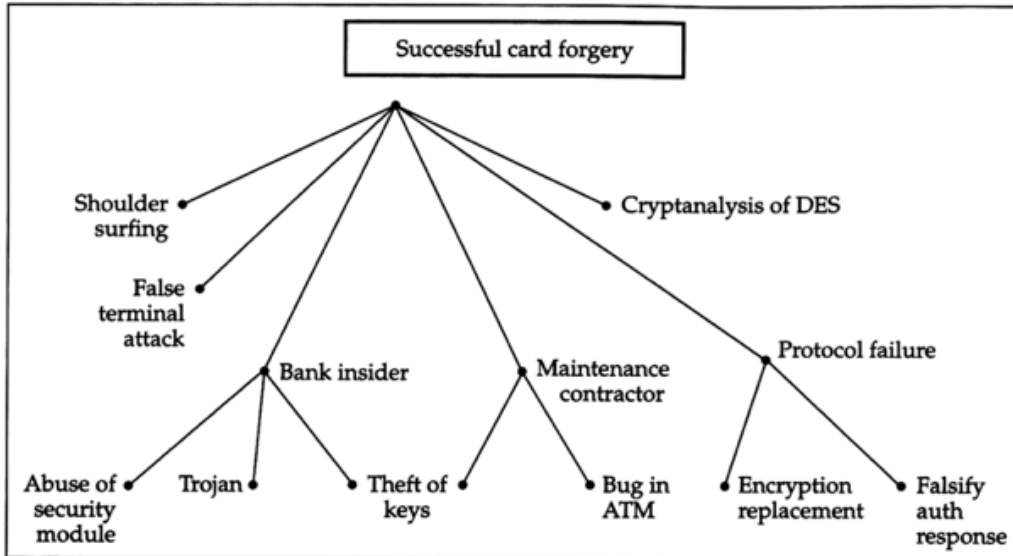
- Look at each component and list failure modes
- Figure out what to do about each failure
  - Reduce risk by overdesign?
  - Redundancy?
  - ...
- Use secondary mechanisms to deal with interactions
- Developed by NASA

34

Example: You've got a person sitting on a rocket. What could go wrong? What would cause the rocket to explode on the ground? What would cause the second stage not to separate from the first? And so on. You can write down all the failures bottom-up. An extreme example: what would happen if bolt number 40674 fails when attached to a rocket fin? Could we reduce the risk of failure by increasing it's diameter, length or material used in manufacture? Could we introduce two bolts (redundancy) in order to reduce the risk of failure?

Example 2: For planes, the basic failure you worry about is the engine failing; examples include running out of fuel, engine on fire, etc. This is only a real problem when flying over mountains, forests or the ocean since otherwise an emergency landing is possible. The question you might ask is how long does the aircraft need to survive on the flight?

# Fault tree analysis (top-down)



*Work backwards from bad outcome we must avoid to identify critical components*

35

This is the opposite to failure modes idea: Start at the top with a list of bad things you wish to avoid, and for each one work down.

[Talk through the examples on each of the nodes in the tree.]

This works particularly well if there's a small, finite, number of bad things which could happen: you start with each bad outcome and work your way down from it.

## Example: nuclear bomb safety

Don't want Armageddon caused by a rogue pilot, a stolen bomb, or a mad president, so require

- Authorisation: president releases code
- Intent: pilot puts key in bomb release
- Environment:  $N$  seconds zero gravity

Independent, simple, technical mechanisms

36

NATO countries require three things: authorisation, intent, environment.

Authorisation: there's a code which needs typing into the weapon. This code is kept by a responsible person with authority to use the bomb.

Intent: the pilot needs to decide when and where the bomb will be released.

Environment: This is really important since it's quite hard for the malicious person to get, say, 20 seconds of zero-gravity without access to a jet engine.

We hope that these things are orthogonal and therefore you can only set off a bomb unless you have the bomb, a mad or otherwise compromised president, a well-trained evil pilot, and access to a jet plane.

## Bookkeeping, 8-4<sup>th</sup> millennium BCE



37

These are bullae from the British Museum. Each bulla was used to record the stock stored in the granary so that when you deposit the wheat or olive oil you receive a bulla back; you can then present your bulla later in the year to get the goods back. Note that the bulla were pushed into clay, and the clay is then fired so that you had a seal (each party kept one half and therefore this lets each other validate the outcome.)

This is where writing comes from: "As the clay tokens and bulla became difficult to store and handle, impressing the tokens on clay tablets became increasingly popular. Clay tablets were easier to store, neater to write on, and less likely to be lost. Impressing the tokens on clay tablets was more efficient but using a stylus to inscribe the impression on the clay tablet was shown to be even more efficient and much faster for the scribes. Around 3100 BCE signs expressing numerical value began. At this point, clay tokens became obsolete, a thing of the past."

[https://en.wikipedia.org/wiki/Bulla\\_\(seal\)](https://en.wikipedia.org/wiki/Bulla_(seal))

## Bookkeeping, circa 1100 AD

- Double-entry bookkeeping: each entry in one ledger is matched by opposite entries in another
- Ensure each ledger is maintained by a different subject so bookkeepers have to collude to defraud
- Example: a firm sells £100 of goods on credit, so credit the sales account, debit the receivables account. Customer subsequently pays, so credit the receivables account, debit the cash account.

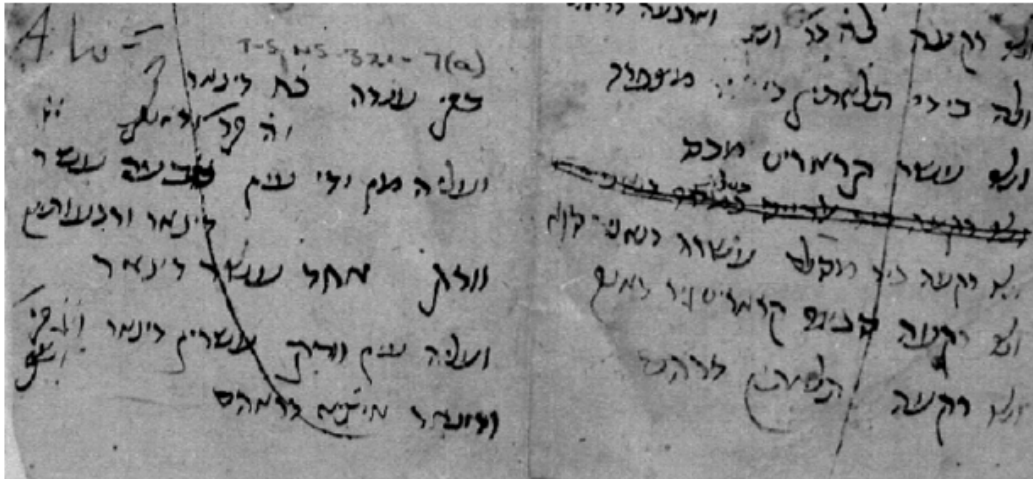
38

How do you manage a business that's grown too big to staff with your own family members?

Double-entry bookkeeping allows you to check the behaviour of bookkeepers are consistent. This leads to the phrase "the books balance". It requires separation of duties: in other words, that subjects cannot take on two roles in the bookkeeping system such that a single subject can commit fraud and ensure that the books still balance.

Further information, including the ability to build your own simple accounting system, can be found here: <https://anvil.works/blog/double-entry-accounting-for-engineers>

## Double-entry bookkeeping found in the Genizah Collection



39

“Jewish bankers in Old Cairo used a double-entry bookkeeping system which predated any known usage of such a form in Italy, and whose records remain from the 11th century AD, found amongst the Cairo Geniza”

[https://en.wikipedia.org/wiki/Cairo\\_Geniza](https://en.wikipedia.org/wiki/Cairo_Geniza)

## 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: authorization from two distinct subjects**

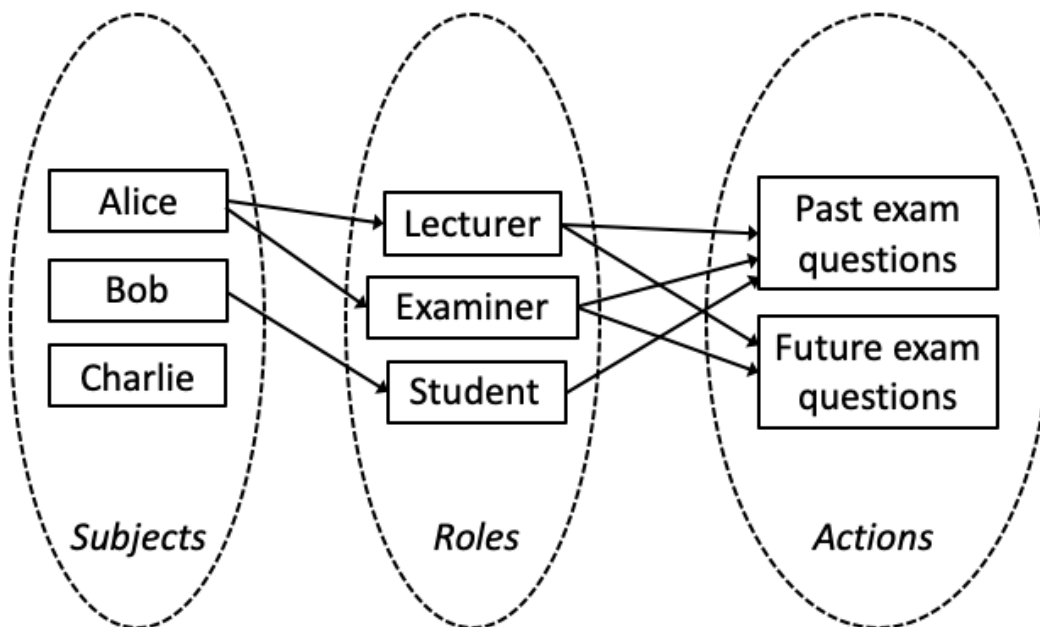
40

Serial or sequential separation assigns a different subject to each role in the double-entry bookkeeping model. In the example, the lecturer interacts with many distinct employees of the University in order to carry out financial transactions. These provide checks against malice or mischance.

An alternative to the sequential approach is the parallel model which requires two subjects to sign an agreement. This might be used for large, irreversible, operations such as signing a guaranteed cheque, or signing a significant contract.



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



41

RBAC adds an extra level of indirection between the subjects and the actions a subject can carry out. We call these intermediate states *roles*. Roles provide flexibility. For example, in a university with 20,000 people, you can't set up individual accounts with separate Unix file permissions for each person. Instead, you define roles and then assign roles to people. This way the complexity is manageable. Example roles in other domains might be Officer of the Watch in the Army, Branch Manager in a banking context, Charge Nurse at a hospital, and so on.

RBAC helps with complexity, but doesn't remove it entirely. You still need to write the policy, and this policy might be wrong.

It is also possible to combine RBAC with other models. For example, SELinux offers MLS with RBAC.

# Summary of security and safety

- 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, ...

42

First, define what you are trying to do at a high level. Some examples:

1. Stop a story reaching the front page of The Guardian
2. Prevent a Branch Manager from running off with the cash
3. Etc.

Given an overarching aim, then construct a protection profile or high-level safety case. What are the threats? Look at the failure modes (bottom-up) or conduct a fault-tree analysis (top-down). This will allow you to construct a detailed security target or safety case, which will inevitably involve consideration of specific *mechanisms*.

We will now look at different mechanisms, starting with user behaviour.

# Do not ignore user behaviour

- Many systems fail because users make mistakes
- Banks routinely tell victims of fraud “our systems are secure so it must be your fault”
- Most car crashes are user error; yet we now build cars with crumple zones

43

This is the most overlooked mechanism. Serious accidents can occur by ignoring the limitations of humans...

Example: Tell the user to choose a password that can't be guessed and don't write it down. Then ask them to create a different password for each of the hundreds of websites that they use (some infrequently). Then say you must change each of these passwords every three months. This is simply not an acceptable cognitive load on an individual, so coping mechanisms arise. Can you think of specific examples of poor understanding of user behaviour in computer systems? What are your coping mechanisms?

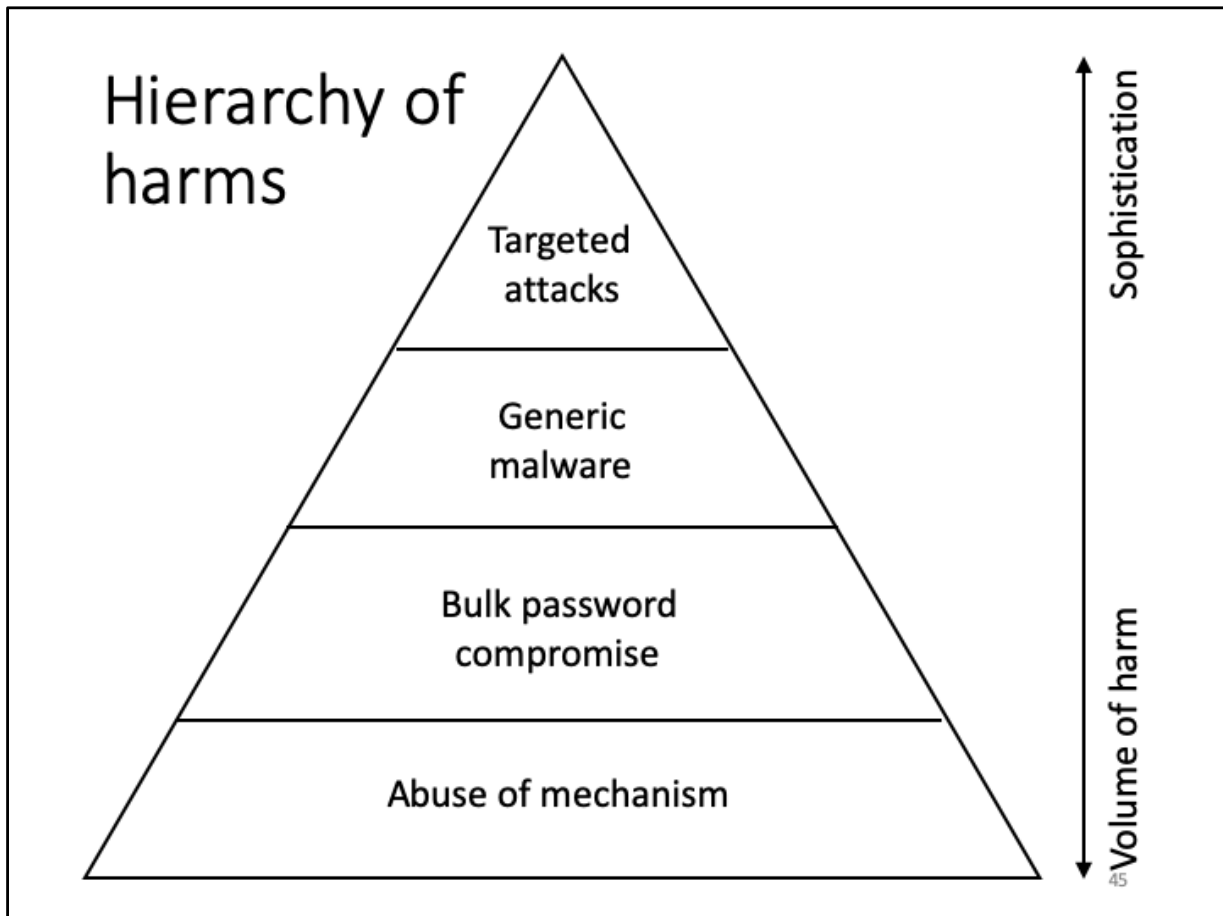
## Chevrolet 1959 vs 2009



<https://www.youtube.com/watch?v=fPF4fBGNKOU>

44

For many years the car industry prospered while blaming the user: "it's not our car which is defective, but the driver...". The mantra from the manufacturers was "sue the driver not the manufacturer". It wasn't until the 1960s that the users managed to challenge this state-of-affairs. It took many decades for the motor industry to get to this point. This sort of dumping of failures onto the user happens time and again as new industries start, and the computer industry is no exception.



Cybercrime today makes up around half of all crime by both cost and volume.

The hierarchy of harms: targeted attacks, generic malware, bulk password compromise, abuse of mechanism. With each step down in this hierarchy, the number of victims goes up by an order of magnitude or more. Most harm occurs at the bottom, and perhaps not too surprisingly these attacks are also the least technically sophisticated.

At the top are the targeted attacks (e.g. Russian intelligence get access to Hilary Clinton's email), but these are really rare. Bulk malware, such as Zeus or Dridex, infects millions of users. Bulk password compromise for 50-100 million people. Perhaps cracking the password file and guessing that the users use the same passwords for Gmail and provides access to 100,000 email accounts.

At the bottom is abuse. Abuse of mechanism can exist in all systems; examples include cyberbullying. What's this got to do with car crashes? Car crashes are an abuse of the mechanisms provided; just the same as cyberbullying where messaging platforms are used to bully others. What responsibility do the messaging platforms have here?

# Many abuses of mechanism

- Cyberbullying
- Doxing
- Fake rental apartments
- ...

How can we protect against these attacks?

46

Cyberbullying. Example: using existing messaging platforms to bully others

Doxing. Researching then broadcasting private information of the victim.

Fake rental apartments. We have seen websites advertise apartments in Cambridge which either don't exist or rather they aren't for rent and then you cheat out of the deposit. What can the website do to push back on this? What can the University do? Well, the University could write to all accepted applicants and tell them to use the official accommodation service and warn them of the scams... but this still doesn't work. Indeed we need the ideas -- this is very much an unsolved problem.

# Useable privacy is also hard

- Traditional approaches – anonymisation and consent – are really hard to deliver
- Problem gets harder as systems get larger
- Automated data collection (e.g. from sensors) makes the situation more difficult again

47

Hierarchy of harms is often focused on security or safety. Privacy also needs to be usable.

The traditional solutions: informed consent and anonymisation.

Consent is hard to do right – how do you know users made informed decisions? Is there really genuine choice on current platforms with their often impenetrable privacy notices?

Anonymisation aims to remove personally-identifying information from a dataset while still preserving utility. Unfortunately this is extremely difficult. Consider, for example, location data. It turns out that where you live and where you work is often unique, so "anonymous" traces of the movements of people can be reidentified simply by combining the location trace data with the electoral roll (home location) and employee database (where you work).

Further reading: Golle, Philippe, and Kurt Partridge. "On the anonymity of home/work location pairs." *International Conference on Pervasive Computing*. Springer, Berlin, Heidelberg, 2009. [https://link.springer.com/chapter/10.1007/978-3-642-01516-8\\_26](https://link.springer.com/chapter/10.1007/978-3-642-01516-8_26)



How many CPUs in the pictures and how does this system differ from the car? [Ask audience] This is the intensive care ward in Swansea. [Click next to run animation]

Note the difference from the car: In a car they are all on the CAN bus and integrated; in the hospital they are all separate with their own interface.





Here are seven infusion pumps. Note that all the controls are different!

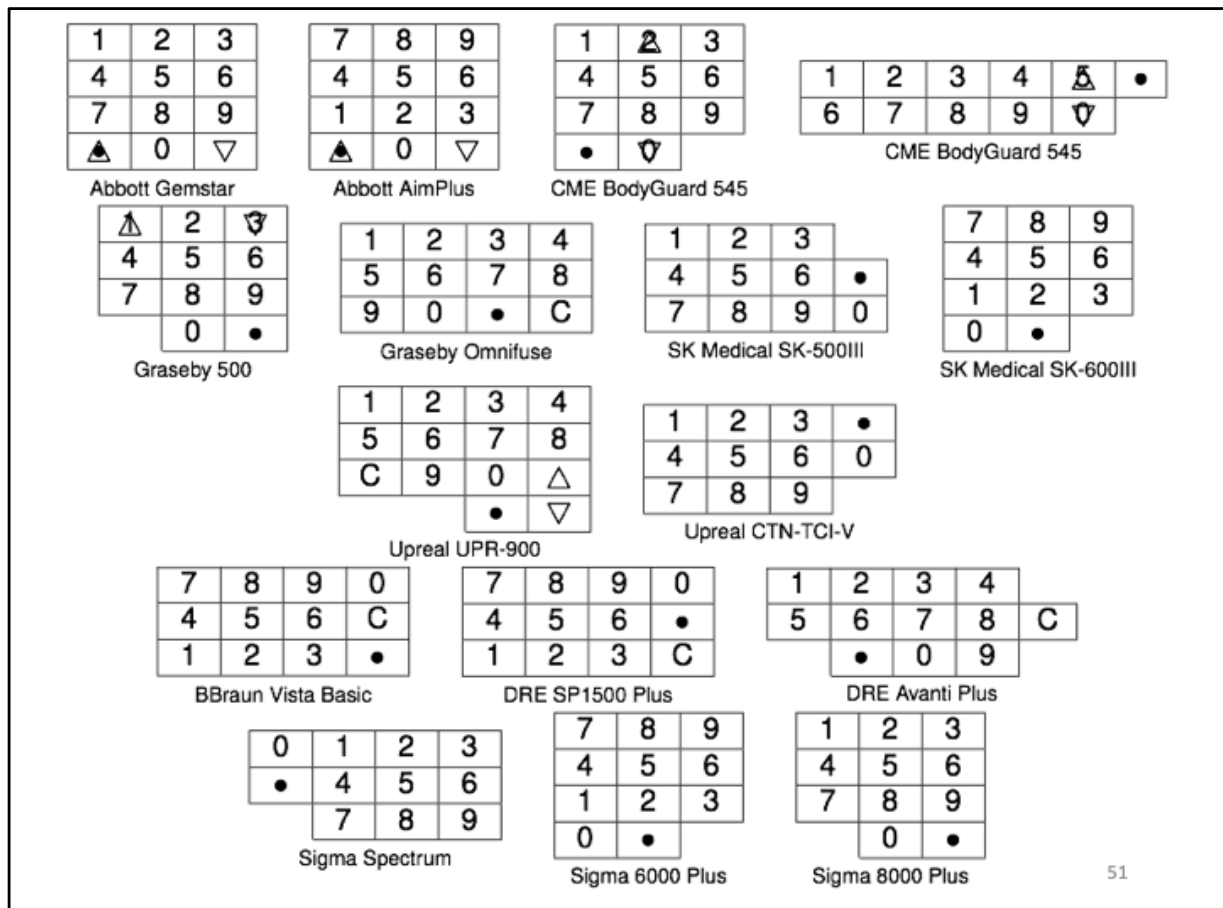
“Approximately 11% of patients in UK hospitals suffer adverse events, of these half are preventable, and about a third lead to moderate or greater disability or death [1]. Medication errors seem to be one of the most preventable forms of error: more than 17% of medication errors involve miscalculation of doses, incorrect expression of units or incorrect administration rates [2]. The Institute of Healthcare Improvement’s “global trigger tool” suggests adverse events may be ten times higher [3]. These figures come from research in different countries with different methodologies and assumptions, and suffer from a lack of reliable information [4], but there is general agreement that preventable mortality is numerically comparable to road accident fatality rates [5].”

Quote from: Thimbleby, Harold. "Improving safety in medical devices and systems." *2013 IEEE International Conference on Healthcare Informatics*. IEEE, 2013. <https://ieeexplore.ieee.org/abstract/document/6680455>

How do we standardise? Even where there are standards, they aren't followed: International standards say that litres should be written with a capital-L, but note that some don't do this (and therefore a lowercase-l could be confused with a one).



Even the same device make and model (here BodyGuard 545) have different versions with different user interfaces.



Here are a range of keyboard layouts from infusion pumps. They are all different. If the layout problems weren't enough, there are also challenges in how key presses are interpreted.

Example problem in this area, although from the banking domain: "In 2008, Grete Fossbakk transferred 500,000 kroner to her daughter using the web interface to her Union Bank of Northern Norway account. Unfortunately, she admits, she miss-keyed her daughter's bank account number and a repeated 5 in the middle of the account number made it too long. The Union Bank of Northern Norway's web site then silently truncated the erroneous number, and this new number (which was *not* the number Fossbakk had keyed) happened to match an existing account number. The person who received the unexpected 500,000 kr spent it. Only on the steps to the court room did the bank relent and refund Fossbakk." (quote from Thimbleby's paper). See also K. A. Olsen, "The \$100,000 keying error," *IEEE Computer*, vol. 41, no. 4, pp. 108–106, 2008.

Similar problems occur with pocket calculators, which are also used by nurses to calculate medical doses.

# 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!

# Bulk password compromise

- Example: in June 2012, 6.5m LinkedIn passwords stolen, cracked (encryption did not have a salt) and posted on a Russian forum
  - Method: SQL injection (see later)
  - Passwords were reused on other sites, from mail services to PayPal.
  - Reused passwords were used on those third-party sites
- There have been many, many such exploits!
- What can we do about password reuse?

53

[Ask the audience to chat to their neighbour and write their own top-5 list of things they would do to prevent password reuse. Discuss.]

## Phishing and social engineering

- Card thieves call victims to ask for PINs
- A well-crafted email sent to company staff, with apparently authority, can get 30% yield
- Some big consequences (see next)
- Think like a crook (see Mitnick reading)

54

Phishing or social engineering aids an attacker at all levels in the hierarchy of harms. An example story. A malicious person finds a card in the street and phones up the owner and says "Hi it's Barclay's here... We have noticed your card was used incorrectly in a number of transactions, have you still got your card?". "I'm sorry, I lost it in Tesco 30 minutes ago, I was going to ring you!". "No problem, we can you sort it out for you, could you just tell us your PIN so we can cancel your card for you?". Sending a generic email to all employees, perhaps purporting to be from the boss, with a URL in it can be very effective. The email simply needs a cover story. This could be anything from "please complete the staff survey" to "your inbox is full; please click here to increase your quota".

Another example story, this time spearphishing. An attacker compromises an email server and finds the inbox from the CEO. The attacker then examines the recent email traffic, and works out who the financial controller (chief clerk) is and also what recent business is going on which might lead to transfers of cash out of the company. The attacker then crafts an email purportedly from the CEO to the financial controller asking him or her to send a large sum of money to a plausible sounding company whose bank details are actually under the control of the company.

The Chinese Government wanted to infiltrate the Dali Lama's office and ended up compromising 30 out of the 50 or so computers. The way in appears to have been a compromise of one Monk's computer. This machine was used to compromise the mail server used by the Monks, so that when an attachment is sent from one Monk

to another it could be rewritten to include malware in the attachment and therefore spread the attack to more machines . This is great for covert ops – all mail sent and received is legitimate, so the Monks can check with each other to see if they did send the email (which they did) but it's still malicious!

# Software and Security Engineering

Lecture 3

**Alastair R. Beresford**

arb33@cam.ac.uk

*With many thanks to Ross Anderson*



Warm-up: Write down your own top three pieces of password advice

- Talk to your neighbour
- What password advice would you give and why?

## John Podesta email compromise by Fancy Bear (allegedly Russia)

- White House chief-of-staff; chair of Hillary Clinton's 2016 US Presidential Campaign
- Gmail account was compromised
- 20,000 emails subsequently published by WikiLeaks
- Authenticity of some emails questioned

57

“In March 2016, the personal Gmail account of John Podesta, a former White House chief of staff and chair of Hillary Clinton's 2016 U.S. presidential campaign, was compromised in a data breach, and some of his emails, many of which were work-related, were stolen. Cybersecurity researchers as well as the United States government attributed responsibility for the breach, which was accomplished via a spear-phishing attack, to the hacking group Fancy Bear, allegedly affiliated with Russian intelligence services.

“Some or all of the Podesta emails were subsequently obtained by WikiLeaks, which published over 20,000 pages of emails, allegedly from Podesta, in October and November 2016. Podesta and the Clinton campaign have declined to authenticate the emails. Cybersecurity experts interviewed by PolitiFact believe the majority of emails are probably unaltered, while stating it is possible that the hackers inserted at least some doctored or fabricated emails. The article then attests that the Clinton campaign, however, has yet to produce any evidence that any specific emails in the latest leak were fraudulent. A subsequent investigation by U.S. intelligence agencies also reported that the files obtained by WikiLeaks during the U.S. election contained no "evident forgeries".”

[https://en.wikipedia.org/wiki/Podesta\\_emails](https://en.wikipedia.org/wiki/Podesta_emails)

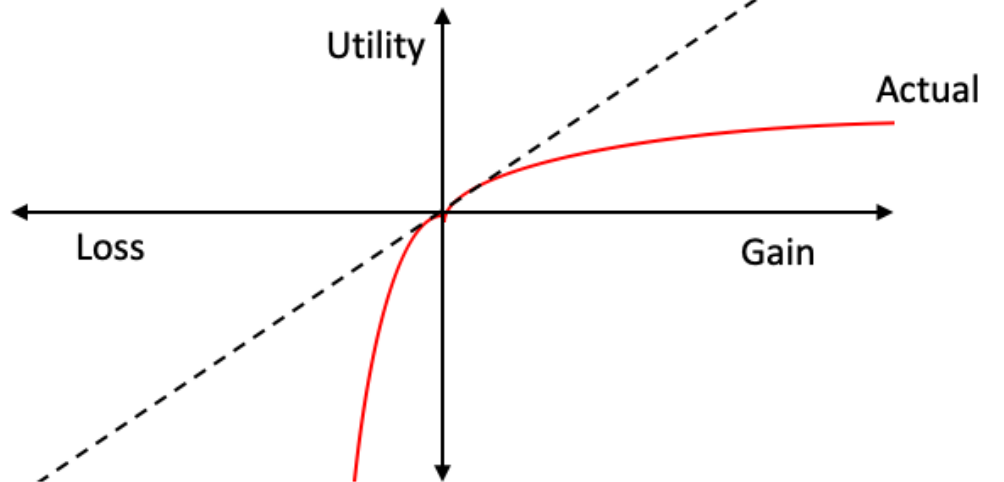
# Cognitive factors

- 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 give way to skill
- Our ability to automate routine actions leads to absent-minded slips, or following the wrong rule
- There are also systematic limits to rationality in problem solving – so called *heuristics* and *biases*

58

It turns out the psychology is really important in this space. Once you become skilled at something, such as playing the Piano, you start to do things automatically (e.g. play the D major scale). The ability to automate familiar actions can be used against us. For example, if you're the Chief Clark and you get a request to pay an invoice from the CEO, and it has similar phrasing and so as the last N emails, then you arrange for payment of the invoice as requested; you don't stop and consciously consider whether this is in fact part of a spear-phishing attack.

## 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!

59

“Asymmetry between gains and losses: People are risk averse with respect to gains, preferring a sure thing over a gamble with a higher expected utility but which presents the possibility of getting nothing. On the other hand, people will be risk-seeking about losses, preferring to hope for the chance of losing nothing rather than taking a sure, but smaller, loss (e.g. insurance).”

“Threshold effects: People prefer to move from uncertainty to certainty over making a similar gain in certainty that does not lead to full certainty. For example, most people would choose a vaccine that reduces the incidence of disease A from 10% to 0% over one that reduces the incidence of disease B from 20% to 10%.”

[https://en.wikipedia.org/wiki/Risk\\_perception](https://en.wikipedia.org/wiki/Risk_perception)

## Framing decisions about risk, or the *Asian disease problem*

Scenario A, choose between:

- 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).

Scenario B, choose between:

- 1) "400 will die"
- 2) "with  $p = 1/3$ , no-one will die,  $p=2/3$ , 600 will die"

Here 78% prefer (2) over (1)

60

Risk misperception: empirical studies have shown that "a bird in the hand is worth two in the bush". Modern prospect theory explains the irrationalities that humans have when it comes to risk. This can be used to manipulate. Decisions are heavily influenced by framing. The Asian disease problem is one of the most famous. Here 600 people are infected with a deadly, fictional disease. The numbers and percentages come from Tversky and Kahneman (1981) with a summary here: [https://en.wikipedia.org/wiki/Framing\\_\(social\\_sciences\)](https://en.wikipedia.org/wiki/Framing_(social_sciences))

[Present the details on the slide]

The ability to framing decisions to change perception is why marketers talk about a 'discount' or 'saving' while fraudsters exploit the fact that people facing losses take more risks. There is more on this in the Economics, Law and Ethics course next year.

# 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: give a gift, to increase your chance of receiving one

61

Milgram showed that the lab setting with a white coat gave authority and many participants would do as directed, even when they ask participants to electrocute students who get answers wrong. The student was actually an actor; no electricity involved. [https://en.wikipedia.org/wiki/Milgram\\_experiment](https://en.wikipedia.org/wiki/Milgram_experiment)

Most people will follow the herd: here seven actors and one subject look at two lines, A obviously longer than line B. Yet when the seven actors say that line B is longer, the subject will follow them and confirm B is longer.

[https://en.wikipedia.org/wiki/Asch\\_conformity\\_experiments](https://en.wikipedia.org/wiki/Asch_conformity_experiments)

Reciprocation: even monkeys do tit-for-tat. Further information:

[https://en.wikipedia.org/wiki/Reciprocity\\_\(social\\_psychology\)](https://en.wikipedia.org/wiki/Reciprocity_(social_psychology))

For further information on these and other areas, see Robert B. Cialdini, Influence: Science and Practice (ISBN 0-321-18895-0).

[https://en.wikipedia.org/wiki/Influence:\\_Science\\_and\\_Practice](https://en.wikipedia.org/wiki/Influence:_Science_and_Practice)

# 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

Note: the *mark* is the person being defrauded

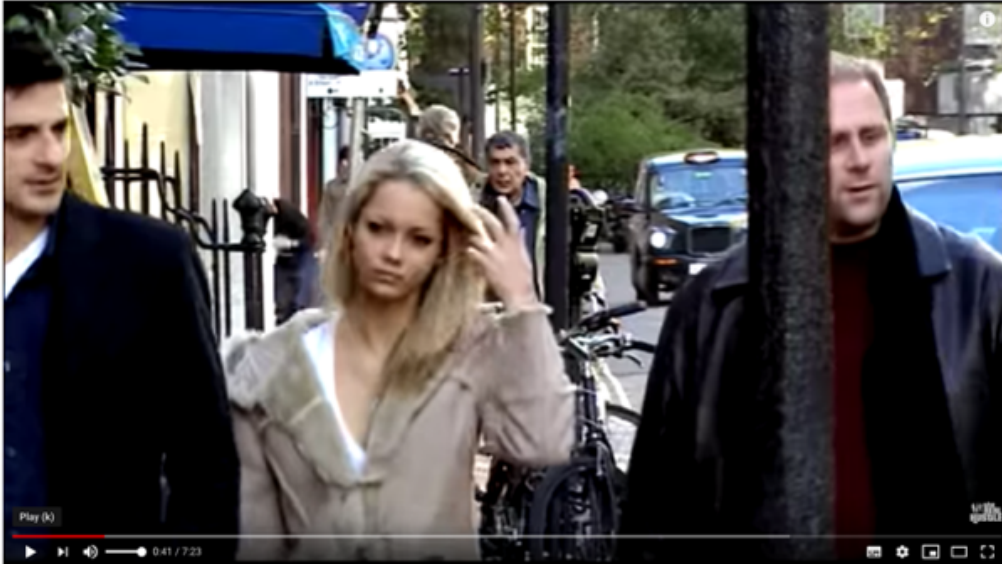
62

Note: the mark is someone who is destined to be defrauded.

- Down a pub: "I'm a bit short of cash, so I wonder whether you could do me a favour and buy this a TV for £40?" (Clearly TVs cost more than this.)
- Via email: "I need help safeguarding \$400 million from ..."
- Sales training school: "if you need someone to sign on the line for something, you put the pen on the clipboard, push the clipboard towards the person who is doubtful and "accidentally" drop the pen off the clipboard towards the mark, who then catches it. Now the mark has the pen in their hand and they are more likely to sign.

For further reading see: Stajano and Wilson, Understanding scam victims: seven principles for systems security, University of Cambridge Computer Laboratory Technical Report 754. <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-754.pdf>  
Also, search for "The Real Hustle" videos on YouTube.

# The Lottery Scam



<https://www.youtube.com/watch?v=ol2gBLn6CU8>

63

[Start playing at 40 seconds in. Run for around two minutes, then explain the remainder.]



## People only follow advice which confirms their own world view

- Users have different mental models. Explore how your users see the problem – the ‘folk beliefs’
- Given a model of their world view, target approach to appeal to it.

64

Therefore you often need to offer one than more piece of advice in order to find the one which fits their own view.

# 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<sup>1</sup>  
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

65

Alma Whitten asked 12 subjects who had no previous experience of public-key cryptography to use PGP to conduct a simple (encrypted) email exchange. Results were poor: participants in the experiment could not get things right. They sent the private key to the corresponding person by mistake. They forgot to encrypt and sign. And so on. The essence of the problem is that PGP was simply not useable by the non-expert (and likely some experts too).

See: A Whitten, JD Tygar. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. USENIX Security Symposium, 1999.

[https://www.usenix.org/legacy/events/sec99/full\\_papers/whitten/whitten.ps](https://www.usenix.org/legacy/events/sec99/full_papers/whitten/whitten.ps)

# The power of default

Most people don't opt in or out; they go with default

Can exploit this for good (or evil):

- Pensions
- Privacy settings in an online service
- Use of crypto
- ...

Therefore defaults may be contentious

66

In the past, many people didn't enrol in a pension scheme because they never got around to going to payroll to set it up or sign up with a third-party provider. Pensions are now offered by default, but you can opt out, which sets a safer default for everyone (less poverty in old age which requires support from the state).

There are also conflicts of interest: engineering defaults might suggest one approach (no crypto means less CPU load) and security requirements might suggest another (crypto protects passwords as we shall see later). Similarly, advertising performance, and therefore revenue, might suggest no HTTPS so adverts work better. There are tensions and these are hard to resolve.

# Economics versus psychology

*Most people don't worry enough about computer security, and worry too much about terrorism*

How could we fix this, and why is it not likely to be?

67

Two approaches to terrorists: 1) play it up as much as possible (e.g. George Bush Jr) and say "woe is us, this is terrible, we must invade these countries, ..." Or 2) this is terrible but we will get them in the end (e.g. George Bush Sr).

If we want to reduce the effect of terrorism, then you need to make it less salient: remove the guns and visible security in airports, etc. Replace guns with pastel sofas, and so on.

## The compliance budget

- '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
- Violations of rules also matter: they're often an easier way of working, and sometimes necessary
- The 'right' way of working should be easiest: look where people walk, and lay the path there

68

Further reading: Beutement, Adam, M. Angela Sasse, and Mike Wonham. "The compliance budget: managing security behaviour in organisations." *Proceedings of the 2008 New Security Paradigms Workshop*. ACM, 2009.  
<https://dl.acm.org/citation.cfm?id=1595684>

## Where should the path be?



69

Make the easiest path also the one which is safe and secure. Otherwise, people will do this...

# Differences between people

- Ability to perform certain tasks varies widely across subgroups of the population, including by
  - Age
  - Gender
  - Education
  - ...
- Yet all customers receive complex password rules and anti-phishing advice

70

When you work at a new tech start-up, it's very easy to assume that everyone is 20, has 20-20 vision and has a degree in computer science. This leads to the situation where you say "use a randomly generated password on each website; don't write them down". However most of the population will struggle with this guidance. Indeed performance at tasks varies significantly across the population. Sometimes there is correlation with age (e.g. due to physical mobility or vision requirements) or gender (in societies with gendered interest in IT).

## More accidents with Volvos?



Volvo ÖV 4, April 1927

71

Volvos have a reputation for safety. So, why are there more accidents involving more Volvo drivers? Two possible explanations: (1) Bad drivers buy Volvos; (2) Volvo drivers drive faster because they think that they are protected and safe in a Volvo. It's really hard to tell. This is called risk compensation.

Other examples: "It has been observed that motorists drove faster when wearing seatbelts and closer to the vehicle in front when the vehicles were fitted with anti-lock brakes. By contrast, shared space is a highway design method which consciously aims to increase the level of perceived risk and uncertainty, thereby slowing traffic and reducing the number of and seriousness of injuries."

[https://en.wikipedia.org/wiki/Risk\\_compensation](https://en.wikipedia.org/wiki/Risk_compensation)



# Understanding error helps us build better systems

- Significant psychology research into errors
- Slips and lapses
  - Forgetting plans, intentions (strong habit intrusion)
  - Misidentifying objects, signals
  - Retrieval failures (“its on the tip of my tongue”)
  - Premature exits from action sequences (using the ATM)
- Rule-based mistakes; applying the wrong procedure
- Knowledge-based mistakes; heuristics and biases

72

Human brains exhibit a number of different errors. We need to understand these if we are to build robust, human-centred systems.

Strong habit intrusion: When I cycle to the train station from the Computer Lab, I often find myself turning into Queens’ College gates on the way there. The reason for this is because I frequently cycle from the CL to Queens’ so I do this by default. I’m “on autopilot”.

When you go to the cash machine should you give the customer the cash then their card (US); or card then cash (UK). Cash second is best: that's why you went to the machine in the first place, therefore you will leave once you have the cash (and leave the card behind). Men are more likely to be goal focused than women; which means that men are more likely to leave their card in the ATM in the US than women.

## Training and practice reduce errors

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	$\sim 1$

73

The automotive industry carried out an analysis into whether training and practice reduce errors. While training does help, there are inherent limits on the ability to reduce the probability of an error. Somewhat predictably, the hardest tasks to perform without error are those which involve creative thinking and unfamiliar operations when time is short.

## Passwords are cheap, but...

- Will users enter passwords correctly?
- Will they remember them?
- Will they choose a strong password?
- Will they write them down?
- Will the password be different in each context?
- Can the user be tricked into revealing passwords?

74

Passwords are great for companies and implementers. They are cheap and users (think) they know how to use them. Important for innovation, since you can grow the user base of an online platform with tiny marginal cost and without the requirement to provide an additional hardware.

It's not helpful to (effectively) say "Choose something you can't remember, and don't write it down". The alternative, which is now considered best practice, is to use a password manager integrated into the web browser and smartphone and able to generate strong random passwords; it has its own drawbacks however – a significant issue is backup. Two factor makes things significantly stronger, however it is less usable. If password manager is not a usable solution, separating accounts into (the few) high-value ones (e.g. bank, email) and the (many) low-value ones and ensuring each class of accounts has a separate password is better than using the same password everywhere.

For further information read: Joseph Bonneau, Cormac Herley, Paul C. van Oorschot and Frank Stajano "The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes" In Proc. IEEE Symposium on Security and Privacy 2012. Extended version available as University of Cambridge Computer Laboratory Technical Report UCAM-CL-TR-817. See: <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-817.html>

## User studies are important

Experiment to see if first-year NatScis could be trained to use passwords effectively. Three groups:

- 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  $Y = G > C$


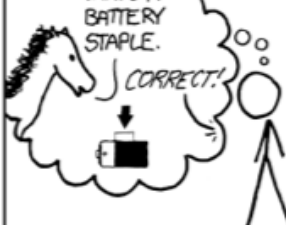
Expected resets:             $Y > G > C$ ; got  $Y = G = C$

*We had 10% non-compliance*

75

Twenty years ago Ross Anderson and Alan Blackwell ran a simple experiment. Split NatSci students into three groups: a control group, a group told to use a memorable phrase, and a group told to choose 8 characters at random. 10% non-compliance is amazing: these students volunteered to take part in an experiment, they are scientists, keen, and yet they didn't do as instructed. Take-home message: if you want to find things out you need to do a proper randomised control trial with real people. We would never have guessed that 10% would be in non-compliance.

Further reading: Jianxin Yan, Alan Blackwell, Ross Anderson and Alasdair Grant. The memorability and security of passwords – some empirical results. University of Cambridge Computer Laboratory Technical Report 500, September 2000. <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-500.pdf>

<p>UNCOMMON (NON-GIBBERISH) BASE WORD</p> <p>ORDER UNKNOWN</p> <p>Tr0ub4dor &amp; 3</p> <p>CAPS? COMMON SUBSTITUTIONS NUMERAL PUNCTUATION</p> <p>(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT '0' IS ONLY ONE OF A FEW COMMON FORMATS.)</p>	<p>~28 BITS OF ENTROPY</p> <p><math>2^{28} = 3 \text{ DAYS AT } 1000 \text{ GUESSES/SEC}</math></p> <p>(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STORED HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)</p> <p>DIFFICULTY TO GUESS: <b>EASY</b></p>	<p>WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?</p> <p>AND THERE WAS SOME SYMBOL...</p>  <p>DIFFICULTY TO REMEMBER: <b>HARD</b></p>
<p>correct horse battery staple</p> <p>FOUR RANDOM COMMON WORDS</p>	<p>~44 BITS OF ENTROPY</p> <p><math>2^{44} = 550 \text{ YEARS AT } 1000 \text{ GUESSES/SEC}</math></p> <p>DIFFICULTY TO GUESS: <b>HARD</b></p>	<p>THAT'S A BATTERY STAPLE.</p> <p>CORRECT!</p>  <p>DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT</p>

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.

NB: NIST only recently rescinded the advice to change passwords regularly.

## Hardware and online support to limit brute force is challenging

- Online services and tamperproof hardware can be used to limit brute-force guessing, such as
  - Bank card PIN (3 guesses on card; 3 online)
  - iPhone PIN (timeouts)
  - Login attempts to webservices (timeouts; care required)
  - ...

**If the typical person has five cards with the same PIN, how many wallets do you need to find before you get lucky?**

77

If passwords are easy to brute-force by repeated guessing, can we limit the number of possible attempts? Note that this is simply not possible in some settings (e.g. encrypted data stored on a stolen harddisk). Success or otherwise depends on the distribution of PINs and passwords (likely distinctly not random) and the number of accounts, cards or devices you have at your disposal.

For example, if you find a wallet in the street with five cards in it, assuming all PINs are the same, what is the chance that you can guess the PINs before cards start getting blocked (e.g. inside 25 attempts). People tend to pick PINs which are easy to remember! Could be 1234, could also be year of birth of children (e.g. 2002). Therefore you need many fewer wallets in reality than random guessing would suggest; 11-18 wallets turns out to be enough.

For more information see: Bonneau, Joseph, Sören Preibusch, and Ross Anderson. "A birthday present every eleven wallets? The security of customer-chosen banking PINs." *International Conference on Financial Cryptography and Data Security*. Springer, 2012. [https://link.springer.com/chapter/10.1007/978-3-642-32946-3\\_3](https://link.springer.com/chapter/10.1007/978-3-642-32946-3_3)

## Mitigate worst effects of a stolen password file

- Use key stretching techniques such as PDBKF2:

```
public PBEKeySpec(char[] password, byte[] salt,  
                 int iterCount, int keyLength)
```

- Establish breach reporting laws
- Externalise the problem with Oauth
- Use other factors to determine whether login legit

78

Defence in depth is important (see Swiss Cheese Model earlier). What can we do to limit harm if the password file is stolen? This is important since users often reuse passwords across websites and apps, and email addresses are typically used as the username and therefore are also likely to be the same across multiple sites. Example: use `javax.crypto.spec.PBEKeySpec`

Rather than storing any passwords in plaintext, use a cryptographically secure one-way hash function. This means that, given a hash of the password, you cannot determine the plaintext. To check whether a password is valid, simply hash the user-supplied password and compare with the hash version previously stored. Given that there are a small number of potential passwords that many people use, a hash function alone is not very secure – an attacker could pre-compute the hashes of many common passwords to allow easy inversion at scale. Therefore, store a per-user cryptographic salt (random number) along with the hashed value. This means any inversion table needs to be built *per-user*, which does not offer any performance benefit.

A breach reporting laws says that the breach must be reported to the individuals who have been compromised. Users can then take action. This also means that other companies can find out about it when the breach is large (they are individuals too). Therefore these third-party companies can take appropriate action as required.

Oauth offers a potential solution since you no longer have to store passwords. Sounds

great in principle, but it then means the website's operation is reliant on a third party. No third-party, no access to any accounts. It also means that if the Oauth vendor is hacked, your site is compromised. A related example: banks rely on SMS as second factor, so go to phone company, pretend to be the customer, and get a new SIM issued. Privacy is a problem with Oauth: Facebook knows how many customers you have if you use Oauth for authentication; bad for users and also bad for you when you try and sell the company to Facebook -- they know how often customers log in and how long there on the site (e.g. with "like" buttons).

Authentication is no longer a binary yes/no, but good systems use lots of side-information (e.g. location of login, speed of typing, etc) . Authentication systems get benefits of scale, thus encouraging use of centralisation (e.g. with Oauth) since smaller sites simply don't have the expertise, data and dedicated security team.



# Password recovery is a weak point

- Password recovery often involves basic info which doesn't change:
  - What was the name of your first school?
  - What was the name of your first pet?
  - ...
- Little ability to change this information
- Accounts for public figures are especially vulnerable

79

Famously Sarah Palin's AOL account got hacked because password recovery was poor: the answers to her recovery questions were in the public domain, so access to her email was obtained through public data.

# A poor implementation of password recovery...

Answer Security Questions

What is your phone number?

Your answer cannot contain repeating characters.

*"I did it. I found the all-time dumbest security question answer requirement. Good job [@fedex](#)."*  
Luke Millar (@ltm on Twitter), 28<sup>th</sup> April 2019

80

Source: <https://twitter.com/ltm/status/1122290624940560385>

## Externalities need consideration

- 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 or redirecting the browser to third-party domains for payment
- Much 'training' amounts to victim blaming

81

It's not enough to look at things in isolation. For example, people are on Facebook because their friends are on Facebook – the so-called network effect. Similarly, compromise of one website results in a compromise of another website because the passwords for many users are the same.

## Iterative guessing of card details with botnet on websites works

- Of Alexa top 500 websites, 26 use Primary Account Number (PAN) and expiry date
- 37 use PAN + postcode (numeric digits only for some, add door number for others)
- 291 ask for PAN, expiry date and CVV2

There is enough variation in requirements across websites that you can iteratively generate valid credentials

82

This is an externalities issue because you can first go to sites which require an account number and expiry date and use these to find valid combinations of these by brute force. Some sites require some of the postcode, so you can then guess this by using several such sites, and so on.

“We came to the important observation that the difference in various websites' security solutions introduces a practically exploitable vulnerability in the overall payment system. An attacker can exploit these differences to build a distributed guessing attack that generates usable card payment details (card number, expiry date, card verification value [CVV2], and postal address) one field at a time. Each generated field can be used in succession to generate the next field by using a different merchant's website.” For further information, see: Ali, Mohammed Aamir, Budi Arief, Martin Emms, and Aad van Moorsel. "Does the online card payment landscape unwittingly facilitate fraud?." *IEEE Security & Privacy* 15, no. 2 (2017): 78-86.

<https://ieeexplore.ieee.org/abstract/document/7891527>

MAT HONAN GEAR 08.06.12 08:01 PM

# HOW APPLE AND AMAZON SECURITY FLAWS LED TO MY EPIC HACKING



83

Mat Honan. “In the space of one hour, my entire digital life was destroyed. First my Google account was taken over, then deleted. Next my Twitter account was compromised, and used as a platform to broadcast racist and homophobic messages. And worst of all, my AppleID account was broken into, and my hackers used it to remotely erase all of the data on my iPhone, iPad, and MacBook. In many ways, this was all my fault. My accounts were daisy-chained together. Getting into Amazon let my hackers get into my Apple ID account, which helped them get into Gmail, which gave them access to Twitter. Had I used two-factor authentication for my Google account, it’s possible that none of this would have happened, because their ultimate goal was always to take over my Twitter account and wreak havoc.”

<https://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/>

He lost all of the data stored on his laptop, including all the photos of his one-year old daughter.

## Amazon → Apple ID → Gmail → Twitter

(And all they wanted was his three letter Twitter handle!)

- Twitter: find personal website, then Gmail, home address
- Gmail: account recovery gave “m••••n@me.com”
- Amazon: call with name, address, email to associate a new credit card number (fake) to the account
- Amazon: call (again) with name, address, credit card number and associate new email address with the account
- Amazon: Use web password reset to new email address; get last four digits of all credit cards in the account
- Apple: Call with billing address and last four digits credit card to get temp password for “m••••n@me.com”
- Gmail: reset password sent to “m••••n@me.com”
- Twitter: reset password sent to Gmail

84

Attack worked back from Twitter. Twitter profile listed his personal website, which listed his Gmail address. Whois record against the website provided home address, which is also the billing address for his credit card. The attacker used account recovery on Gmail to reveal backup email address as “m••••n@me.com” which is then guessable given his name.

Then the attacker called Amazon as Hanon and requested to add a credit card to his account. For this the attacker needed the name of the account holder, home address and email address. The attacker had all this. The attacker then provided the credit card information (a fake one will do) which is added to the account; hang up. Attacker called Amazon back and said he'd lost control of Hanon's account, provided the account holder name, billing address and credit card number (the one just added). Amazon then associated a new email address with the account. The attacker then went to the Amazon website, used the reset password link and got the reset password sent to the new email address. The attacker then viewed the last four digits of all credit cards associated with the account.

The attacker then called AppleCare and gave them name, address and last four digits of credit card. This then allowed the attacker to gain access to Matt's Apple ID and control of the iCloud account. From here the attacker reset the Gmail password to the backup email address (“m••••n@me.com”) which the attacker then controlled. From there reset the Twitter account via the Gmail account.

# Social media influencer plotted to take internet domain at gunpoint. It didn't end well



By Faith Karimi, CNN

🕒 Updated 1251 GMT (2051 HKT) April 21, 2019



Rossi Lorathio Adams II

85

(CNN) — The plan was like a bad movie script -- complete with an attacker in a puzzling outfit

Image source: <https://edition.cnn.com/2019/04/21/us/iowa-social-media-influencer-domain-name-trnd/index.html>

You can avoid the difficulties of a technical attack by simply using force. While this doesn't scale well, it might be an effective means of carrying out a targeted attack. Thankfully it's harder than it might at first appear. The same approach has been used for stealing Cryptocurrency: "Robbers order gunpoint Bitcoin transfer after Moulford break-in. Four robbers broke into a house and demanded at gunpoint the occupants transfer Bitcoins into another account. Thames Valley Police said the aggravated burglary happened in Reading Road, Moulford, Oxfordshire, at about 09:40 GMT on 22 January. Four men broke in and threatened the two men and a woman inside with what appeared to be a firearm. One was told to transfer an amount of the digital currency but the transaction failed, police said." <https://www.bbc.co.uk/news/uk-england-oxfordshire-42864053>

# Software and Security Engineering

Lecture 4

**Alastair R. Beresford**

arb33@cam.ac.uk

*With many thanks to Ross Anderson*



## Warm-up: which password hashing solution is the best? Why?

	Alice	Bob	Charlie
Nothing Ltd	123456	qwerty	123456
Hash 1 Ltd	a832gsl47g...	84hskubvg...	a832gsl47g...
Hash 2 Ltd	a832gsl47g...	84hskubvg...	a832gsl47g...
Global Salt Plc	<i>salt: h3okl...</i> <i>hash: slau44...</i>	<i>salt: h3okl...</i> <i>hash: klas3...</i>	<i>salt: h3okl...</i> <i>hash: slau44...</i>
Per-User Salt Inc	<i>salt: h3okl...</i> <i>hash: glhy5...</i>	<i>salt: 9shk4...</i> <i>hash: zay4a...</i>	<i>salt: 0ag3b...</i> <i>hash: lav1za...</i>

87

Alice, Bob and Charlie use the same username and password combination with five companies. Nothing Ltd uses no hashing and stores the passwords in plain text. Hash 1 and Hash 2 Ltd apply a hash function to the password. Global Salt Plc applies a global salt. Per-User Salt Inc applies a per-user salt to each password entry. Which one has the best password management strategy? What are the externalities?

# Security protocols

- Security protocols are another 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...

88

Adversarial thinking is really important: there are lots of weaknesses which you can exploit which don't require pulling fingernails of a customer to get their bank account PIN. The earlier example with Matt Hanon demonstrates the failure of security protocols neatly; stealing a domain name at gun point demonstrates that (metaphorically) pulling fingernails also works.

# Ordering wine in a restaurant

1. Sommelier presents wine list to host
2. Host chooses wine; sommelier fetches it
3. Host samples wine; then it's served to guests

Security properties?

89

[Ask the audience]

Example properties include:

- Confidentiality – of price from guests
- Integrity – can't substitute a cheaper wine
- Non-repudiation – host can't falsely complain

## Car unlocking protocols

Static	Non-interactive	Interactive
$T \rightarrow E: K$	$T \rightarrow E: T, \{T, N\}_K$	$E \rightarrow T: N$ $T \rightarrow E: \{T, N\}_K$

**N:** *nonce*; a sequence number, random number or timestamp

**E:** engine unit

**T:** car key fob or *transponder*

**K:** secret key shared between E and T

$\{x\}_K$  : encrypt  $x$  with  $K$

90

[Introduce the notation; explaining what is on the slide carefully.]

Static suffers from a replay attack: record the transmission of  $K$  and replay to unlock. Additionally, some systems are susceptible to brute-force attacks: some garage door openers still use the static approach with a 16-bit key, so fly a plane over Cambridge spitting out all the combinations in quick succession and watch all those garage doors go up.

The nonce is critical to the success of the other two protocols. A sequence number, a random number or a timestamp are all possible, but they need to be implemented carefully. Random requires us to keep a list of previous numbers to prevent replay attacks; sequence can go out of sync (e.g. dog presses transponder lots of times when out of range) so could look for sequences of two presses, one number apart, which suggests the user is next to the car; timestamp is okay, but problematic if clocks go out of sync or if there are time zone issues.

One problem with interactive is the relay attack. A claim, concerning keyless car keys: Audi's new key contains a motion sensor that shuts off its signal "when the key is laid down and not moving". A similar Porsche device sleeps after 30 seconds and all new Mercedes keys shut down after two minutes.

<https://twitter.com/kentindell/status/1117341970068910080?s=09>

# Identify Friend or Foe (IFF)

- Basic idea: fighter challenges bomber

$F \rightarrow B: N$

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

- What can go wrong?

91

[Ask the audience]

## Person-in-the-middle attack...

- Basic idea: fighter (F) challenges bomber (B)

$F \rightarrow B: N$

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

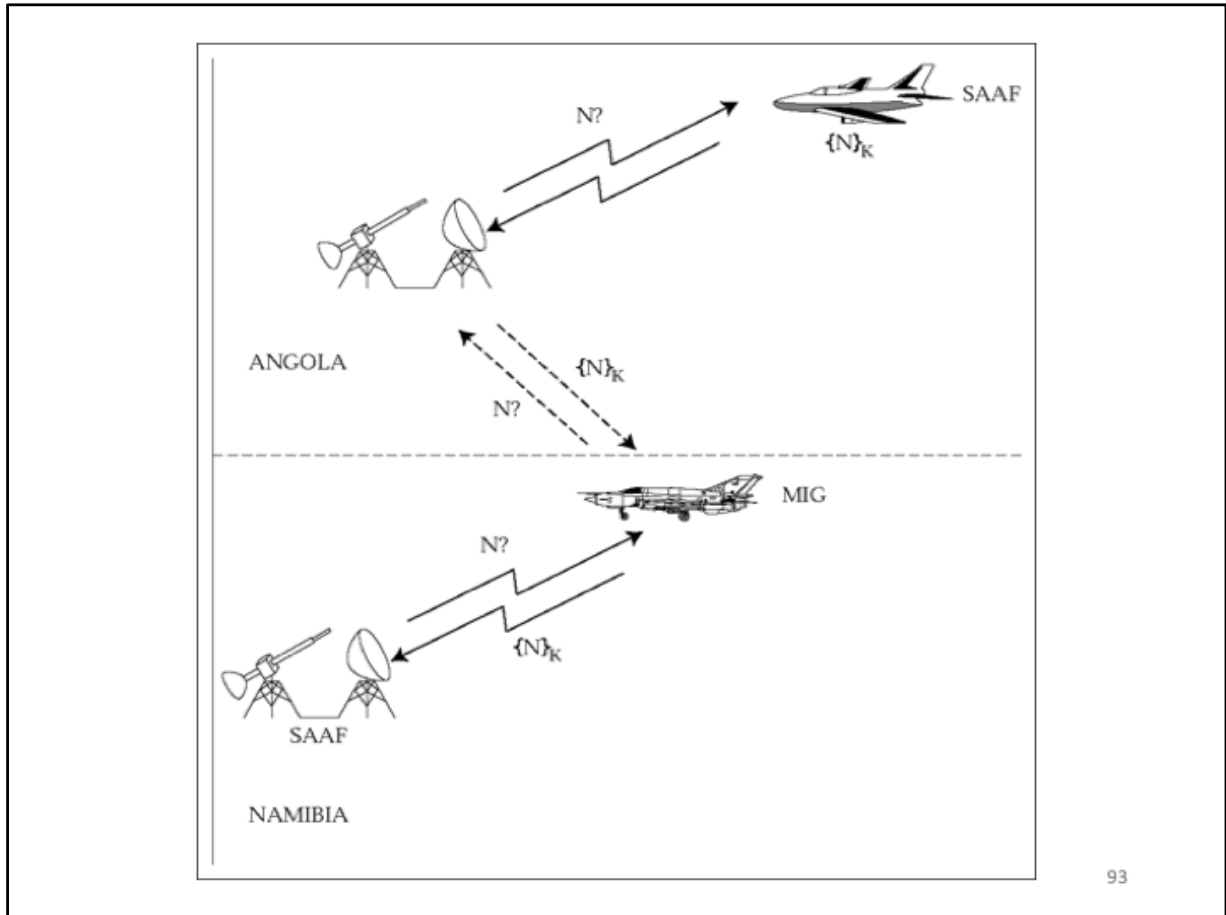
- What if the bomber reflects the challenge back at the fighter's wingman (W)?

$F \rightarrow B: N$

$B \rightarrow W: N$

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

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



This was used against the South African Air Force in the late 1990s, when South Africa were bombing the capital of Angola. Cuba (who were helping Angola) sent in the MIG which relayed IFF to enable access to South African airspace and led to the bombing of an airport in South Africa. More detail in the course text book: Ross Anderson, Security Engineering.

## Two-factor authentication (2FA)

$T \rightarrow U: N$

$U \rightarrow C: N, \text{PIN}$

$C \rightarrow U: \{N, \text{PIN}\}_K$

$U \rightarrow T: \{N, \text{PIN}\}_K$



T: terminal

U: user

C: calculator

K: key known to bank and C

PIN: secret known to bank and U

94

Example from the 1990s. This system provided two-factor authentication where you typed in a challenge from the terminal into the calculator together with the PIN. The calculator then encrypted  $N$  and PIN under key  $K$ .

[Ask audience how they would hack it]

Hacks: steal the calculator; MITM attack; take over a session that is in progress -- the data in the 1990s was not encrypted; infect the terminal with malware; and so on. Nevertheless, this is still much better than just passwords -- attacks don't scale well since you can't just hack into a server and steal all the passwords.



# Card authentication protocol



- Allows EMV cards to be used 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?

95

This is a modern version of the system shown on the previous slide. Note the difference from last one – this new machine tells you whether you have got the PIN right or not. The previous version one would just spit out a random (incorrect) challenge. This appears to be superior in terms of usability until you realise that its popular with muggers. Previously a mugger would have to drag a victim to the cash machine – a risky endeavour; now a criminal can now force people at knife point to reveal and check the PIN wherever the mugging takes place.

# Alice and Bob want to talk. They each share a key with Sam. How?

- Alice contacts 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?

96

This originated from the 1970s where we suddenly had network computers (e.g. at Xerox Parc). Then we want Bob, Alice, and so on to be able to communicate. Also true for other components in the system, including the printer, mail server, and so on. Having every computer or device keep a full list of all keys of everything else is going to be painful. Solution: centralise key management, but then the question is how to avoid all communications going through the central server.

## Kerberos uses tickets to support communication between parties

$A \rightarrow S: A, B$

$S \rightarrow A: \{T_S, L, K_{AB}, B, \{T_S, L, K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

$A \rightarrow B: \{T_S, L, K_{AB}, A\}_{K_{BS}}, \{A, T_A\}_{K_{AB}}$

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

**A:** Alice                      **B:** Resource (e.g. printer)

**S:** Server                      **T<sub>S</sub>:** Server timestamp

**K<sub>AS</sub>:** Secret key shared between A and S

**K<sub>BS</sub>:** Secret key shared between B and S

**K<sub>AB</sub>:** Shared session key for A and B

**L:** Lifetime of the session key

97

We use this for access control in the Computer Laboratory. When I want to access the fileservers, I need to type in kinit before I can access my home directory. This is good for remote access: first connect to `slogin.cl.cam.ac.uk`, where you need an SSH key to get in (something you have) and then a password (something you know) to actually access the fileservers.

[Talk through the protocol in detail.]

There is still some trust here. For example, Alice trusts that Sam sends the right timestamps. This protocol allows things to scale: you can have different ticket granting machines (S) for different departments, and so on. There are a whole series of protocols built on top of this for distributed systems. For now, you just need to know about this protocol as an example. Later lecture courses will cover these type of things better, and also how to prove correctness and so on.

## Europay-Mastercard-Visa (EMV) How might you attack this?

C → M:  $\text{sig}_B\{C, \text{card\_data}\}$

M → C: N, date, Amt, PIN (if PIN used)

C → M:  $\{N, \text{date}, \text{Amt}, \text{trans\_data}\}_{K_{CB}}$

M → B:  $\{\{N, \text{date}, \text{Amt}, \text{trans\_data}\}_{K_{CB}}, \text{trans\_data}\}_{K_{MB}}$

B → M → C:  $\{\text{OK}\}_{K_{CB}}$

C: Card                       $\text{sig}_Y\{x\}$ : message  $x$  digisigned by  $Y$

M: Merchant                 $\{x\}_K$ : Message  $x$  encrypted under  $K$

B: Bank                       $K_{XY}$ : Shared key between  $X$  and  $Y$

98

[Describe protocol. Ask the audience for ideas on how to attack.]

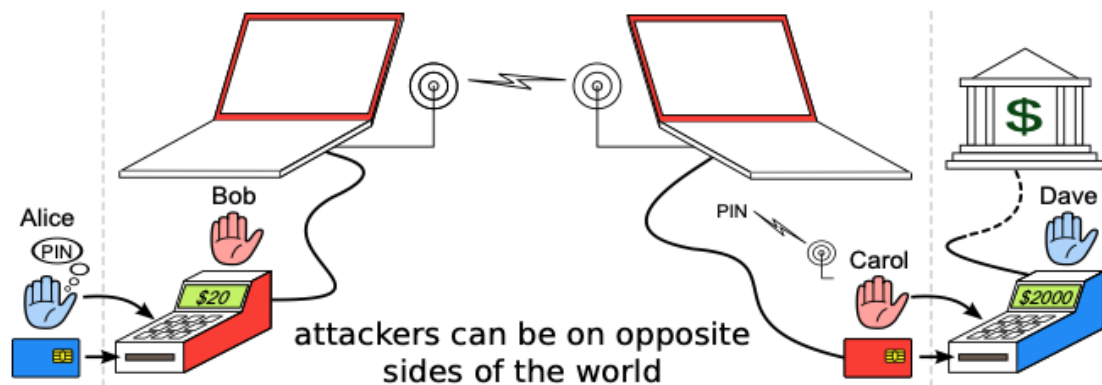
There are lots of attacks which involve replay and pre-plays which we will get to. There were a lot of attacks years ago which involve a wiretap to collect account number from a merchant device, then video PIN being typed in; then you can make a mag stripe clone of the card. Less good now as mag stripe fall back does not work in many countries.

## Replace insides of the terminal with your own electronics



- Capture card details and PINs from victims
- Use to perform person-in-the-middle attack in real time on a remote terminal in a merchant selling expensive goods

## The relay attack: unstoppable but unrealistic – too hard to scale



This attack is almost unstoppable. Steven Murdoch demonstrated this attack 10 years ago: a journalist thought they were buying a coffee, but actually bought an expensive book in another shop. This attack has not been used in real life. It just doesn't scale. The important engineering point here is that flaws need to have scale -- without that they won't be useable.

## Magstripe fraud is scalable



Photo credit: Brian Krebs, krebsonsecurity.com

- Install fake terminal and collect card data and PINs
- Either physically or wirelessly collect data

101

Terminals (PIN entry devices) at Shell garages were doctored by malicious service engineers. Terminal supplier went bust.

Customers at BP garage in Girton in 2008 found their cards cloned and used in Thailand.

These remain big in the US, particularly when you pay at the pump. Further info on petrol pump skimmers: <https://krebsonsecurity.com/tag/gas-pump-skimmers/>

## The no-PIN attack (2010)

$C \rightarrow M: \text{sig}_B\{C, \text{card\_data}\}$

$M \rightarrow \acute{C}: N, \text{date}, \text{Amt}, \text{PIN}$

$\acute{C} \rightarrow C: N, \text{date}, \text{Amt}, \text{No PIN required}$

$C \rightarrow M: \{\text{N, date, Amt, trans\_data}\}_{K_{CB}}$

$M \rightarrow B: \{\{\text{N, date, Amt, trans\_data}\}_{K_{CB}}, \text{trans\_data}'\}_{K_{MB}}$

$B \rightarrow M \rightarrow C: \{\text{OK}\}_{K_{CB}}$

$\acute{C}$ : MITM card shim

C: Card                       $\text{sig}_Y\{x\}$ : message  $x$  digisigned by  $Y$

M: Merchant                 $\{x\}_K$ : Message  $x$  encrypted under  $K$

B: Bank                       $K_{XY}$ : Shared key between  $X$  and  $Y$

102

Apply a MITM attack to the protocol, convincing the card that it has performed a chip and signature transaction, and the terminal that it has performed a chip and PIN transaction. This allows you use a card where you don't have the PIN.

You can now use a SIM shim (140 microns thick!) to MITM the protocol and implement the attack described.



## Fixing the no-PIN attack: simpler protocol required

- In theory might compare card data with terminal data at terminal, acquirer, or issuer
- In practice has to be the issuer since incentives for terminal and acquirer are poor
- Barclays introduced a fix July 2010; removed December 2010. 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

103

Barclays likely removed fix in December 2010 due to too many false positives. It took the banks four years to block this. Some countries still don't.

The EMV spec is 4000 pages thick. This is a real problem as there are lots of interactions between different features. This is good for the bad guys: they can exploit any and all potential feature interactions. It is a disaster for the defender since it represents a huge attack surface which is hard to check.

## 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  $N$ ,  $d$  and  $Amt$  using the key it shares with the bank,  $K_{CB}$
- What happens if I can predict  $N$  for date  $d$ ?
- Answer: if I have access to your card I can precompute an authenticator for  $Amt$  and  $d$

104

Ross provided representation for a Scottish sailor who bought a round of drinks for 33 Euros, and later found he had 4 transactions of 3300 Euros on his card. These four transactions were made one hour apart and placed through three different acquirer banks. When you think about it, you have in your wallet three or four cards, and each card may have £5000 available on it (e.g. because you can get an overdraft, or you have a large credit limit). This means you're walking around with £20k. Would you walk into a dodgy place with £20k in cash in your pocket? The problem is that people don't think this way -- they think that their PIN offers security and their bank will protect them in the case of failure.

# Symmetric key cryptography requires careful sharing of keys

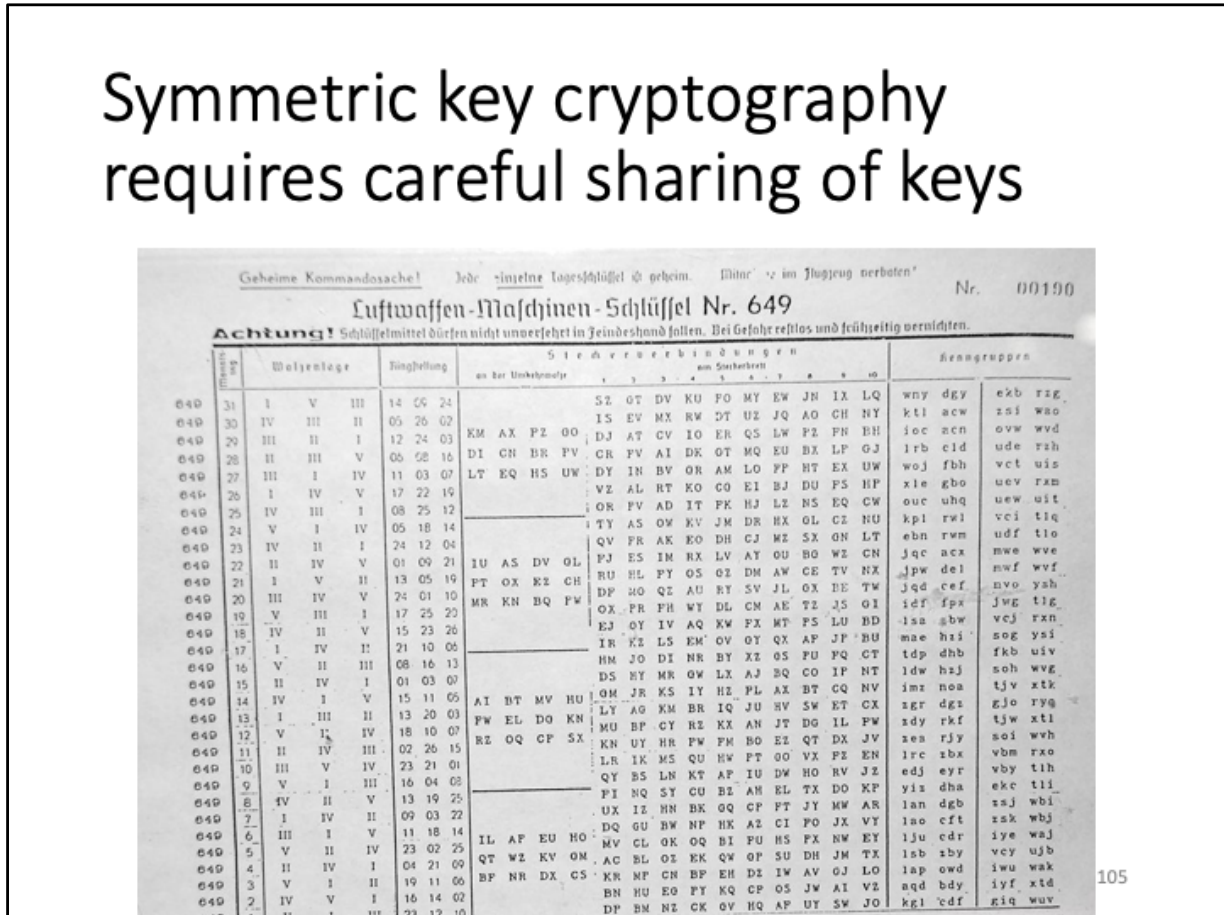


Photo source: [https://commons.wikimedia.org/wiki/File:Enigma\\_keylist\\_3\\_rotor.jpg](https://commons.wikimedia.org/wiki/File:Enigma_keylist_3_rotor.jpg)

A list of keys for a German Enigma cipher machine.

English translation of text along the top (from Wikipedia):

“Secret Command Document! Every individual key setting is secret. Forbidden to bring on aircraft.

Luftwaffe Machine Key No.649

Attention! Key material must not fall into enemy hands intact. In case of danger destroy thoroughly and early.”

# Software and Security Engineering

Lecture 5

**Alastair R. Beresford**

arb33@cam.ac.uk

*With many thanks to Ross Anderson*

106

# Public key cryptography

Allows two parties with no prior knowledge of each other to jointly establish a shared secret key over an insecure channel

Examples include Diffie-Hellman and RSA

## Diffie Hellman revision

Alice and Bob publicly agree to use  $p = 23$ ,  $g = 5$

1. Alice chooses secret integer  $a = 4$ , then  
 $A \rightarrow B: g^a \bmod p = 5^4 \bmod 23 = 4$
2. Bob chooses secret integer  $b = 3$ , then  
 $B \rightarrow A: g^b \bmod p = 5^3 \bmod 23 = 10$
3. Alice computes  $10^4 \bmod 23 = 18$
4. Bob computes  $4^3 \bmod 23 = 18$

Alice and Bob now agree the secret integer is 18

Example derived from [https://en.wikipedia.org/wiki/Diffie-Hellman\\_key\\_exchange](https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange)

108

You saw Diffie Hellman in Discrete Maths. This simple version uses a multiplicative group of integers modulo  $p$ , where  $p$  is prime and  $g$  is a primitive root modulo  $p$ . The values of  $p$  and  $g$  are chosen in this way to ensure that the resulting shared secret can take on any value from 1 to  $p-1$ .

This protocol has a significant limitation: it is susceptible to a person-in-the-middle attack.

## Physical public key crypto with locks

- Anthony sends a message in a box to Brutus. Since the messenger is loyal to Caesar, 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?

109

Anthony wants to kill Caesar, but needs Brutus' help to do so. How can Anthony send a message to Brutus yet not let the messenger read the message? This proposal is insecure: it is vulnerable to a MITM attack, as is Diffie Hellman.

Here Anthony has shared the secret message with someone, but Anthony doesn't know who it is!

# Asymmetric public-key crypto

- Separate keys for encryption and decryption
- Publish *encryption* key widely (the “public key”) allowing anyone to create an encrypted message; only holder of *decryption* key (“private key”) can decode the message and read it
- Digital signatures are the other way around: only you can sign but anyone can verify
- Example: RSA

110

More on this in the Part IB Security course and Part II Cryptography course. Required knowledge at this point is as stated above and expanded on in the lecture.

Note that asymmetric public-key crypto has the same problem as Diffie-Hellman: how do you know that you have the right public key for Alice and you are not subject to a MITM attack?



# Public-key Needham-Shroeder

- Proposed in 1978:

$$\begin{aligned} A \rightarrow B: \{N_A, A\}_{K_B} \\ B \rightarrow A: \{N_A, N_B\}_{K_A} \\ A \rightarrow B: \{N_B\}_{K_B} \end{aligned}$$

- $N_A$  and  $N_B$  are nonces generated by A and B respectively
- $K_A$  and  $K_B$  are public keys for A and B respectively
- The idea is to use  $N_A \oplus N_B$  as a shared key

Is this okay?

111

Once public key crypto is discovered, people then looked for ways to use it. Background: Needham went to California every summer to work at Xerox Parc. He got a preprint of the RSA paper and decided to apply it to the problem on the Xerox network computer project. Kerberos, discussed earlier, was derived from the Needham-Shroeder protocol, and in 1978 Needham proposed the following public-key variant of the protocol.

This version does not require an online server, Sam. Instead the nodes now need the long-term public keys of each other. Here,  $K_A$  and  $K_B$  are the *public* keys of A and B respectively, and the aim is to use these in order to derive a symmetric session key between A and B (symmetric cryptography is computationally cheaper).

## MITM attack found 18 years later

A → C:  $\{N_A, A\}_{KC}$

C → B:  $\{N_A, A\}_{KB}$

B → C:  $\{N_A, N_B\}_{KA}$

C → A:  $\{N_A, N_B\}_{KA}$

A → C:  $\{N_B\}_{KC}$

C → B:  $\{N_B\}_{KB}$

The fix is explicitness. Put all names in all messages.

112

Here Charlie can pretend to be Alice when talking to Bob (line 2). Doing so means that Charlie gets  $N_A$  (line 1) as well as  $N_B$  (line 5) and therefore can compute the shared key between Bob and Alice. Don't beat yourself up with if you didn't spot it. It took 18 years to spot the problem as shown on this slide.

## Binding keys to principals is hard

- Physically install binding on machines
  - IPSEC, SSH
- Trust on first use; optionally verify later
  - SSH, Signal, simple Bluetooth pairing
- Use certificates with trusted certificate authority
  - Sam signs certificate to bind Alice's key with her name
  - Certificate =  $\text{sig}_s\{A, K_A, \text{Timestamp}, \text{Length}\}$
  - Basis of Transport Layer Security (TLS) as used in HTTPS
- Use certificate pinning inside an app
  - Used by some smartphone apps

# Transport Layer Security (TLS)

- Uses public key cryptography and certificates to establish a secure channel between two machines
- Protocol proven correct (Paulson, 1999)
- Yet, the protocol is broken annually
- Often a large number of root certificate authorities. Are these all trustworthy?

114

Earlier versions of this protocol were called Secure Sockets Layer (SSL). There's been around one bug every year in TLS since 1999. The first series of attacks were timing attacks: look at how long it takes a server to respond and use this to determine certain bits of the key. The challenge here is that compilers and security engineers fight. Compilers attempt to make code as fast as possible, and may optimise away “make work” inserted by security engineers who are attempting to ensure constant-time execution for critical operations. There are many more technical hacks here, but these are for later courses.

Another major problem is that it's really hard to fix bugs when found. In order to change the protocol you need to make changes to both the client and the server. This is hard for the Web since you have to upgrade both all web browsers and all web servers and no single party is in control of the overall ecosystem. There are poor incentives. There are 187 root certificates installed on my Mac. Web browsers typically trust all of them, and any of these certificates may be used to license other providers with the power to create further certificates for arbitrary domains.

## DigiNotar went bust after issuing bogus certificates

- Dutch certificate authority
- More than 300,000 Iranian Gmail users targeted
- More than 500 fake certificates issued
- Major web browsers blacklisted all DigiNotar certs

115

Iranian Gmail users were found to have been given fake certificates for Gmail, allowing a MITM attack to take place. Further investigation revealed that over 500 fake certificates were issued. No public investigation provides conclusive proof of all steps in the process, but the Iranian Government and the NSA have both been suggested as potential attackers. The behaviour of governments here has a significant influence on the security of everyone else. The cryptowars of the 1990s, where governments attempted to mandate exceptional access to encrypted key material, are being revisited. See: Ableson et al. Keys Under Doormats: mandating insecurity by requiring government access to all data and communications. <https://www.schneier.com/academic/paperfiles/paper-keys-under-doormats-CSAIL.pdf>

Further reading: <https://en.wikipedia.org/wiki/DigiNotar>

# TLS security landscape is complex



You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > [www.cst.cam.ac.uk](#)

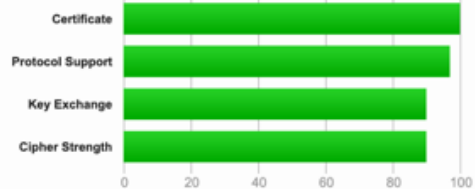
## SSL Report: [www.cst.cam.ac.uk](#) (131.111.150.25)

Assessed on: Fri, 05 Apr 2019 15:49:48 UTC | [Hide](#) | [Clear cache](#)

[Scan Another »](#)

### Summary

Overall Rating



Visit our [documentation page](#) for more information, configuration guides, and books. Known issues are documented [here](#).

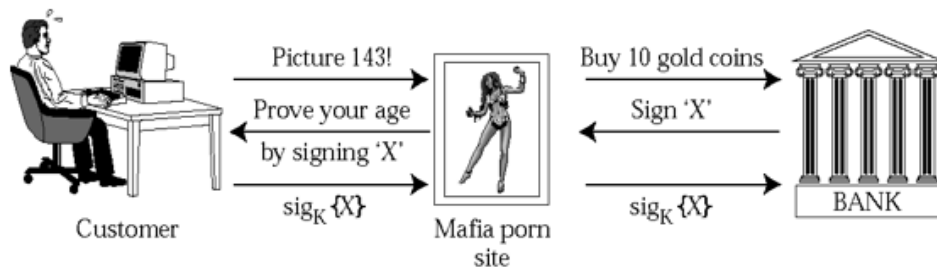
This site works only in browsers with SNI support.

116

Look at the security rating site OpenSSL Labs for the Department's certificate. The landscape here is very complex. You need a detailed tool to check whether your certificate and setup has all the appropriate defences deployed for the various flaws found in the protocol over the years. Note the provision of client compatibility too.

# Chosen protocol attack

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



117

The experience with TLS is challenging for operators because they are heavily reliant on third parties. Just as the saying goes "there is no cloud, just someone else's computer". Such reliance can be abused directly of course, but it also opens up new opportunities for the attacker. Here the mafia has repurposed (deliberately) a signing protocol to extort money from you.

# Bugs are found in and around code

- Bugs in the code
  - Arithmetic
  - Syntactic
  - Logic
  - Concurrency
- Bugs around the code
  - Code injection
  - Usability traps

118

Maurice Wilkes: "It suddenly occurred to me when I was at the corner of the stairs, that I would spend a large part of my life discovering bugs in my own programs."

The first documented use of the term "bug" for a technical malfunction was by Thomas Edison; In the year 1878 he mentioned the term in a private letter. This counters an oft-mentioned view that the term bug is derived from a moth getting trapped in a computer, although perhaps this latter event popularised the term. For further information, see [https://en.wikipedia.org/wiki/Software\\_bug](https://en.wikipedia.org/wiki/Software_bug)



# Patriot missile failures in Gulf War I



German Air Force; CC-BY-SA, Darkone, Wikipedia



Afgan National Army; PD, Davric, Wikipedia

- Failed to intercept an Iraqi Scud missile in first Gulf War on 25<sup>th</sup> February 1991
- Scud struck US barracks in Dhahran; 28 dead
- Other Scuds hit Saudi Arabia, Israel

119

The MIM-104 Patriot is a surface-to-air missile (SAM) system, the primary of its kind used by the United States Army and several allied nations. The picture on the left is a Patriot system used by the German Air Force, August 2005 ([https://en.wikipedia.org/wiki/MIM-104\\_Patriot](https://en.wikipedia.org/wiki/MIM-104_Patriot)). The picture on the right is of a Scud missile and launcher in use by the Afgan National Army.

## Caused by arithmetic bug

- System measured time in 1/10 sec, truncated from  $0.0001100110011\dots_b$
- Accuracy upgraded as system upgraded from air-defence to anti-ballistic-missile defence
- Code not upgraded everywhere (assembly)
- Modules out by 1/3rd sec after 100h operation
- Not found in testing as spec only called for 4h tests

**Lesson: Critical system failures are typically multifactorial**

120

As you will know from the Numerical Analysis course, not all decimal fractions are precisely representable as binary floating-point numbers.

System was upgraded from anti-aircraft to anti-ballistic missile. This required an increase in accuracy since ballistic missiles such as the Scud travel much faster than aircraft. Unfortunately the code was not updated everywhere. This meant that different modules (some with upgraded accuracy, some not) then fell out of sync with each other, resulting in the failure of the Patriot system to effectively target Scud missiles. This problem was not caught by static analysis tools since the code was written in assembly, and therefore there was no high-level language features such as a strong type system which could have helped. Testing was also inadequate – missile defence systems are often operated continuously for hundreds of hours, yet the testing regime only called for testing over a 4-hour period. Short-term solution was to reboot Patriot every 4 hours until the underlying cause was determined.

## Syntactic bugs arise from features of the specific language

For example, in Java:

```
1 + 2 + "" evaluates to "3"
```

```
"" + 1 + 2 evaluates to "12"
```

This is due to coercion from primitive integers to `java.lang.String`

121

Java supports implicit type conversion or *coercion* from primitive integers to Strings. This is typically helpful, however implicit type conversion interacts with implicit operator precedence in the above example, leading to different outcomes for what initially appear to be quite similar expressions. Removing all implicit type conversion may also result in (different) errors since programmers may then insert explicit type conversions which themselves might be problematic.

Further reading: Joshua Bloch and Neal Gafter, Java Puzzlers: Traps, Pitfalls, and Corner Cases, Addison-Wesley. <http://www.javapuzzlers.com/>

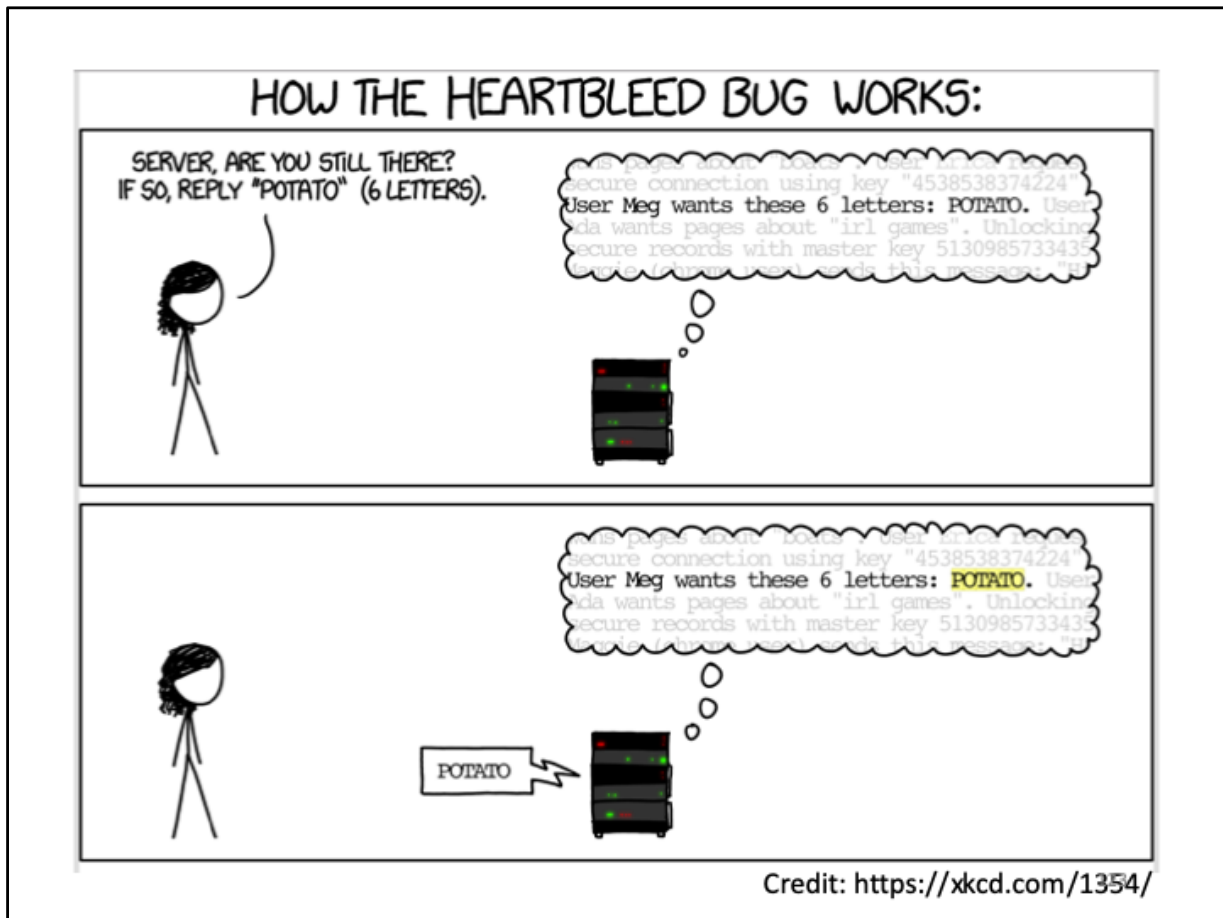
## Apple's goto fail bug (2014)

```
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; //error: this line should not exist
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    //...
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

122

This is a control-flow (logic) bug. Note the two consecutive lines containing “goto fail”; the second is erroneous and the control flow therefore unconditionally executes the code at the “fail” label. It's not clear how this failure was introduced. Perhaps it was an erroneous merge on a commit, either automated or manual by a user. Better unit tests might have helped.

Further reading: <https://www.imperialviolet.org/2014/02/22/applebug.html>



This is another logic bug. The heartbeat feature allowed either the client or the server to ask the other party to reply with a specified message of a given length after a period of time, allowing the requesting party to know that the other was still online and available. Unfortunately the requesting party could claim the provided message was much larger than reality. This led to a buffer over-read vulnerability: the requesting party would receive their message appended with any additional contents found in the server or clients memory. The bug's name derives from heartbeat. NB: In the absence of malice, the code worked just fine.

Further reading: <https://en.wikipedia.org/wiki/Heartbleed>

## Heartbleed allows clients to read the contents of server memory

Therefore a malicious client could read:

- Secret keys of any TLS certificates used by server
- User creds such as email address and passwords
- Confidential business documents
- Personal data

The attack left no trace of use in server logs

124

The potential impact of this vulnerability is huge. Potentially the entire contents of the server's process address space were accessible.

One significant risk was for webservers connected to the public Internet. Since the attack left no trace of use in server logs, this meant that all servers needed to not only upgrade their software to fix the vulnerability, but to replace all important key material. TLS certificates in use by the server are an important example, since the private keys may have been compromised. Ideally all user passwords should have been replaced too as these may have been compromised, but the risk of this type of failure depends on details of any implementation.

## Notification and clean-up difficult

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

125

The original flaw was introduced into the source code repository for OpenSSL on 31<sup>st</sup> December 2011, and was released in OpenSSL in version 1.0.1 on 12<sup>th</sup> March 2012. The bug appears to have been found by multiple people, including members of the security team at Google who produced a fix which appeared on RedHat's issue tracker on 21<sup>st</sup> March 2014. Codenomicon also discovered the problem independently and reported on 3<sup>rd</sup> April 2014. [Dates sourced from <https://en.wikipedia.org/wiki/Heartbleed>]

A significant issue with notification is it was essentially impossible to do so quietly: the number of servers and clients which needed fixing is simply too large. Another problem is that many server operators did not realise that they may have been compromised and therefore did not replace their certificates (potentially allowing a MITM attack on all connections, and in the absence of a version of the protocol with forward secrecy, a passive data capture followed by later processing).

A surprising outcome was that many firms decided to outsource certificates to companies like CloudFlare. This is great for the CEO who no longer gets woken up in the middle of the night with things like Heartbleed; now it's CloudFlare's problem. Unfortunately data may be less secure: encryption now runs from customer to CloudFlare, but not necessarily from CloudFlare to company actual servers unless a premium option is purchased. Of course companies don't pay the premium. So now data is backhauled across the Internet where it can be read with passive taps.

## Intel AMT Bug

- AMT allows sysadmins remote access to a machine, even when turned off (but mains power on)
- Provides full access to machine, independent of OS
- A sketch of the protocol for authentication between machine and remote party is as follows:

C → S: “Hi. I’d like to connect”

S → C: “Please encrypt  $X$  with our secret key”

C → S: “Here are the first  $x$  bytes of  $\{X\}_{KCS}$ ”

126

This is a logic bug in the implementation of the protocol. The failure here occurs because the client (not the server) gets to choose how many bytes ( $x$ ) to return, so a malicious client can choose to return zero bytes. Further reading:

[https://en.wikipedia.org/wiki/Intel\\_Active\\_Management\\_Technology](https://en.wikipedia.org/wiki/Intel_Active_Management_Technology)



## Concurrency bug: time of check to time of use (TOCTOU)

Check

```
...  
File file = new File(args[0]);  
if(!file.canWrite())  
    return;
```

Use

```
RandomAccessFile fp = new  
    RandomAccessFile(file, "rw");  
fp.writeChars("Some replacement text");  
fp.close();  
...
```

Adapted example from [https://en.wikipedia.org/wiki/Time\\_of\\_check\\_to\\_time\\_of\\_use](https://en.wikipedia.org/wiki/Time_of_check_to_time_of_use)

127

In this example, a programmer is writing a program which has the `setuid` bit set (see Operating Systems course from last term). Therefore the programmer first checks whether the user has access to a particular file, then if true, uses the file by writing some data to it.

The bug occurs if the operating system can be coerced into performing a context switch at the red line, during which time a malicious user then (e.g. by updating symbolic links) swaps the file accessed. Then sometime later the program will write to a file which the user has specified which the user may not have write access to. This is called a *race condition*. We will see another race condition bug later in the course (Therac-25). Note that the concurrency here is not within the program itself, which has only a single thread of execution. Rather it occurs because the operating system supports multiple concurrent processes in execution.

## Clallam Bay Jail inmates perform code injection on payphones

1. Inmate typed in the number they wished to call
2. Inmate selected whether the recipient spoke Spanish or English
3. Inmate was asked to say their name; “Eve”, say
4. The phone then dialed the number and read out a recorded message in chosen language and appended inmate name to the end:

“An inmate from Clallam Jail wishes to speak with you. Press three to accept the collect call charges. The inmate’s name is” ... “Eve”

128

The hack is to select the language that the callee *doesn't* speak (e.g. Spanish), and then state your name in Step 3 as “To hear this message in English press three”.

Lesson: remember that you are protecting the whole system, including against potentially malicious users.

# Software and Security Engineering

Lecture 6

**Alastair R. Beresford**

arb33@cam.ac.uk

*With many thanks to Ross Anderson*

Okay Google, what's a Whopper?



130

<https://www.youtube.com/watch?v=InOmTxmq1lk> [Start at 15 seconds in since this video starts with the punchline.] Is it ethical for Burger King to have run this ad? Is it legal (e.g. Computer Misuse Act)?

Back in the 1980s, Ross watched a demo of a DOS system and asked the audience what he should demo. Someone in the audience shouted "Run 'FORMAT C:'" which the demonstrator duly did. And that wiped the OS from the system...

## The Morris Worm: breaking into computers at scale (1988)

- Exploited vulnerabilities in sendmail, fingerd, rsh
- Used a list of common weak passwords
- Gov. assessment: \$100k to \$10M in damage
- 6,000\* machines infected
- Internet partitioned for days to prevent reinfection
- Robert Morris was the first person convicted under the 1986 Computer Fraud and Misuse Act.
  - 3 year suspended sentence
  - 400 hr community service
  - \$10k fine.

131

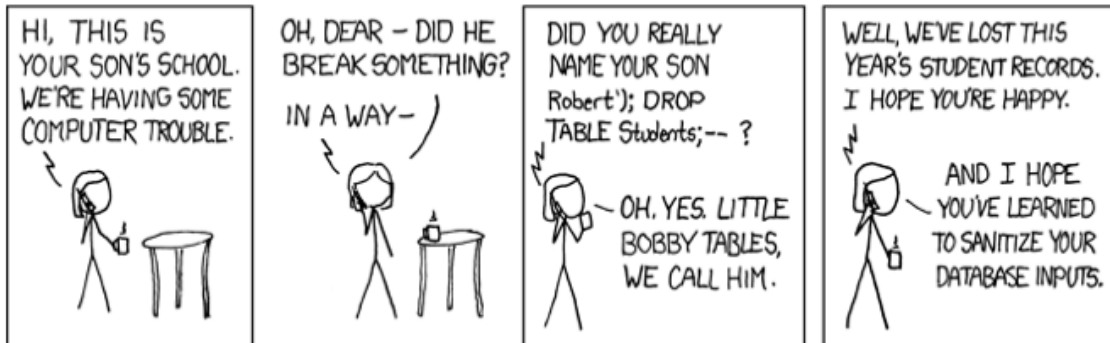
Robert (Tappan) Morris was a graduate student at Cornell University. His father, also called Robert Morris, was a researcher and cryptographer at Bell Labs, later chief scientist at the NSA. Robert Jr is now a tenured professor at MIT.

The attack abused a buffer overflow attack in fingerd; more on buffer overflow attacks next year. The fact that individuals with accounts on two or more computers would often tie them together so you could login from one machine to another without entering a password was also used. Finally the worm tried logging in with a list of common passwords. The worm also checked whether it was already present on a machine, but still attempted to copy itself 14% of the time. The aim was to increase robustness, but the reality was that it resulted in high system loads, bringing the attack to the attention system administrators. Morris was convicted under the Computer Fraud and Misuse Act and “he was sentenced to three years of probation, 400 hours of community service, and a fine of \$10,050 plus the costs of his supervision”.

[https://en.wikipedia.org/wiki/Morris\\_worm](https://en.wikipedia.org/wiki/Morris_worm)

[https://en.wikipedia.org/wiki/Robert\\_Tappan\\_Morris](https://en.wikipedia.org/wiki/Robert_Tappan_Morris) (more info on this page on how the work actually worked as well as the sentence)

# SQL Injection attack: failure to sanitize untrusted inputs



```
String sql =  
    "INSERT INTO Students (Name) VALUES ('"  
    + studentName  
    + "')";
```

132

There continue to be an amazing number of SQL injection attacks. That's because many programming languages contain libraries which makes this the default or at least easy to get wrong – an example in Java is given above. How can we solve this problem? In Java, always use the PreparedStatement class which sanitizes inputs; other languages have similar support structures. Use them! Further reading: <https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html>

# Software countermeasures: systems and tools

- Operating system protections
  - Data execution prevention
  - Address space layout randomisation
  - ...
- Tools, e.g. Coverity
  - Static analysis
  - Dynamic analysis
  - Testing frameworks
  - ...
- Automated update systems to install patches

133

A number of security features were already discussed in the Operating Systems course, such as the access control features to protect memory regions in systems which support either paging and/or segments. These can be used to mark portions of memory as either read and execute or read-write. There are many, many tools. We will look at Coverity later in this course.

## Software countermeasures: reducing bug number and severity

- Defensive programming
- Secure coding standards
  - See Howard and LeBlanc on MS standards for C
- Contracts, e.g. in the Eiffel language
- API analysis
  - Combining API calls may lead to vulnerabilities
  - Challenging for APIs accessible over the Internet



# We cannot write code without latent vulnerabilities

## Milk or Wine: Does Software Security Improve with Age? \*†

Andy Ozment  
MIT Lincoln Laboratory‡

Stuart E. Schechter  
MIT Lincoln Laboratory

### Abstract

We examine the code base of the OpenBSD operating system to determine whether its security is increasing over time. We measure the rate at which new code has been introduced and the rate at which vulnerabilities have been reported over the last 7.5 years and fifteen versions.

We learn that 61% of the lines of code in today's OpenBSD are *foundational*: they were introduced prior to the release of the initial version we studied and have not been altered since. We also learn that 62% of reported vulnerabilities were present when the study began and can also be considered to be foundational.

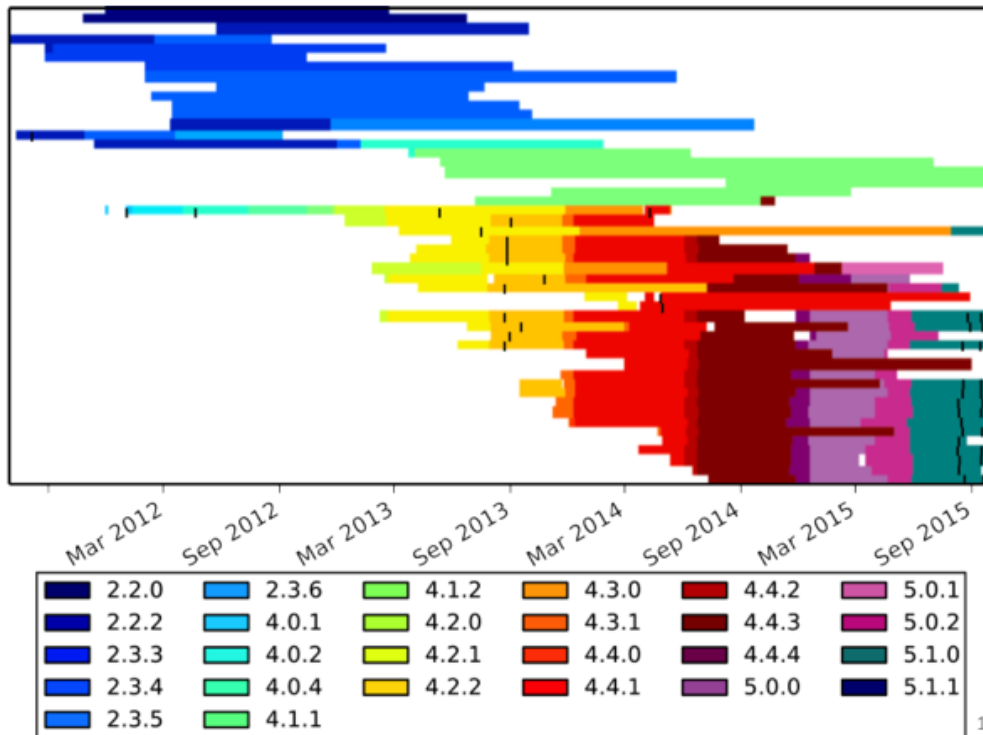
### 1 Introduction

Many in the security research community have criticized both the insecurity of software products and developers' perceived inattention to security. However, we have lacked quantitative evidence that such attention can improve a product's security over time. Seeking such evidence, we asked whether efforts by the OpenBSD development team to secure their product have decreased the rate at which vulnerabilities are reported.

In particular, we are interested in responding to the work of Eric Rescorla [11]. He used data from ICAT<sup>1</sup> to argue that the rate at which vulnerabilities are reported has not decreased with time; however, limitations in the

Researchers analysed the code for the OpenBSD operating system and measured the rate at which new vulnerabilities were reported over 7.5 years. Over that period 61% of the code didn't change at all and 62% of the vulnerabilities reported were found in that initial version. There was good evidence that the rate at which vulnerabilities were found in that foundational code reduced over time – good news! Unfortunately it's not fast – vulnerabilities had a median lifetime of 2.6 years. It's also worth remembering that the code base changes over time (39% was introduced since the start of the measurement period). Therefore we should not expect to ever end up with bug-free software. All code essentially has latent vulnerabilities in it.

## OS versions of 50 LG handsets



136

We ran Device Analyzer, an Android data collection project in the Computer Lab, from 2010 until 2019. DA collected Android usage statistics from study participants – around 30,000 people in total. Amongst other things we collected was OS version numbers and build numbers for the handsets. Here you can see the OS version numbers for 50 LG handsets in the database. Note that some of the handsets run versions which never change, and some see frequent change. The black vertical ticks in the graph delineate changes to the build number which do not result in changes to the OS version (likely security fixes).

Are most Android handsets like the top half of this figure, or the bottom half? If they are like the top half, does that mean that they are vulnerable to many critical flaws which are discovered over time or is Android somehow lacking in vulnerabilities?

# Link OS versions to database of vulnerabilities

Match OS version information to OS and Build Number to put each handset into one group:

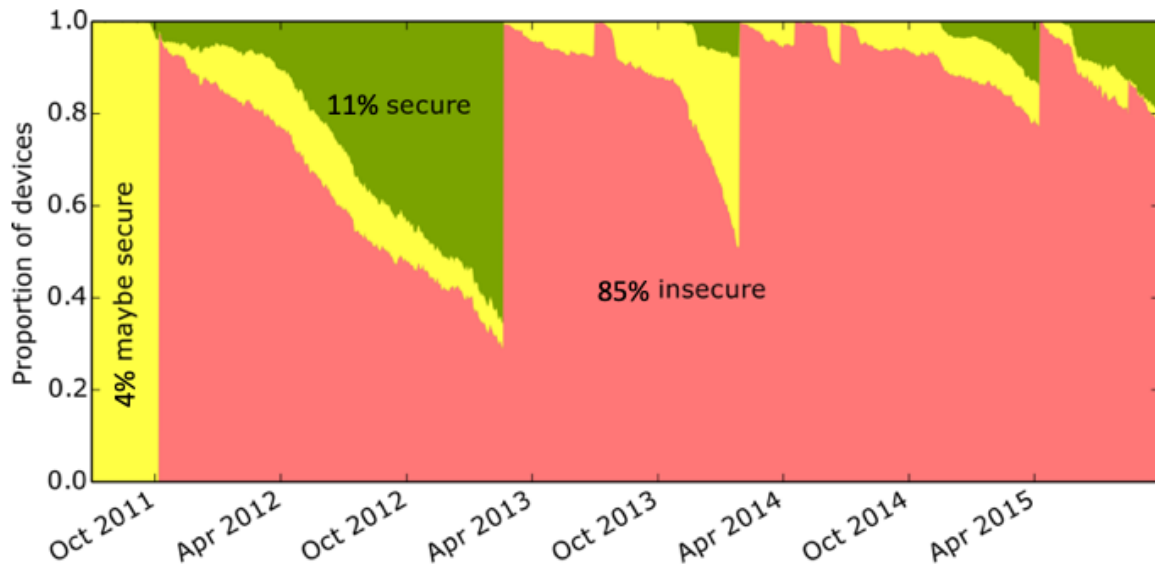
- Insecure
- Maybe secure
- Secure

137

To determine whether Android handsets are running vulnerable versions of the OS we need some data on vulnerabilities. Therefore we built a database of vulnerabilities and matched this to OS version number. One problem is that it is possible that manufacturers *backport* critical fixes to old versions of Android. We wanted to exclude this possibility, so we put each device in our dataset on each day into one of these three categories.

- Secure: if the device is running a secure version of Android on a specific date.
- Maybe secure: if the device is running a vulnerable version of Android on a specific date, and we did *not* see the build number in the wild before the date of disclosure for the vulnerability (so the OS may contain a backported fix).
- Insecure: if the device is running a vulnerable version of Android on a specific date, and we saw the build number in the wild before the date of disclosure for the vulnerability (so the OS cannot contain a backported fix).

## On average, 85% are vulnerable



138

This is what we get when we plot the proportion of handsets in each of these three categories over time. This graph uses 11 vulnerabilities which we could identify and clearly trace. There were many more vulnerabilities over this period, so this is an underestimate of the security of Android.

Further reading: Thomas, Daniel R., Alastair R. Beresford, and Andrew Rice. "Security metrics for the android ecosystem." *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, 2015. <https://dl.acm.org/citation.cfm?id=2808118>

## The *Software Crisis*

- Software still lags behind hardware's potential
- Many large projects are late, over budget, dysfunctional, or abandoned (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)

139

CAPSA: Pensions project in the University which was years late and over budget.

NPfIT: National Programme for IT (NHS).

DWP: Universal Credit

Addenbrookes: was put into special measures because they put in a new computer systems which was unable to produce all the right stats for the government to convince government that all was well at the hospital.

# London Ambulance Service disaster

- Widely cited example of project
- Many aspects of the failure widely repeated since
  
- Attempt to automate ambulance dispatch in 1992
- Result left London without service for a day
- Number estimated deaths ran as high as 20
- CEO sacked; public outrage

140

LAS is explored in detail in this course because it combines together many of the errors and difficulties in large projects and it is also well documented. You are strongly encouraged to read the original report rather than simply rely on the material presented here. You can download the main report and also additional material from this page:

<http://www0.cs.ucl.ac.uk/staff/A.Finkelstein/las.html>

## Project background

- Attempt to automate in 1980s failed – system failed load test
- Industrial relations poor; pressure to cut costs
- Public concern over service quality
- South West Thames Regional Health Authority 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

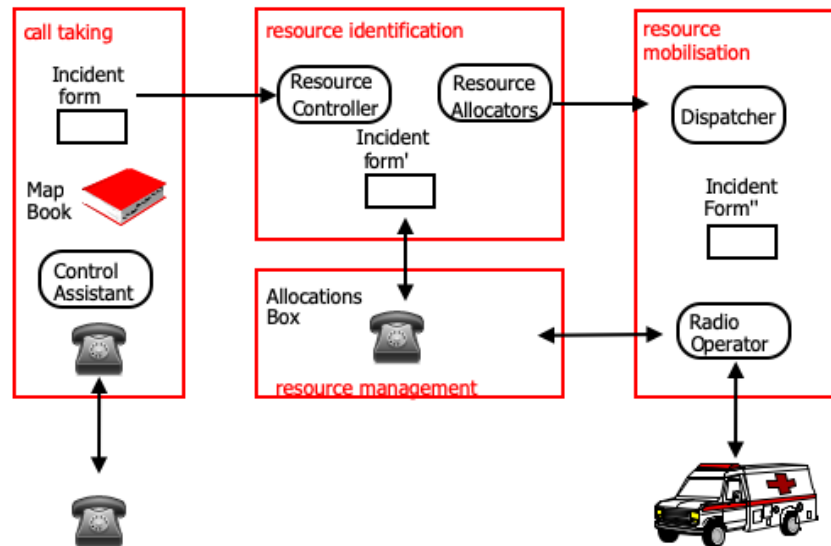
141

Recall this was 1992, before almost all of you were born. There were essentially no mobile phones, and GPS not practical. If you called for an ambulance, you did so with a landline. If you wanted to know where an ambulance was, you had to speak to the driver and hope they could explain where they were succinctly and accurately.

There is also the political background. This was just after the Thatcher era, and relations between Government, industry and the trade unions was poor. There was also pressure to cut costs. Here "email" wasn't the modern email to a smartphone, but a custom solution was needed to route a short message to the ambulance.

AVLS: Ambulance Vehicle Location Service

# Original dispatch system worked on paper with regional control



142

1. 999 calls written on paper tickets; map reference looked up; conveyor belt brought paper to central point
2. Controller deduplicates tickets and passes to three regional divisions: NW / NE / S
3. Division controller identifies vehicle and puts note in its activation box
4. Ticket passed to radio controller who contacted the ambulance concerned



## Many problems with original system

- It took 3 minutes to dispatch an ambulance
- It required 200 staff (out of 2700 in total).
- There were errors, especially in deduplication
- Queues and bottlenecks, especially with the radio
- Call-backs tiresome

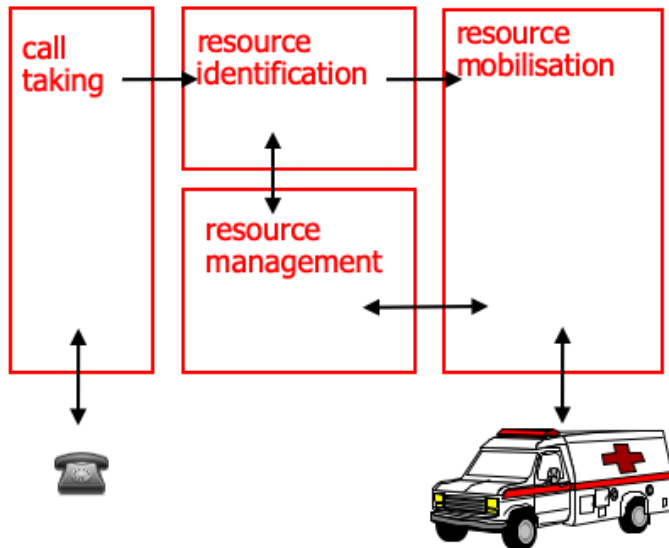
143

Three minutes is a long time for some illnesses such as heart attacks, so there was significant interest in a system which was able to work more quickly.

Radio queues were a problem since if an ambulance driver was on the radio to one person he or she couldn't talk to another.

There were therefore good reasons to look for a better solution. Automation particularly appealing as it had the potential to dramatically reduce waiting times. Automation so appealing that LAS had tried to use computers before, and the previous attempt had failed.

# Computer-aided dispatch system



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

144

This is a seriously challenging project.

## Tender process was poor

- Idea of a £1.5m system stuck; idea of AVLS added; proviso of a packaged solution forgotten; new IS director hired
- Tendered on 7<sup>th</sup> Feb 1991; completion due Jan 1992
- 35 firms looked at tender; 19 proposed; most said timescale unrealistic, only partial automation possible by early 1992
- Tender awarded to consortium of Systems Options Ltd, Apricot and Datatrak for £937,463
  - £700K cheaper than next lowest bidder!

145

People tend to round down rather than round up, so £1.9m became £1.5m in the eyes of the buyers. The idea of Ambulance Vehicle Location Service (AVLS) stuck, even though this was not included in the costing undertaken by the consultants (instead this was an extra, and at the time a challenging technical problem in its own right).

The new IS director meant there was no institutional knowledge. Many experienced tenders thought the set of requirements were possible (IBM said two years and £2m). In these days, tender was typically awarded to the cheapest, which in this case was a small firm with four people with arrangements to subcontract to third-party suppliers for some of the work.

Systems Options only include £35k for software development.

## Phase one: design work 'done' in July and contract signed in August

Minutes of a progress meeting in June recorded:

- A 6-month timescale for an 18-month project
- A lack of methodology
- No full-time LAS users providing domain knowledge
- Lead contractor (System Options) relied heavily on cozy assurances of subcontractors

Unsurprisingly LAS told in December that only partial automation by January deadline – front end for call taking, gazetteer, docket printing

146

Systems Options had no leverage over the big suppliers that they relied on, yet these were essential.

Radio kit didn't work well and there were radio blackspots. Poor industrial relations and diversity of working practices meant that ambulance drivers might take a tea break, yet management didn't want them to do this. Drivers were concerned about being "watched" and so would take their tea breaks in the radio blackspots.

## Phase two: full automation

- Server never stable in 1992; client and server lockup
- Radio messaging with blackspots and congestion; couldn't cope with established working practices
- Management decided to go live on 26<sup>th</sup> Oct 1992
- Independent review had called for volume testing, implementation strategy, change control, ...all ignored
- CEO: "No evidence to suggest that the full system software, when commissioned, will not prove reliable"
- On 26 Oct 1992, room was reconfigured to use terminals, not paper. There was no backup...

147

In a counter-narrative to the CEO, there was plenty of evidence that the full system would be unreliable.

[Ask the audience to list all the evidence they have heard already.]

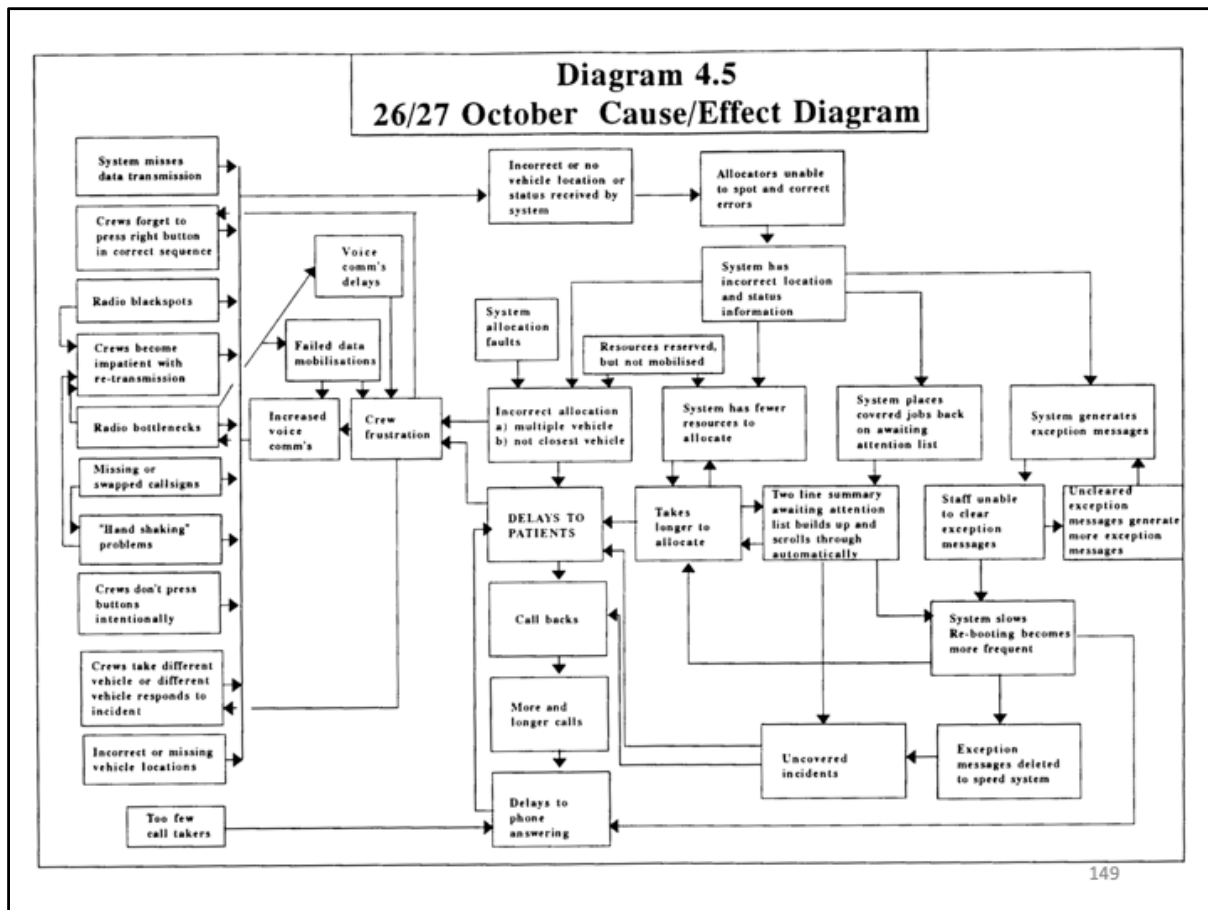
## Circle of disaster on 26/7<sup>th</sup> October

- System progressively lost track of vehicles
- Exception messages scrolled 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
- System slowdown and congestion leading to collapse

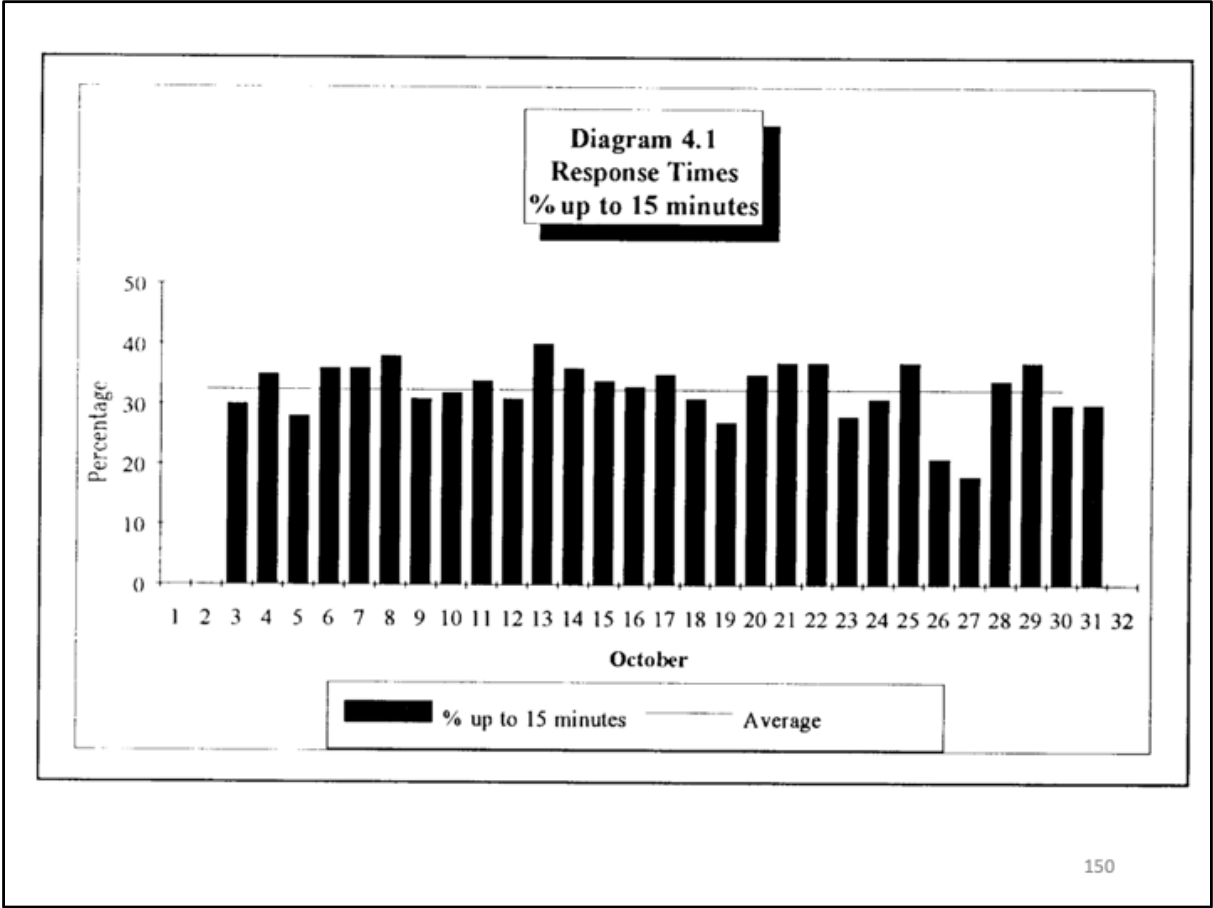
148

This is an example of cascade failure.

Switch back to semi-manual operation on 26th and to full manual operation on 2<sup>nd</sup> November after system crash.

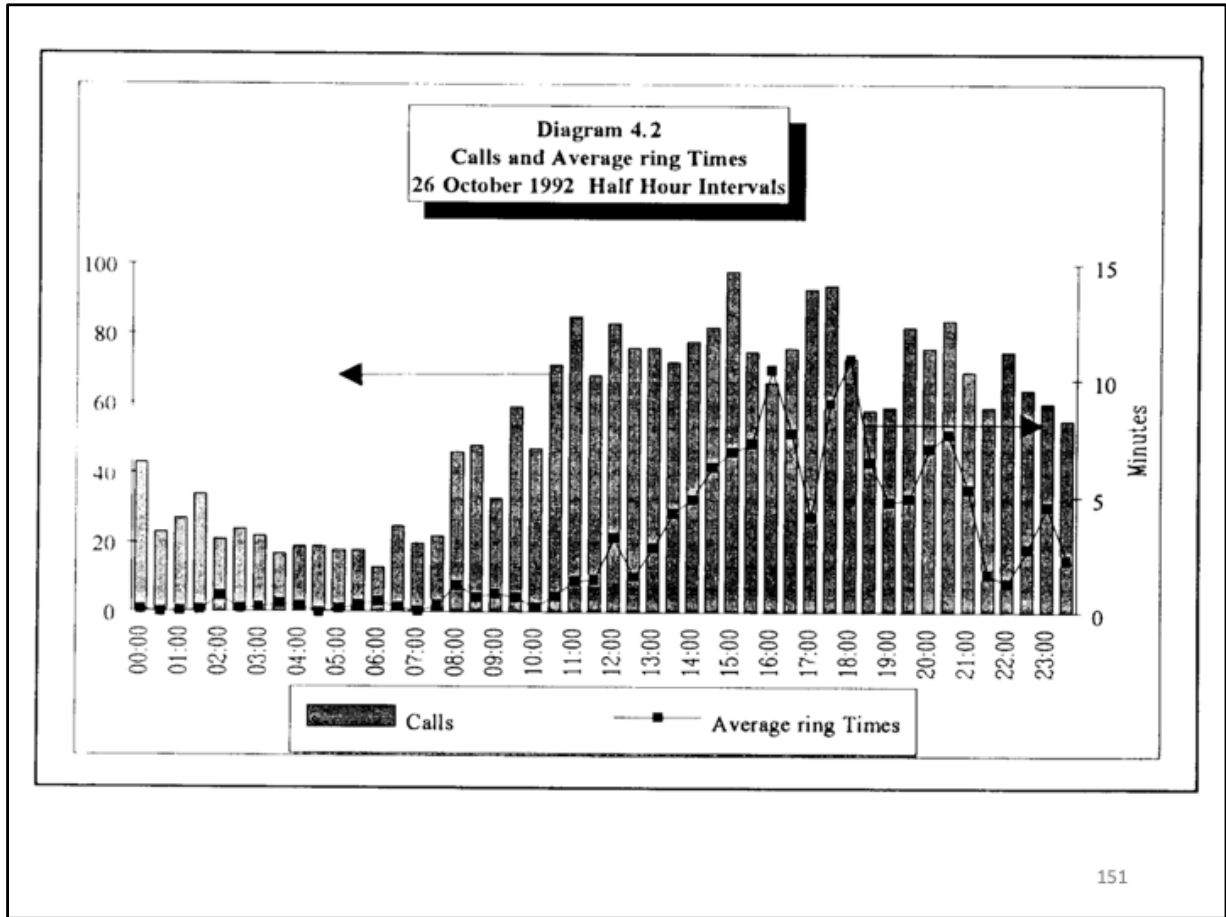


The report contains a detailed description of the cascade of failures.

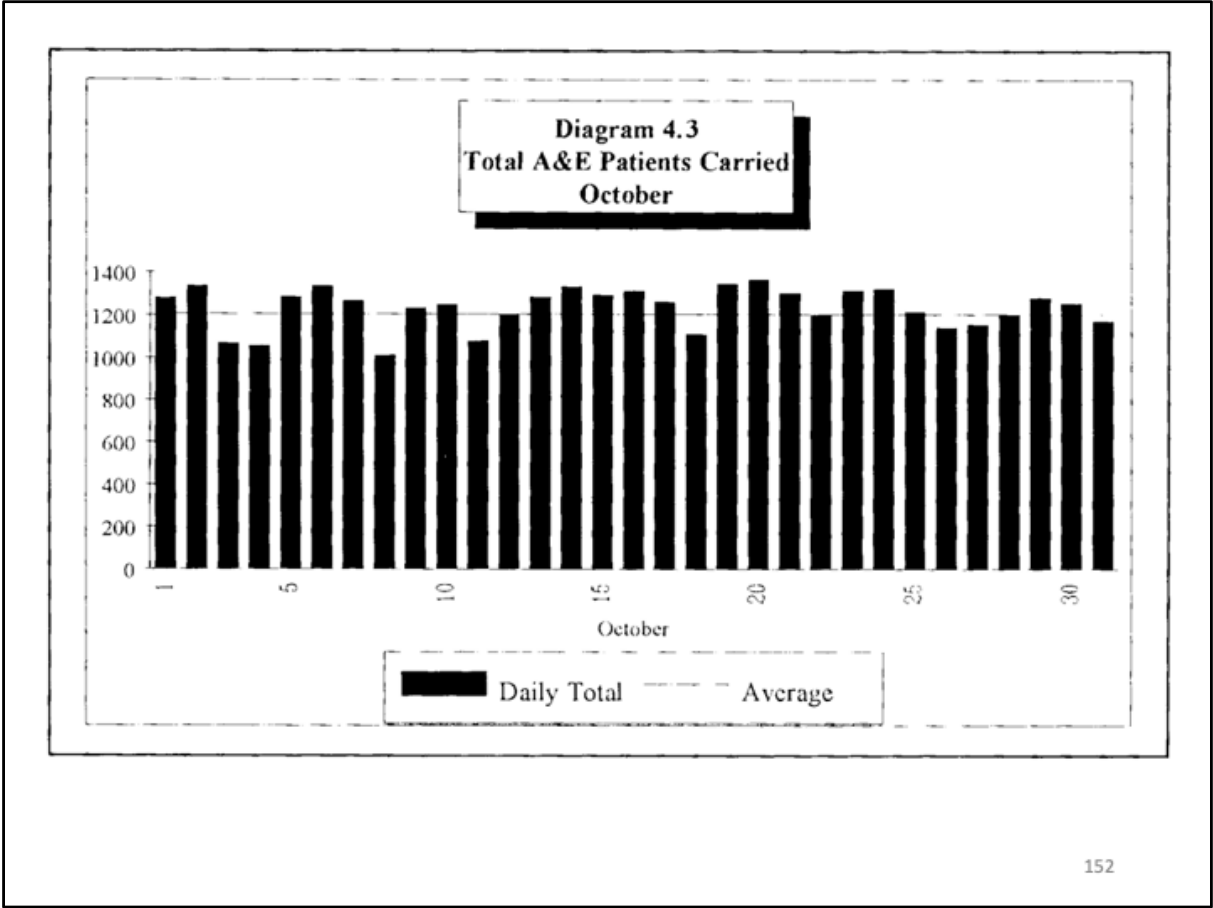


Response times were worse with the new system.



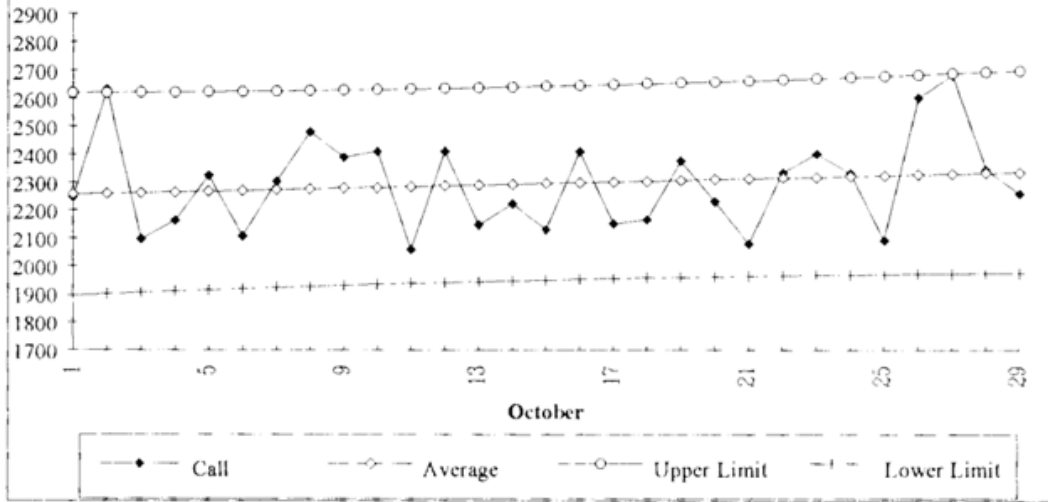


Note the congestion build from around 10-11am. Call volumes keep going up and up as people keep calling back, so the ring time goes from under one minute to ten minutes. Imagine that. Calling 999 for ten minutes and nobody answers! Presumably individuals end up getting a taxi to hospital or simply dying on the spot.



There was no large, atypical surge in admissions to A&E, so the failure of the system was not down to an anomalously challenging day.

**Diagram 4.4**  
**Calls recorded by call logger**



## Collapse likely resulted in deaths

- One ambulance arrived to find the patient dead and taken away by undertakers
- Another answered a 'stroke' call after 11 hours and 5 hours after the patient had made their own way to hospital
- ...
- Chief executive resigns

# Software and Security Engineering

Lecture 7

**Alastair R. Beresford**

arb33@cam.ac.uk

*With many thanks to Ross Anderson*

155

## Warm up: What mistakes were made in the LAS system?

- Specification
- Project management
- Operational

## Specification mistakes

- 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

157

If you hire consultants which say that it's going to take two years and £2m, believe them! Insufficient due-diligence performed on the capability of a small company.

In the case of the LAS, people need to think about how the current system works now, but instead they thought about how to please the health secretary and meeting KPIs (e.g. an ambulance arrives within 15 minutes). Attempting to change working practices to move to central command from a scenario where there was lots more flexibility (e.g. tea breaks, which weren't official, but with one driver covering for another or swapping tasks worked out well for patients, but was not "approved" or perhaps even known about). Such information needed to be collected and prepared at an early stage.

# Project management mistakes

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

158

The LAS project was proceeding in the wrong way and had the wrong goals. An example you will have next year is the Part IB project: here you will learn that organisation matters; you will also need to work from a poor specification and work out how to turn this into something shiny for demo day.

Remember that the tech in these days was inferior today: they were using Windows 3.0 -- a museum specimen by today's standards. But there were tools which could have been used to help, including revision control systems. RCS was available in those days, which wasn't as good as Git, but is much better than nothing. Usability engineering is important: the unfamiliar and nasty interface for the ambulance crew - - just like we saw in the infusion pumps – caused problems. If crews pressed the wrong button and the ambulance disappears from the central system then things are going to fail.



# Operational mistakes

- 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 effective back-up system in place

# NHS National Programme for IT

Idea: computerise and centralise all record keeping for every visit to every NHS establishment

- 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 or never worked
- Coalition government: NPfIT 'abolished'

160

Cue the 2000s and something even bigger than LAS. Like LAS, this was an attempt to centralise power and change working practices. This is problematic. After Blair won his second election, there was a meeting in Downing Street and the idea was pitched to him. The aim was to invest in IT to improve medical care. Blair apparently asked whether this could be done by 2005 (i.e. before the next election) even though it was really a five-year project. The person from NHS said "yes", in order to close the deal, even though this was unlikely to happen.

Part of the problem was that the specification was determined by asking doctors to write down a long list of "nice to have" things. One example was care pathways: the idea is to computerise and centralise all record keeping for every visit to every NHS establishment. So if you feel unwell, you visit your GP, then you are referred to a consultant at Addenbrookes, who sends you to another hospital for specialist care, and then back to your GP for drug prescriptions; all this would be recorded centrally.

For further information, see the case history written by Masters of Public Policy students discussed and linked to from here:

<https://www.lightbluetouchpaper.org/2014/08/13/largest-ever-civil-government-it-disaster/>

## Universal Credit: fix poverty trap

Idea: Hundreds of welfare benefits which means there is often little incentive to get a job.

- Initial plan was to go live in October 2013
- A significant problem: big systems take seven years not three; doesn't align with political cycle
- Complexity was huge, e.g. depended on real-time feed of tax data from HMRC, which in turn depended on firms

161

There is a poverty trap where people get stuck in welfare, particularly those which have dependants; often going out to work results in less income overall, so there is little incentive to change. The idea was to replace all the hundreds of separate benefits with one, integrated system, which had better incentives. The basic problem is that large IT projects take seven years, not three, to roll out, and this then means that it doesn't align well with political cycles which are too short to support the launch of a complex project.

# NAO: poor value for money, not paying 1 in 5 on time



<https://www.youtube.com/watch?v=qE2fpNSrrpc>

162

2018 National Audit Office report, summary and introductory video:  
<https://www.nao.org.uk/report/rolling-out-universal-credit/>

## Smart meters: more centralisation

Idea: expose consumers to market prices, get peak demand shaving, make use salient

- 2009: EU Electricity Directive for 80% by 2020
- 2009: Labour £10bn centralised project to save the planet and help fix supply crunch in 2017
- 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 ...

163

“A smart meter is an electronic device that records consumption of electric energy and communicates the information to the electricity supplier for monitoring and billing. Smart meters typically record energy hourly or more frequently, and report at least daily. Smart meters enable two-way communication between the meter and the central system. Such an advanced metering infrastructure (AMI) differs from automatic meter reading (AMR) in that it enables two-way communication between the meter and the supplier.” [https://en.wikipedia.org/wiki/Smart\\_meter](https://en.wikipedia.org/wiki/Smart_meter)

There was a forecast in 2009 that we would have a shortage of electricity; the credit crunch has fixed that. There are basic problems, like the fact that there aren't enough registered gas fitters to fit this many gas meters. This is a rare example of a policy that all politicians supported. It was Ed Milliband's idea, but the Coalition supported it and it was in the Coalition agreement; The Greens backed it because it sounded green. There are now different meters being installed against different specs, and some are incompatible so that if you change suppliers, your smart meter becomes dumb. The politicians are now saying that "everyone can have one by 2020" not that 80% of homes will have them, which is actually a rather different target to the one originally set.

## Software engineering is about managing complexity at many levels

- Bugs arise at micro level in challenging components
- As programs get bigger, interactions between components grow at  $O(n^2)$  or even  $O(2^n)$
- The 'system' isn't just the code: complex socio-technical interactions mean we can't predict reactions to new functionality

**Most failures of really large systems are due to wrong, changing, or contested requirements**

164

What this all teaches us is that software engineering is more than just managing the complexity of your code. There are problems at all levels.

Social-technical systems are a real problem. Consider the smart meter market: there are millions of households, dozens of key players (including the electricity companies, national grid, politicians, etc).

Most failures are due to wrong, or changing, or contested requirements. (Or indeed no requirements at all!) Think about the LAS: the requirements didn't fit working practice. The smart meter project also is a good example.

## Project failure, circa 1500 BCE



165

“The Tower of Babel as told in Genesis 11:1–9 is an origin myth meant to explain why the world's peoples speak different languages. According to the story, a united humanity in the generations following the Great Flood, speaking a single language and migrating eastward, comes to the land of Shinar. There they agree to build a city and a tower tall enough to reach heaven. God, observing their city and tower, confounds their speech so that they can no longer understand each other, and scatters them around the world.”

[https://en.wikipedia.org/wiki/Tower\\_of\\_Babel](https://en.wikipedia.org/wiki/Tower_of_Babel)

“Come, let us build ourselves a city and a tower with its top in the heavens, and let us make a name for ourselves, lest we be dispersed over the face of the whole earth.” And the Lord came down to see the city and the tower, which the children of man had built. And the Lord said, “Behold, they are one people, and they have all one language, and this is only the beginning of what they will do. And nothing that they propose to do will now be impossible for them. Come, let us go down and there confuse their language, so that they may not understand one another’s speech.” So the Lord dispersed them from there over the face of all the earth, and they left off building the city. Therefore its name was called Babel, because there the Lord confused the language of all the earth. (From the Bible.)

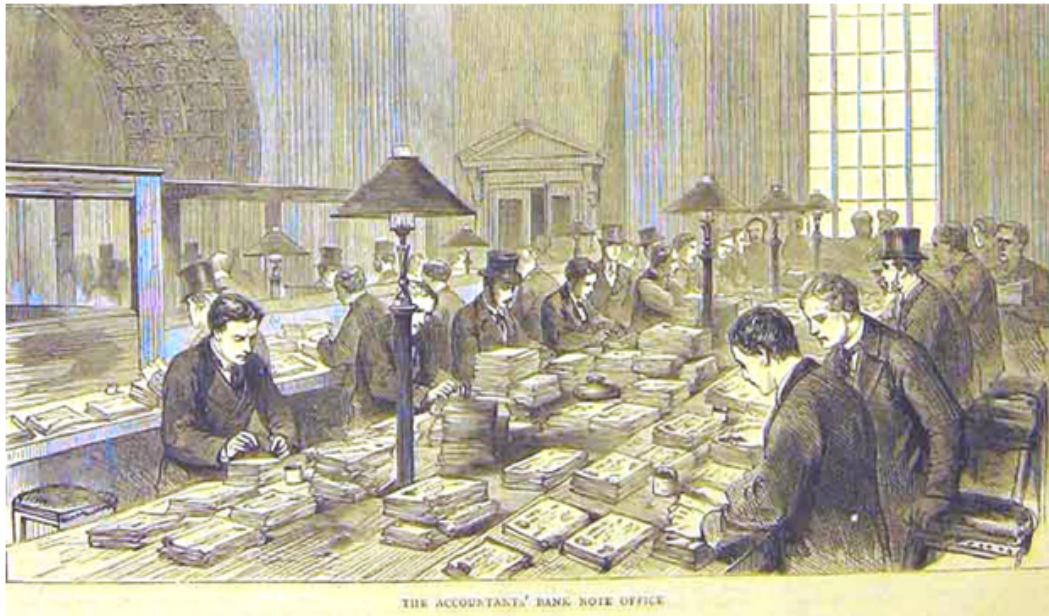
## On contriving machinery

*“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”*

*Charles Babbage*



## Bank of England, 1870



167

Let's start the challenge on the design of computer systems by looking at some early examples. This is a picture of banking in the Victorian era. There are a large number of computers crunching the numbers. Can you see them? The humans are the computers. If you look at the surviving materials, the computers are doing sorting, searching, etc; and they are doing security as well (e.g. double-entry bookkeeping) so that there is separation of concerns. No single man in a top hat can defraud the bank.

## Dun, Barlow & Co, 1876



Across the pond it looked the same. Here you can see big wooden cabinets and you could look up whether someone has paid their mortgage or not.

## Sears, Roebuck and Company, 1906



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

169

Sears, Roebuck and Company (colloquially “Sears”) are an American chain of department stores. In the early 20<sup>th</sup> Century, their innovation was mail order, built on the fixed cost of postal delivery by the US Post Office. Their marketing and business strategy was “like it, or your money back”. Others thought they were crazy at the time with such an offer, but it worked. They still use this approach, including the catchy phrase, today.

## First National Bank of Chicago, 1940



170

IBM were running by then. IBM was a start up where Tom Watson took some customers from an earlier company. Note here that you have a preordained market for a mainframe -- there is already an algorithm running here which could be moved from humans to electronic computers.

## The software crisis, 1960s

- Large, powerful mainframes made complex systems possible
- People started asking why project overruns and failures were so much more common than in mechanical engineering, shipbuilding, etc.
- The term *software engineering* coined in 1968
- The hope was that we could things under control by using disciplines such as project planning, documentation and testing

171

Suddenly people found out that software projects failed much more frequently than in other engineering disciplines. In 1968 NATO organised a conference in Newcastle, and Brian Randall came up with the term *Software Engineering*.

His basic point was that we should adopt a more traditional project management approach. For example, when building ships, we know to first lay down the rigs, then attach the skip, then drop in the engines, and so on. We'll see some of the tools you can apply later, but we address the basics first.

## Those things which make writing software fun also make it complex

- Joy of solving puzzles and building things from interlocking 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

172

The non-repeating task aspect really makes it a challenge. For a ship, there's only certain parameters which can (sensibly) be varied; what about software? How do we constrain the task so that we can repeatably build software on time and on budget?

## How is software different?

- Large computer 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

173

Big computer systems become qualitatively different from small ones; in contrast, big ships have more steel, but they have similar design parameters from smaller ships. Physical world objects are also more visceral: it's obvious that changing your mind half way through building a big ship and saying you want an aeroplane instead is stupid; this isn't always obvious for an IT project for a non-technical (or even technical) person. Note that for these big projects, no one person understands how everything works. We need abstraction, but these abstractions sometimes break down.

## Software economics can be nasty

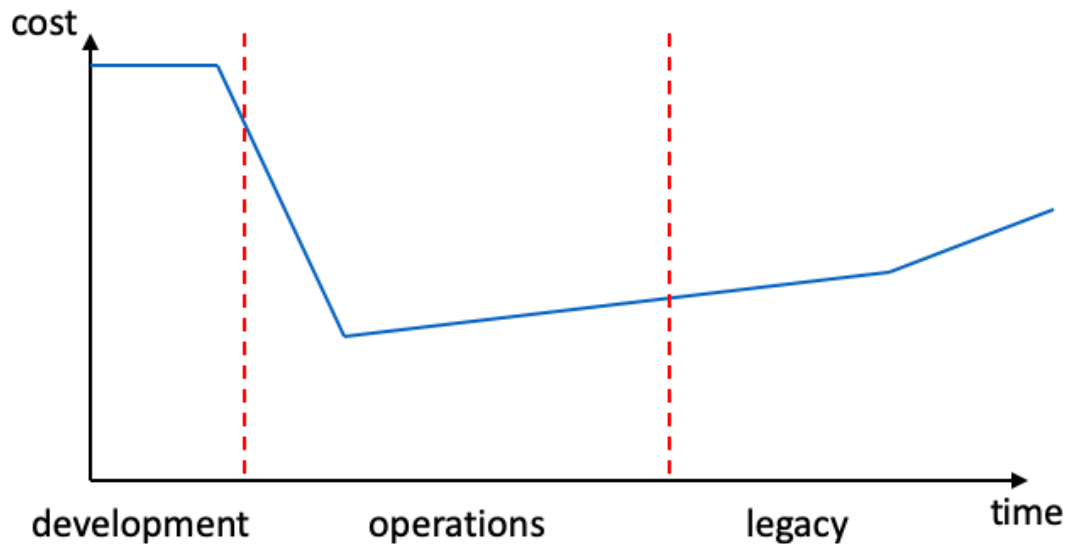
- Consumers buy on sticker price
- Businesses buy based on total cost of ownership
- Vendors use lock-in tactics
- Complex outsourcing

174

This is why printers for customers buy printers for £30, and make their money by charging for ink; businesses buy laser printers because they think of total-cost of ownership.



## Cost of software: development 10%, maintenance 90%



175

In the early days (1950s - 1970s) having bought your computer you first needed to hire some developers to write some code for your saw mill. It works, but then your requirements change, perhaps because you want to make 6 inch planks, not just 4 inch planks. So you need to get more developers in to adjust and tweak the software. So the developer cost of the initial version ends up being small compared to maintenance costs.

Note that the above is an infographic, rather than an accurate representation of an actual project. Even quantifying cost is hard in this space.

# Measuring cost of code is hard

## First IBM measures (1960s)

- 1.5 KLOC per developer-year (operating system)
- 5 KLOC per developer-year (compiler)
- 10 KLOC per developer-year (app)

## AT&T measures

- 0.6 KLOC per developer-year (compiler)
- 2.2 KLOC per developer-year (switch)

176

In the 1960s, IBM found it was getting less than 1.5 KLocs per developer year for OS (in assembler) but much better in apps (in Fortran) which also did much more since one line of Fortran typically did more than one line of assembler. Yay for high-level programming languages!

# KLOC is a poor measure

- ```
//Print out hello
```
1. 

```
for (int i = 0; i < 4; i++) {  
    System.out.println("Hello, world");  
}
```
  2. 

```
for (int i = 0; i < 4; i++) { System.out.println("Hello, world");}  
  
System.out.println("Hello, world");
```
  3. 

```
System.out.println("Hello, world");  
System.out.println("Hello, world");  
System.out.println("Hello, world");  
System.out.println("Hello, world");
```

## Alternatives:

- Halstead (entropy of operators/operands)
- McCabe (graph entropy of control structures)
- Function point analysis

177

Let's say that we want to print out "Hello, world" four times. How should we do it? Here are three options in Java.

The definition of KLOC appears to be precise, however there are significant issues. What is the right KLOC count in the above? Should we include comments? We could try and refine into, say, logical lines of code, which depends on the coding style of the project and the language. This remains problematic however since some code is "easy" and other parts "hard". How could we codify this? Alternative measures for KLOC: compress the source and see how many bits it takes -- an approximation for complexity and effort by the developer. McCabe: look at the control structure of the program and measure the complexity of the graph as output.

For further information, see: [https://en.wikipedia.org/wiki/Function\\_point](https://en.wikipedia.org/wiki/Function_point)

## Early lessons: productivity varies, use a high-level language

- Huge variations in productivity between individuals
- The main systematic gains come from using an appropriate high-level language since they reduce accidental complexity; programmer focuses on intrinsic complexity
- Get the specification right: it more than pays for itself by reducing the time spent on coding and testing

178

If you look at digging holes, then the productivity of the best digger might be 2 or 3 times as much soil compared with the worst digger. Software is not like that.

The star programmer might be an order of magnitude or more times more productive. In addition, they are often able to solve problems which others simply cannot figure out. You can look for proxies such as the success in past projects with significant intrinsic complexity or qualifications. The latter is the basis for the Graduate Ring in the department: we help older alumni find younger alumni to work in their start-up or business.

Why do high-level languages help? There are two types of complexity: there is accidental complexity such that the x value is stored in register 14; this can be fixed in high-level languages where you can label a register as value "x". However there is also intrinsic complexity: for example if you are designing a navigation system there is some hard maths which needs to be solved, and this is true in assembly, Fortran, Java, or whatever. Better tools can help with incidental complexity but not intrinsic complexity.

Every year when the Part IB project ends, the students always say they wish they had done more planning. So let this be a warning to you. Planning may seem boring, and you want to get on to the fun part (coding), but hold in there and make sure you've worked through as many details as you can. Think carefully about the time you should spend on planning and producing a specification, and then double it!

## Barry Boehm surveyed relative costs of software development (1975)

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

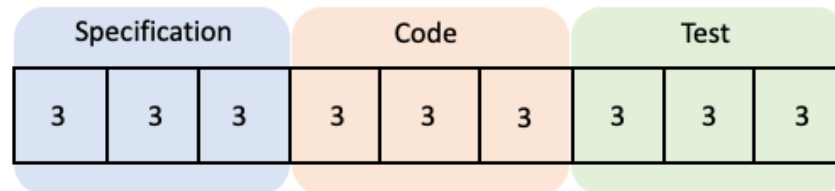
- All stages of software development require good tools

179

Barry Boehm is a famous software engineer ([https://en.wikipedia.org/wiki/Barry\\_Boehm](https://en.wikipedia.org/wiki/Barry_Boehm)). Here's some numbers in terms of costs for different activities in the 1970s. Observe that in every single case the amount of effort put into testing is equally to or greater than coding; same for the specification. Indeed, with the exception of Space, its the specification which got the lion's share of the effort and therefore of the dollars. Therefore, if you're going to build tools to support software development, you shouldn't constrain yourself to building tools to support programming; you should also build tools to support the tasks of specification and testing.

A similar basic point (there are many more errors in the design stage than coding stage) is made in the following: Boehm, Barry W., Robert K. McClean, and D. E. Urfrig. "Some experience with automated aids to the design of large-scale reliable software." *IEEE Transactions on Software Engineering*1 (1975): 125-133. <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6312826>

## Mythical Man-Month: “adding manpower to a late project makes it later”



Example project with 3 developers and 9 months. Initial estimate is 9 person-months each for spec, code and test.

- But spec ends up taking 12 PMs. What do you do?

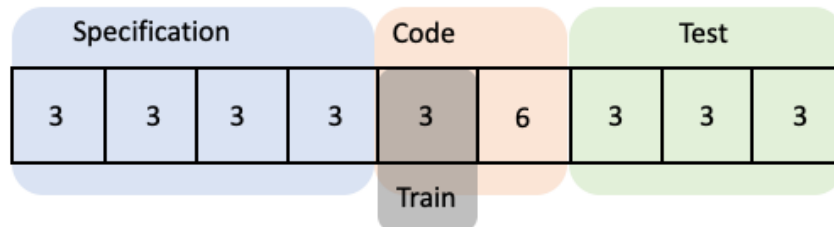
180

While cost metrics such as KLOCs and person months are appealing, they are at best an approximation, and often a poor abstraction. Beware of this when you are involved in project management. Fred Brooks wrote a book called *The Mythical Man-Month: Essays on Software Engineering* ([https://en.wikipedia.org/wiki/The\\_Mythical\\_Man-Month](https://en.wikipedia.org/wiki/The_Mythical_Man-Month)) in 1975. It's a short book and well worth a read, or at the very least the Wikipedia page for the summary of the main points.

Brook's experience came from managing the writing of OS/360. This was the first time when you had 40-50 people writing one OS. Suddenly you really had communication issues between teams. You had entire departments working on different areas together with departmental managers. Fred debunks the interchangeability of the man-month. Read Fred Brooks' essay, which is linked from the course material page.

[Discuss the example in the slide. Here each box represents one month, and the number inside is the number of people working that month.]

## Mythical Man-Month: “adding manpower to a late project makes it later”



We try to catch up:

- Train 3 more developers in the first month, then use all 6 developers in the next month
- But: work of 3 developers in 2 months can't be done by 6 developers in 1 – interaction costs maybe  $O(n^2)$

181

We could try and catch up, but this plan doesn't work since interaction costs between team members will slow down the project. Moreover who is going to do the training? If you use the existing team, then they won't get the work done in the first month dedicated to coding. Hence the adage “adding manpower to a late project makes it later”.

Time to first shipment is cube root of developer-months (Boehm, 1984)

$$T = 2.5\sqrt[3]{d}$$

where  $T$  is time to first shipment and  $d$  is developer months

- With more time, costs rise slowly
- With less time, costs rise sharply
- Hardly any projects succeed at  $\frac{3}{4}T$
- Some projects still fail

182

Barry Boehm conducted empirical studies into the costs of development.

Boehm found that the time to first shipment is the cube root of the number of developer months. So a project requiring 1000 developer-months, it will take  $2.5 * \text{cuberoot}(1000) = 25$  months.

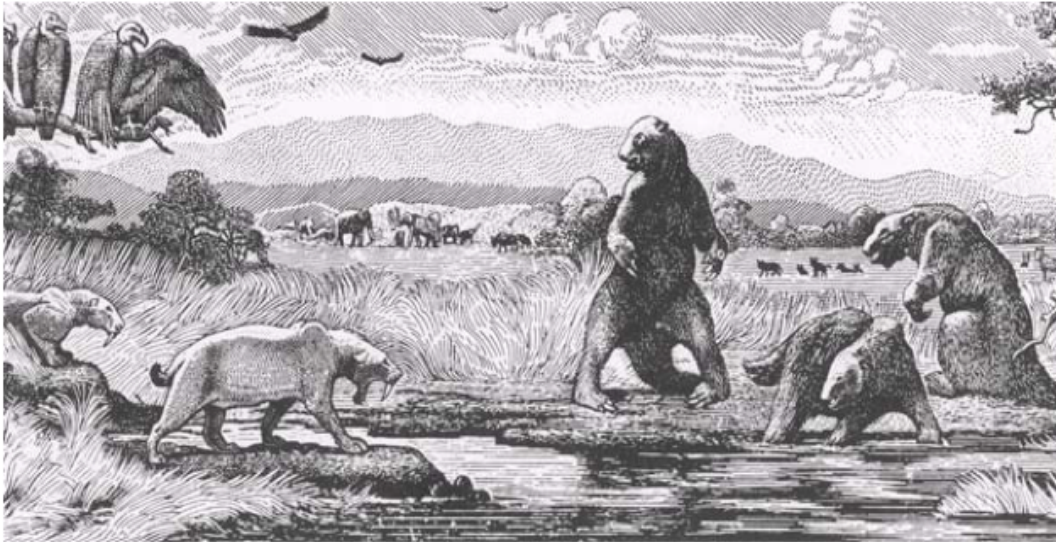
Note that these kinds of estimates are not an exact science. As a result, there is per-project variability around the prediction given by the COCOMO model (which itself is more complex than the cube-root approximation given here). Boehm found that 68% of the projects he studied came within 20% of the actual value given by the model, and there remained significant outliers. With tools like these, you would have thought that we could get at least a reasonable handle on software development. But we still get huge failures which are not predicted by such models.

Additional reading: Boehm, Barry W. "Software engineering economics." *IEEE transactions on Software Engineering* 1 (1984): 4-21.

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5010193>



# The Software Tar Pit



183

Has anyone been to Los Angeles? Well, there you see tar bubbling to the surface. Over the millennia, many animals get stuck (and many extinct species have now been dug out). People describe software in this way: if you fix any one bug in your software, it creates other problems elsewhere. With poorly written software, you can never escape! Just like an animal with four feet can get anyone foot out, but never all four, and moving always pulls the average depth of all feet down...

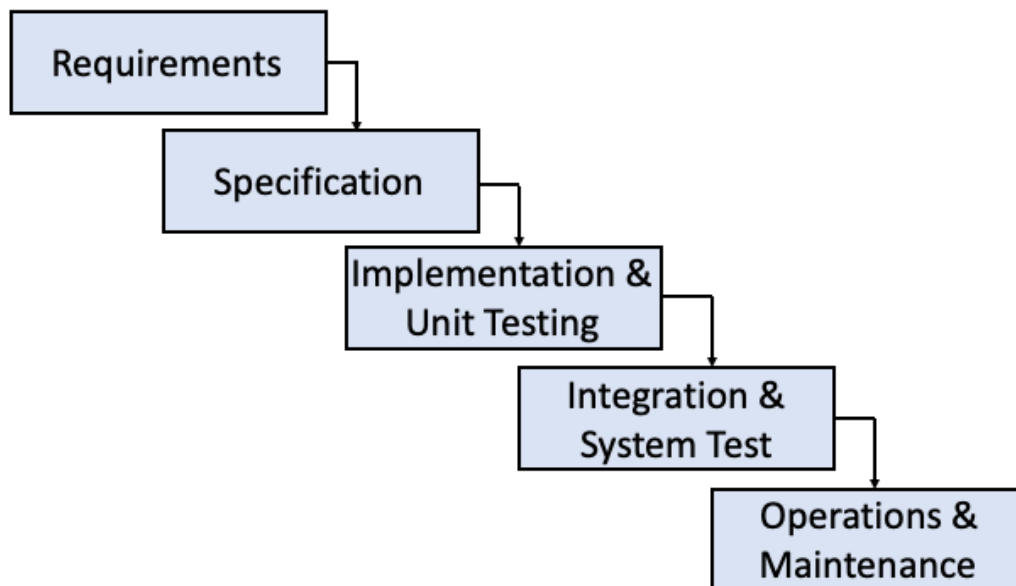
## Take a structured, modular approach

- Only practical way forward is *modularisation*
- Divide a complex system into small components
- Define clear APIs between them
- Lots of methodologies based on this idea:
  - SSDM
  - Jackson
  - Yourdon,
  - UML,
  - ...

184

There are lots of buzz words in this space, but the basic idea is always the same: chop up a complex problem into modules with small, cleanly designed interfaces. The challenge is how, and when, to perform the division.

# The Waterfall Model (1970)



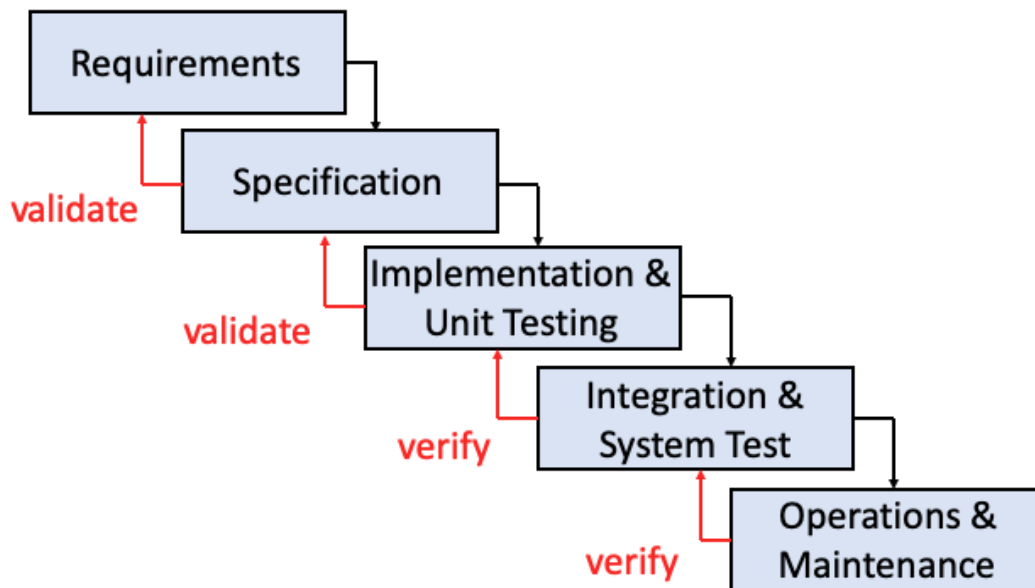
185

Developed by Henry Royce in the 1970s for the USAF. Developed in response to problems with previous software purchases.

Requirements are written in the user's language. For example, if you did this for aircraft, you'd write it in "pilot language". Specification is written in the system language. In other words, the specification should translate the requirements into a language that the programmer can understand. There can be more steps than this; for example, a system spec, a 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. It later became a US DoD standard.

# The Waterfall Model (1970)



186

Feedback is used to validate whether the next stage meets the requirements of the previous. 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 Model has advantages

- Compels early clarification of system goals
- Supports charging for changes to the requirements
- Works well with many management and tech 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)

187

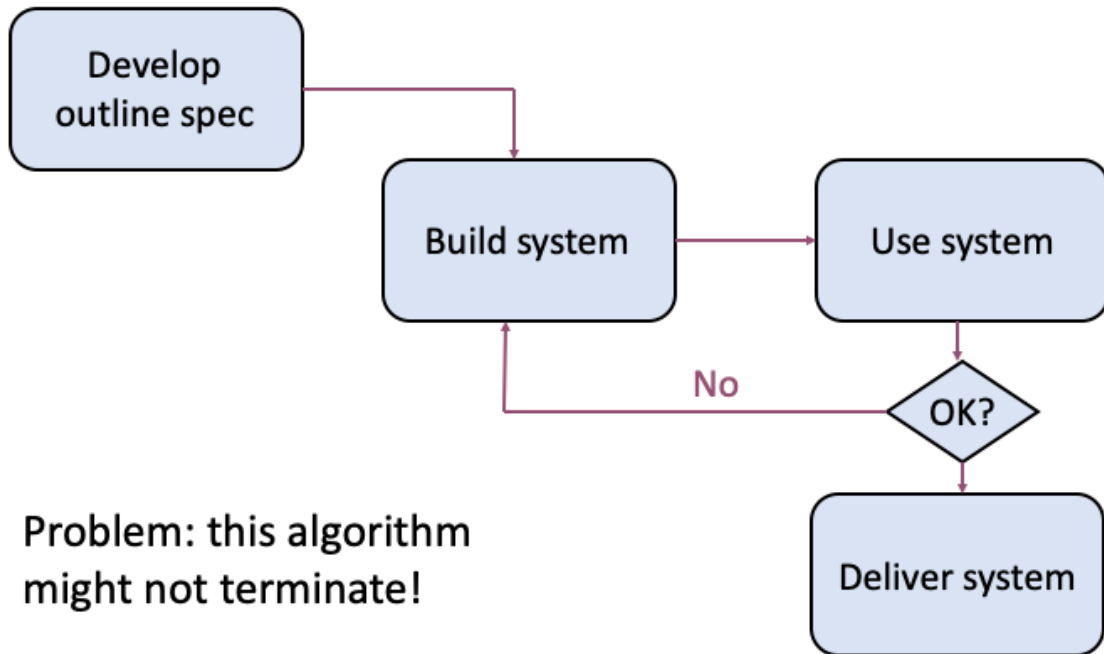
The point around charging is a good one: it provides an evidence base on why system development might cost more or might be delayed. Nevertheless, it can be used to loot naïve customers like government: when the system doesn't work then it's the customer's fault as he signed off the specification.

Government has moved away from using the Waterfall Model for many projects. For example the UK Government Digital Service Standard requires the use of iterative development (more later).

## Waterfall fails where iteration is required, such as:

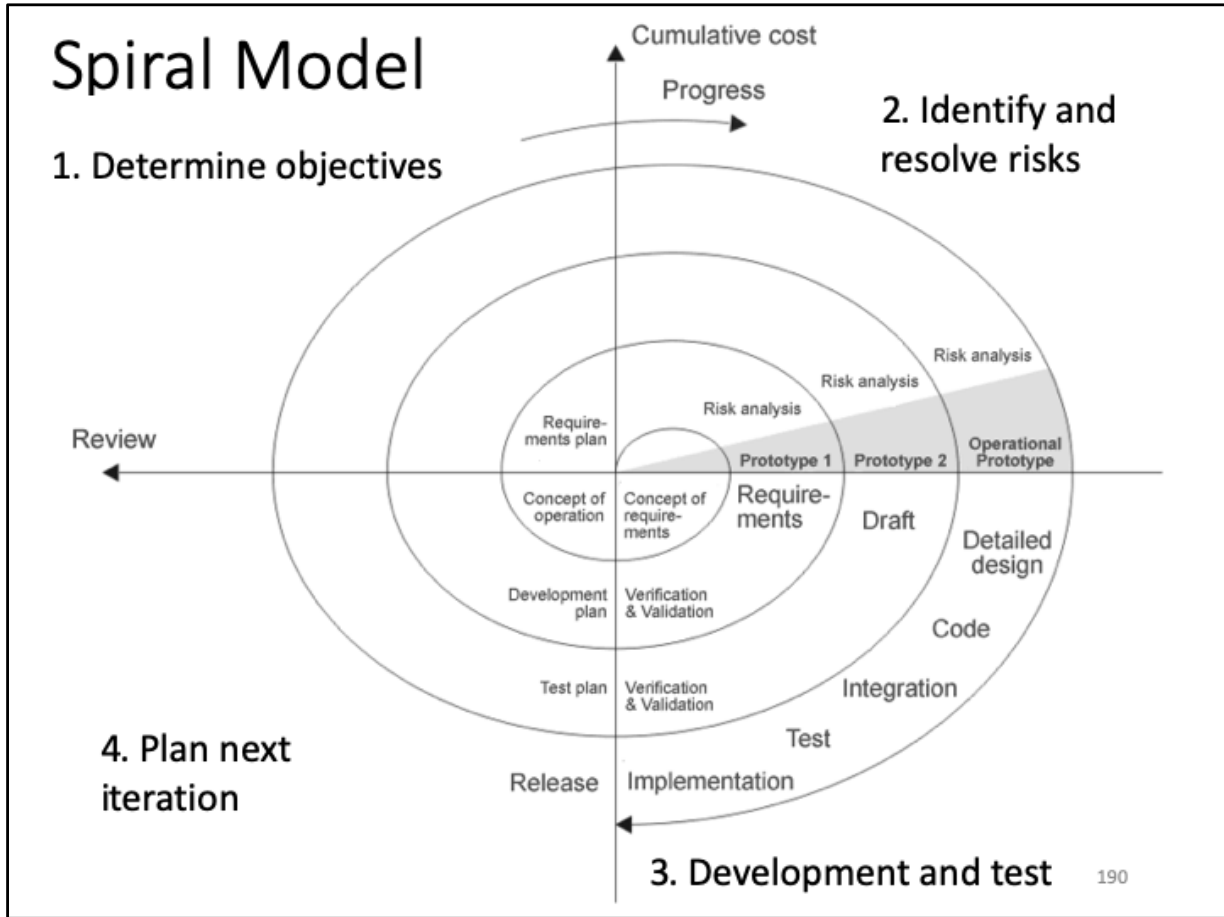
- Requirements not yet understood by developers
- Not yet understood by the customer
- The technology is changing
- The environment (legal, competitive) is changing
- ...

# Iterative development



Problem: this algorithm might not terminate!

189



“This model was first described by Barry Boehm in his 1986 paper "A Spiral Model of Software Development and Enhancement". In 1988 Boehm published a similar paper to a wider audience. These papers introduce a diagram that has been reproduced in many subsequent publications discussing the spiral model.”

[https://en.wikipedia.org/wiki/Spiral\\_model](https://en.wikipedia.org/wiki/Spiral_model)



## Spiral model invariants

- Decide in advance on a fixed number of iterations
- Each iteration is done top-down
- Driven by risk management (i.e. prototype bits you don't yet understand)

# Software and Security Engineering

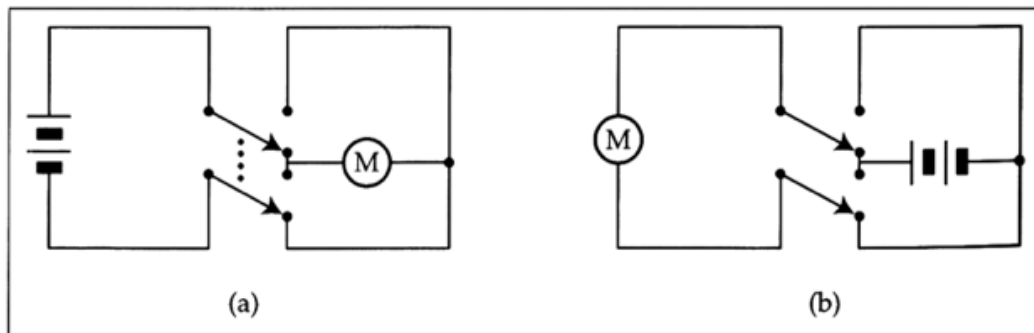
Lecture 8

**Alastair R. Beresford**

arb33@cam.ac.uk

*With many thanks to Ross Anderson*

Warm up: Which motor reversing circuit is the safe?



# Evolutionary model

- By the 1990s some codebases had become so big and complex they *had* to evolve
- Solution: use *automatic regression testing*
- Firms now have huge suites of test cases which run against daily builds of software
- Development cycle is then to add changes, check them into a repository, and test them

194

Microsoft tried to rewrite Word from scratch twice and failed.

Regression testing involves creating a set of test cases which worked on the last released version and are checked against the next version. If the output differs between the two, there's a bug (either in the new version, the old version, or the test). We saw how this approach was developed in detail in Richard Sharp's lecture on Software-as-a-Service. Recall that in some SaaS deployments every commit to the repository passes through testing, and if it passes, on to deployment. Remember also that deployment is staged: we first deploy to a small fraction of the userbase, and then increasing amounts, all the time looking at feedback and metrics on whether the latest release is successful or not.

## The Integrated Development Environment (IDE) includes...


- Code and documentation under version control (Git)
- Code review (Gerrit)
- Automated build system (Maven)
- Continuous integration (Jenkins)
- Dev / Test / Prod deployment (Webserver)

195

If you want to try this whole stack, perhaps over the summer, then there are many tutorials which you can look at to see how these parts come together. Here is one example. Email the course lecturer if you know of a better example (this one is now a bit old):

<https://programmaticponderings.com/2013/11/04/continuous-integration-and-deployment-using-git-maven-jenkins-and-glassfish/>

## Content-heavy apps benefit from four host types

|  |               | Content        |               |
|-----------------------------------------------------------------------------------|---------------|----------------|---------------|
|                                                                                   |               | <i>Latest</i>  | <i>Stable</i> |
| Software                                                                          | <i>Latest</i> | <b>Test</b>    | <b>Dev</b>    |
|                                                                                   | <i>Stable</i> | <b>Staging</b> | <b>Prod</b>   |

196

The concept of the evolutionary model doesn't just have to be applied to code. It can also be applied to curated content too. We run the IsaacPhysics.org site in the department. Here there is a huge amount of curated content which is edited by the content team in a graphical editor which is backed by Git. Content is first entered, then goes through a Quality Assurance (QA) process, then then released. We don't have automated testing of content, so we can't release on every commit. We do have automated regression testing for much of the software, but some still requires a manual process – writing end-to-end regression tests for the web is very expensive in time and money, so we operate on a more traditional two-week release cycle. For the content, the senior team select a commit hash which goes live to different versions of the site. Software and Content releases can be done independently, so we therefore have at least four version of the service at any one point in time: Test, Dev, Staging, Prod.

## Assurance of critical software: must study how things fail

- Critical software avoids 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)

## Tacoma Narrows, 7<sup>th</sup> Nov 1940



<https://www.youtube.com/watch?v=j-zczJXSxnw> 198

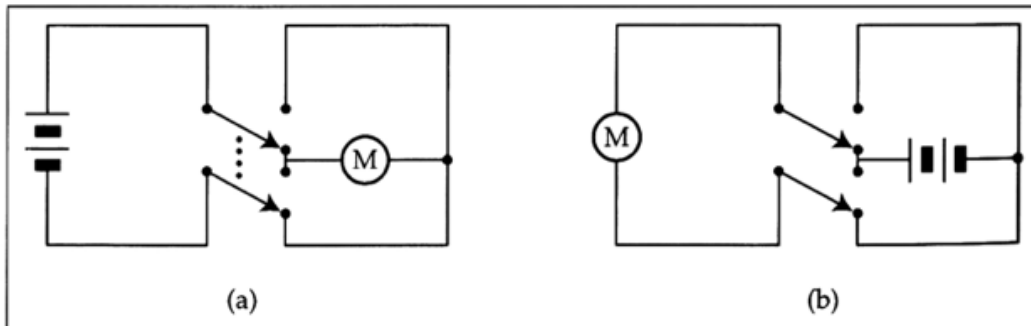
“The first Tacoma Narrows Bridge opened to traffic on July 1, 1940. Its main span collapsed into the Tacoma Narrows four months later on November 7, 1940, at 11:00 a.m. (Pacific time) as a result of aeroelastic flutter caused by a 42 mph (68 km/h) wind ... A contributing factor was its solid sides, not allowing wind to pass through the bridge's deck. Thus, its design allowed the bridge to catch the wind and sway, which ultimately took it down. Its failure also boosted research in the field of bridge aerodynamics and aeroelastic, fields which have influenced the designs of all the world's great long-span bridges built since 1940.”

[https://en.wikipedia.org/wiki/Tacoma\\_Narrows\\_Bridge](https://en.wikipedia.org/wiki/Tacoma_Narrows_Bridge)

It is similarly important in the computer field to look at past failures in order to learn lessons from what can go wrong.



## Hazard elimination



- Which motor reversing circuit is the safe above?
- Some architecture and tool choices can eliminate whole classes of software hazards, e.g. using a garbage collector to eliminate and memory leaks.
- But usually hazards involve more than just software

199

What is the circuit intended to do? How can it fail?

## Ariane 5, 4<sup>th</sup> June 1996



- Ariane 5 accelerated faster than Ariane 4, causing an error in float-to-integer conversion
- The backup inertial navigation set core dumped, which was interpreted by as flight data
- Full nozzle deflection → 20° angle of attack → booster separation

200

# Multi-factor failure

- Many safety-critical systems are also real-time systems used in monitoring or control
- Exception handling is often tricky
- Criticality of timing makes many simple verification techniques inadequate
- Testing is often 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

# Therac-25: radiotherapy machine

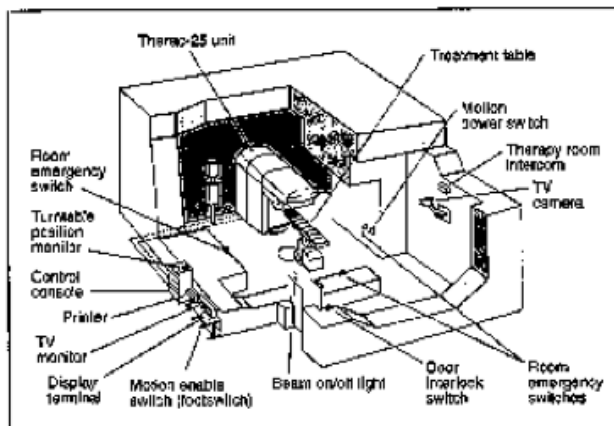


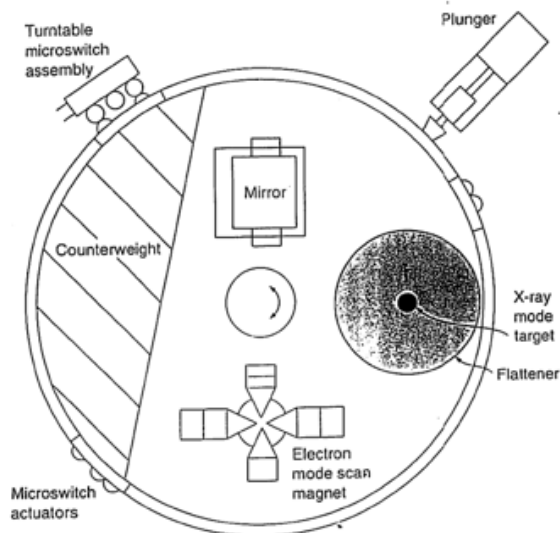
Figure 1. Typical Therac-25 facility.

- Three people died in six accidents
- Example of fatal programming error
- Usability issues
- Poor safety engineering

203

The Therac-25 was a radiotherapy machine sold by AECL. Between 1985 and 1987 three people died in six accidents.

## Therac had two operating modes



- 25 MeV electron focused beam to generate X-rays
- 5-25 MeV spread electron beam for skin treatment

**Safety requirement:**  
don't fire focused beam at humans

204

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)

## Therac-25 used software to enforce safe operation

- Previous models (Therac-6 and 20) used 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 properly documented

## Therac-25 caused injuries

- Marietta, GA, June 1985: woman's shoulder burnt. Settled out of court. FDA not told
- Ontario, July 1985: woman's hip burnt. AECL found microswitch error but could not reproduce fault; changed software anyway
- Yakima, WA, Dec 1985: woman's hip burned. 'Could not be a malfunction'



## Therac-25 killed three people

- East Texas Cancer Centre, March 1986: man burned in neck and died five months later of complications
- Same place, three weeks later: another man burned on the 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, January 1987: man burned on the chest and died due to different bug now thought to have caused Ontario accident

## Therac-25: East Texas deaths due to editing beam type too quickly

|                           |              |              |                  |                       |
|---------------------------|--------------|--------------|------------------|-----------------------|
| 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:              |

208

Clearly this was poor software design. This is a concurrency bug (see earlier in the course).

The Therac-25 also included a "Field light" mode, which allowed the patient to be correctly positioned by illuminating the treatment area with visible light. A second fault allowed the electron beam to activate during field-light mode, during which no beam scanner was active or target was in place. See:

<https://en.wikipedia.org/wiki/Therac-25>

## Therac-25: root cause analysis

- Manufacturer ignored safety aspects of software
- Confusion between reliability and safety
- Lack of defensive design
- Inadequate reporting, follow-up or regulation
- Unrealistic risk assessments
- Inadequate software engineering practices
- Manufacturer left the medical equipment business

209

Good reporting is critical in order to minimize harm. With better reporting, lives might have been saved. In addition, lack of follow-up by the regulator left a second issue seen in the Ontario accident as unexplained and therefore potentially still problematic. This second problem was due to the light field feature of the system. Much went wrong with the software engineering. The specification was an afterthought, machine had a complex architecture, dangerous coding, little testing, careless HCI design. Despite the fact that the manufacturer, AECL, left the medical equipment business, similar accidents are still happening. Poor medical device safety usability still costs many lives. See for example the discussion on infusion pumps at the start of the course.

Further reading: Radiation Offers New Cures, and Ways to Do Harm, 23<sup>rd</sup> January 2010, New York Times.

<https://www.nytimes.com/2010/01/24/health/24radiation.html?hp=&pagewanted=all>

## Software safety myths: cheaper, easy to change, reliable

- Computers are cheaper than analogue devices
  - Shuttle software cost \$10<sup>8</sup> 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

210

The software in the Space Shuttle cost \$100 million a year to maintain.

New electric cars have fewer moving (mechanical) parts, but it has lots of software which requires maintenance. Example: Google in the early years decided that the expensive machines from IBM and Sun were too expensive for a website, so instead got reliable systems through accepting some machines just fail; this still cost a lot! Failover and so on need “devops” teams, testing, and so on.

When the Sizewell B power station was being built, software was suggested as the failsafe mechanism rather than an analogue solution (solder put in to hold cadmium rods which drop in to the core and stop the reaction).

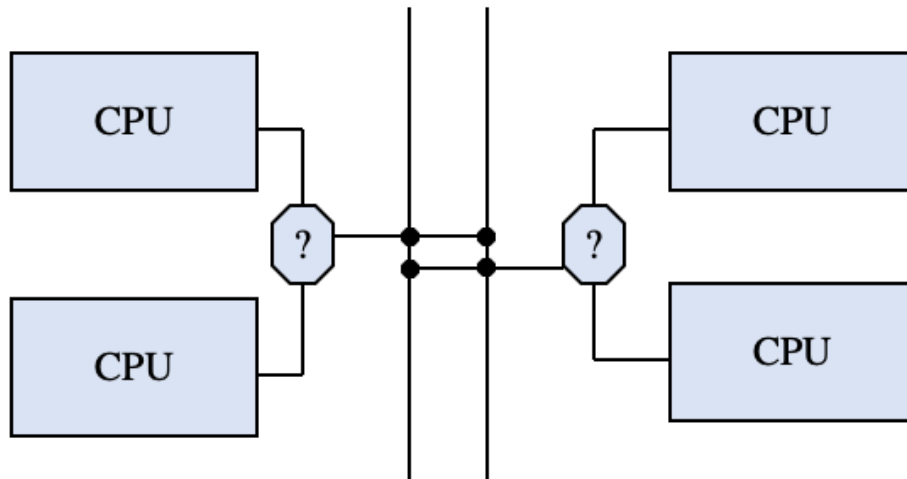
## Software safety myths: reuse, formal methods, testing and automation

- Reuse increases safety
  - Counter examples: Ariane 5, Patriot and Therac-25
- 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
  - Also an opportunity for new types of failure

211

Automation: what happens if kids start running out in front of automated cars and forcing them to perform an emergency stop; this is great until the kids run out in front of a car still under human control and not notice...

## Stratus computer: redundant hardware for non-stop processing



212

Stratus worked with four CPUs. Two CPUs per card. Hardware logic checked the output on every cycle from each CPU; if the output agreed, then all continued as normal. If the CPUs disagreed, then the one in disagreement was left out of future decisions. On release, the share price of Status shot up and their machines were resold under the IBM System/88 brand between 1985 and 1993. This was, for a while, really popular. Why did it not continue to be popular?

[https://en.wikipedia.org/wiki/Stratus\\_Technologies](https://en.wikipedia.org/wiki/Stratus_Technologies)

## Redundant hardware does not solve software engineering issues

- Hardware can still fail; backup inertial navigation failed first on the Ariane rocket
- Redundant hardware creates additional software engineering issues
- Redundant software (*multi-version programming*) sounds promising...
- But: errors are correlated, dominated by failure to understand requirements (Leveson)
- Implementations often give different answers

213

Redundant hardware does not solve any software problems. Indeed, sometimes it makes the situation hard. For example, on the IBM System/88, there was the issue of the lack of familiarity of the programming environment (Fortran not Cobol; odd Operating System) so there were lots of bugs and so the system was not reliable; solution is to make sure such systems support the well-understood environment familiar to the developers.

Leveson in the 1990s did a study where she asked students to write different implementations and then compared them to see if running multiple versions would improve reliability. Leveson found that bugs were not random and identically distributed (so two implementations with a MTTF of 1 in 1000 hours does not give a Mean-Time-To-Failure of 1 in a million hours). The issue is that the specification was not redundant. Can't even solve with two specs: how do you then meaningfully compare them? When the systems give different answers, how do you choose between them? Average them ?!

## Redundancy in the Boeing 737



[Ask the audience to list all the examples of redundancy; what's the most important one?]

Failure could be from a pilot dying – we might see two cases a year; so we have controls for two people, with multiple redundant controls.

A critical piece of interface is the artificial horizon, so you can fly level and straight in cloud or at night. Otherwise a gentle bank might feel like you're upright and then slowly spiral to your death when you hit the ground.



## Panama crash with 47 fatalities 6<sup>th</sup> June 1992



Lower photo: CC-BY-SA Markus Vitzethum

- Need to know which way up
- New EFIS (each pilot), WW2 artificial horizon (top right)
- EFIS failed due to loose wire
- Both EFIS fed off same inertial navigation set
- Pilots watched EFIS, not AH
- And again: Korean Air cargo 747, Stansted 22<sup>nd</sup> Dec 1999

215

All 47 passengers and crew died.

[https://en.wikipedia.org/wiki/Copa\\_Airlines\\_Flight\\_201](https://en.wikipedia.org/wiki/Copa_Airlines_Flight_201)

EFIS: [https://en.wikipedia.org/wiki/Electronic\\_flight\\_instrument\\_system](https://en.wikipedia.org/wiki/Electronic_flight_instrument_system)

Lower picture is from the Wikipedia page. It's the Primary Flight Display of a Boeing 747-400.

Need to know which way is up! One of the electronic systems failed due to failure of a gyro which was wired into both displays (single point of control) and the pilot did not check against the backup system. So, even when we have backup instruments, and professional crew, we still get errors: it's difficult to manage redundancy. If you're a low-hours private pilot then a single engine might be better than a twin: the private pilot has so little experience at landing with only a single engine working with the twin-engine plane that he will screw up, where as in a single-engine plane, then the pilot knows he must land and does so safely in the next field.

An interesting article written by a software engineer and Cessna pilot on the recent issues with the Boeing 737-Max

<https://spectrum.ieee.org/aerospace/aviation/how-the-boeing-737-max-disaster-looks-to-a-software-developer>

## Kegworth crash, 47 fatalities 8<sup>th</sup> January 1989



- Fan blade broke
- Crew shutdown wrong engine
- Emergency landing at East Midlands
- Opened throttle on final approach: no power
- Initially blamed wiring; later cockpit design

216

47 dead, 74 injured.

[https://en.wikipedia.org/wiki/Kegworth\\_air\\_disaster](https://en.wikipedia.org/wiki/Kegworth_air_disaster)

Pilots shutdown the wrong engine, but didn't notice as they also throttled back on the (broken) engine as well as shutting down the working engine. Then on final approach, open up throttle on what was thought to be the right engine, but it was the broken one, so there was no power and the plane crashed. The pilot claimed he had throttled back on the right engine, so blamed the wiring rig to suggest that the throttle cables was swapped over. They found the technician, and it turned out that there were records in the logs that he had worked on the cables, found marijuana in his locker and said it was his fault and fired him. Later crash investigation showed that the cables were wired the right way around.

## Aviation is actually an easy case

- It's a mature evolved system
- Stable components: aircraft design, avionics design, pilot training, air traffic control ...
- Interfaces are stable
- Crew capabilities are well known
- The whole system has good incentives for learning – much better than with medical devices
- Excellent regulation and reporting

*Still complex social-technical system that exhibits failure*

217

# Understand and prioritise hazards

Example from the motor industry:

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 at worst severe
4. *Distracting*; normal response limits and outcome to minor
5. *Nuisance*: affects customer satisfaction but not normally safety

218

Ross was driving on cruise control in a hire car when he hit the "slow down" button on approach to the freeway, but the road was going down hill, and the car speed up slightly, this upward increase in speed confused the cruise control and the car went into max acceleration! This was later found to be a software bug.

# Managing safety and security across the software lifecycle

- Develop a safety case or security policy
- Design a management plan
- Identify critical components
- Develop test plans, procedures, training
- Plan for and obtain certification
- Integrate all the above into your development methodology (waterfall, spiral, evolutionary, ...)

219

Managing a critical property such as safety, security or even hard real-time guarantees is hard.

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

## Most mistakes occur outside the technical phases

Challenging parts are often:

- Requirements engineering
- Certification
- Operations
- Maintenance

This is due to the interdisciplinary nature of these parts, involving technical staff, domain experts, users, cognitive factors, politics, marketing, ...

220

While failures can and do occur during the technical phases of design and implementation, root cause analyses will reveal mistakes before (specification) or after (testing, deployment). The soft spots are therefore requirements engineering, certification, operations and maintenance. All these soft spots are hard because they are interdisciplinary.

The actual areas where you have failures are in the specification (because you don't know the environment in which the product operates); or after release (because the environment changes, or it's popular and gets used in new places you didn't expect). For example, Facebook was designed for students at Harvard; now it's used by everyone, and this means it can be used for manipulation (e.g. elections) or used for virtual harassment and cyberbullying.

## The Internet of Things: safety now includes security

- Cars, medical devices, electricity grid all have 10+ year lifetimes as well as formal certification
- All contain software; will be Internet connected
- Apparent conflict between safety and security
  - E.g. first DDoS attack (Panix ISP) was from driven from hacked Unix machines with medical certification
- Good security requires us to move to monthly patching, yet this conflicts with the safety case

221

Connectivity to the internet changes everything! Need lots of updates. Panix was an ISP in New York which supported unions, and other left-wing organisations; it suffered from a massive DDoS attack. This was carried out by bad guys noticing that hospitals had certifications for Unix machines connected to the Internet. These machines could not be upgraded without invalidating their certification, so known bugs were not patched. These machines were then used to DDoS Panix.

The move to security patching will cause lots of stress for regulators. How do they adapt to this brave new world?

# Software engineering tools help us manage complexity

Homo sapiens uses tools when some parameter of a task exceeds our native capacity. So:

- Heavy object: raise with lever
- Tough object: cut with axe
- ...
- Software complexity: ?



## Good tools eliminate *incidental* and manage *intrinsic* complexity

*Incidental* complexity: dominated programming in the early days, including writing programs in assembly. Better tools eliminate such problems.

*Intrinsic* complexity: the main problem today, since we now write complex systems with big teams. There are no solutions, but tools help, including structured development, project management tools, ...

223

Example is that writing code in assembler may mean you have to remember which variable is at which address, such as where the accelerometer x value is stored at memory address 0x48; we can fix this with better programming languages and tools. Incidental complexity can be eliminated; intrinsic complexity must be managed.

## High-level languages remove incidental complexity

- 2 KLOC per year goes much farther than assembler
- Code easier to understand and maintain
- Appropriate abstraction: data structures, functions, objects rather than bits, registers, branches
- Structure finds many errors at compile time
- Code may be portable; or at least, the machine-specific details can be contained

**Huge performance gains possible, now realised**

224

We have seen huge performance gains from using high-level languages (e.g. Java) over low-level ones (e.g. assembly). While there are new languages coming out all time, we expect to see more modest improvements in the future. Unless, that is, something really exciting and new comes out in the world of programming language design and implementation. Remember, of course, that coding is only a fraction of overall effort in a project, so improving the programming language is not the only part where improvements can occur. Its worth pointing out that a high-level language, can also offer advantages in testing and maintenance as well...

# High-level languages support structure and componentisation

Much historical work on both languages and language features, including:

- “*Goto statement considered harmful*” (Dijkstra, 1968)
- Structured programming with Pascal (Wirth, 1971)
- Object-oriented programming (see OOP course)
- ...

Don't forget: this is to *manage intrinsic complexity*

225

A big win with high-level languages comes from improving maintenance. In the old days, goto was used a lot to make programs run faster, but it also made them entirely unmaintainable. Variables in older programming languages may have overly large, or even global scope, so updating a variable in an inner loop which happens to be the variable for the outer loop means that your loop update is broken.

[https://en.wikipedia.org/wiki/Considered\\_harmful](https://en.wikipedia.org/wiki/Considered_harmful)

# Formal methods find bugs, but it is fallible

## History:

- Turing talked about proving programs correct
- Floyd-Hoare logic; Floyd (1967), Hoare (1969)
- HOL; Gordon (1988)
- Z notation
- BAN logic
- ...

226

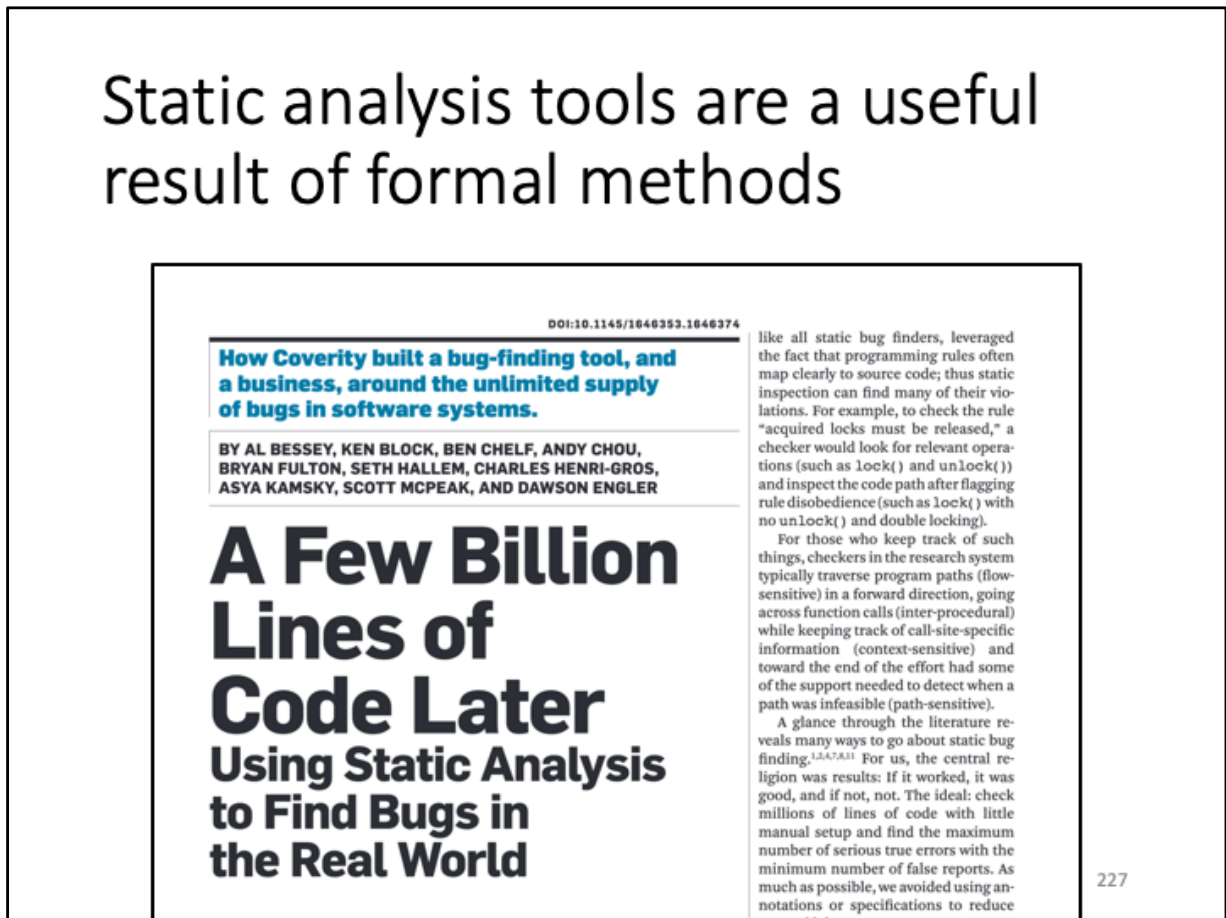
“In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.” [https://en.wikipedia.org/wiki/Formal\\_verification](https://en.wikipedia.org/wiki/Formal_verification)

Mike Gordon designed and led the team who built HOL88,  
[https://en.wikipedia.org/wiki/Formal\\_verification](https://en.wikipedia.org/wiki/Formal_verification)

[https://en.wikipedia.org/wiki/Hoare\\_logic](https://en.wikipedia.org/wiki/Hoare_logic)

BAN: 1989 saw this as a way of verifying cryptographic protocol; was the highest cited CS paper in the 1990s; really useful for finding certain kinds of bugs really well, but it doesn't find all of them. We have many examples of programs proved to be correct, but still wrong (mostly because of incorrect assumptions). Particularly useful for small program representing challenging algorithms. [https://en.wikipedia.org/wiki/Burrows-Abadi-Needham\\_logic](https://en.wikipedia.org/wiki/Burrows-Abadi-Needham_logic)

# Static analysis tools are a useful result of formal methods



The Coverity tool is worth looking at since there is a nice write-up of the challenges in providing tools in this space:

Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, Dawson Engler  
Communications of the ACM, February 2010, Vol. 53 No. 2, Pages 66-75  
<https://cacm.acm.org/magazines/2010/2/69354-a-few-billion-lines-of-code-later/fulltext>

The paper contains many interesting insights in taking a research idea into a product. For example, when releasing a new version of the tool, it suddenly finds many more problems than the last, even on the same code base. So if you're a manager, you find that the bug count on your product goes up, which is disheartening at best!

# Software and Security Engineering

Lecture 9

**Alastair R. Beresford**

arb33@cam.ac.uk

*With many thanks to Ross Anderson*

228

This is the last lecture in the series.

## Chief programmers (IBM, 1970s)

**Aim: avoid loss of great programmers to management and capitalise on wide productivity variance**

- **Teams consisting of chief programmer, apprentice, toolsmith, librarian, admin assistant, etc.**
- **Can be effective during implementation**
- **But each team can only do so much**

229

Now for a slight change in topic. Let's look at team structures.

A problem at IBM was that good programmers were promoted to managers (so they were paid more and got a better parking space). This meant that all the good programmers were no longer programmers! Productivity suffered. IBM then took the idea of chief programmers who got paid like a manager and were given some juniors to look after (and therefore kept the good programmers actually writing code).

## Egoless programming: minimize personal factors (Weinberg, 1971)

- Code should be owned by the team
- Direct opposite to the Chief Programmer approach
- Groupthink can entrench bad practice deeply

230

[https://en.wikipedia.org/wiki/Egoless\\_programming](https://en.wikipedia.org/wiki/Egoless_programming)



## Literate programming (Knuth, 1984)

- Treat programs as literature, readable by humans
- Primarily a work of literature, with code added
- Literate programs are compiled in two ways:
  - *Weaving*: a comprehensive human-readable document about the program and its maintenance.
  - *Tangling*: the machine executable code
- Literate programming is *not* documentation embedded in code, such as Javadoc.

231

[https://en.wikipedia.org/wiki/Literate\\_programming](https://en.wikipedia.org/wiki/Literate_programming)

On the last point, Javadoc (<https://en.wikipedia.org/wiki/Javadoc>) is *not* an example of literate programming; emphasis is the wrong way round. The primary “document” is the human readable one; code is a surrogate, not the primary artefact. Quote from literate programming wiki:

“This misconception has led to claims that comment-extraction tools, such as the Perl Plain Old Documentation or Java Javadoc systems, are "literate programming tools". However, because these tools do not implement the "web of abstract concepts" hiding behind the system of natural-language macros, or provide an ability to change the order of the source code from a machine-imposed sequence to one convenient to the human mind, they cannot properly be called literate programming tools in the sense intended by Knuth.”

Aside: Knuth also famously rewarded people for bugs (e.g. hexadecimal dollar)  
[https://en.wikipedia.org/wiki/Knuth\\_reward\\_check](https://en.wikipedia.org/wiki/Knuth_reward_check)

# Capability Maturity Model (Humphrey, 1989)

1. **Initial** (chaotic, ad hoc, individual heroics) – 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

232

Aim: Nurture the capability for repeatable, manageable performance, not outcomes that depend on individual heroics.

Do so by recognising the fact that people work better with people they know and therefore the team progress through these levels. You can't have teams that are static forever however -- staff come and leave, you need to onboard new people, it's good to rotate people so they have experience in several teams, and so on. It identifies five levels of increasing maturity in a team or organisation, and a guide for moving up.

[https://en.wikipedia.org/wiki/Capability\\_Maturity\\_Model](https://en.wikipedia.org/wiki/Capability_Maturity_Model)

Described in the book by Watts Humphrey, *Managing the Software Process*, 1989.

## Extreme programming (Beck, 1999)

- Iterative development with short cycles
- Automated build and test suites
- Frequent points to integrate new requirements
- Solve the worst problem, repeat
- Avoid programming a feature until needed
- Programming in pairs, one keyboard and screen
- Extensive code review

233

[https://en.wikipedia.org/wiki/Extreme\\_programming](https://en.wikipedia.org/wiki/Extreme_programming)

Extreme programming no longer popular, but inspired Agile software development.

See also: Kent Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999

# Agile software development (2001)

Four values:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Also twelve principles (see related work), including frequent release, daily meetings, working software as measure of progress, regular reflection, etc.

234

Agile now has many variations: it's a family of related approaches which typically share a common set of values and principles.

[https://en.wikipedia.org/wiki/Agile\\_software\\_development](https://en.wikipedia.org/wiki/Agile_software_development)  
<http://agilemanifesto.org/>

# The specification still matters

Curtis (1988) found causes of failure were:

1. Thin spread of application domain knowledge
2. Fluctuating and conflicting requirements
3. Breakdown of communication, coordination

Causes were very often linked, and the typical progression to disaster was 1 → 2 → 3

235

The Curtis study looked at the failure of 17 large demanding systems. These three failures are often linked, so if you look at the case studies such as London Ambulance Service or Smart Meters you'll see these.

Further reading: Curtis, Bill, Herb Krasner, and Neil Iscoe. "A field study of the software design process for large systems." *Communications of the ACM* 31.11 (1988): 1268-1287. <https://dl.acm.org/citation.cfm?id=50089>

## Specification is hard: thin spread of application domain knowledge

- How many people understand everything about running a phone service, bank or hospital?
- Many aspects are jealously guarded secrets
- Some fields try hard to be open, e.g. aviation
- With luck you might find a real 'guru'
- You should expect mistakes in specification

236

You need to get many people together to understand a large system (e.g. phone service or bank). How many? Who? There's a chance of significant mistakes. In some sense it doesn't matter whether you have the Waterfall Model, Spiral Model or Agile to determine this list. It's always hard. For any long-lived product, you need to continue to adjust.

## Specification is hard: fluctuating and conflicting requirements

- Competing products, new standards, fashion
- Changing environment (takeover, election, ...)
- New customers (e.g. overseas) with new needs
- ...

237

Facebook realised that kids were not using Facebook because their parents are on there, so Facebook bought Instagram; then messaging was a risk, so Facebook bought WhatsApp. All these things changes the specification.

## The specification can kill you

- Spec-driven development of large systems leads to communication problems since  $N$  people means  $N(N-1)/2$  channels and  $2^N$  subgroups
- Big firms have hierarchy; if info flows via 'least common manager', bandwidth will be inadequate
- Proliferation of committees, staff departments causing politicking, blame shifting
- Management attempts to gain control result in restricting many interfaces, e.g. to the customer

238

If you have a big system with  $N$  people then there are  $O(N^2)$  channels and  $2^N$  subgroups. How do you resolve disputes with teams of 1,000 people? With committees, people have preferences, so  $Y$  gets put to the back of the queue, so won't get done for a year; but  $X$  relies on  $Y$ , and  $X$  is important. Look at Government and Universal Credit: the government declared it would use Agile development, but this didn't really happen because in a large organisation there is significant top-down control.



# Project management: plan, motivate, control

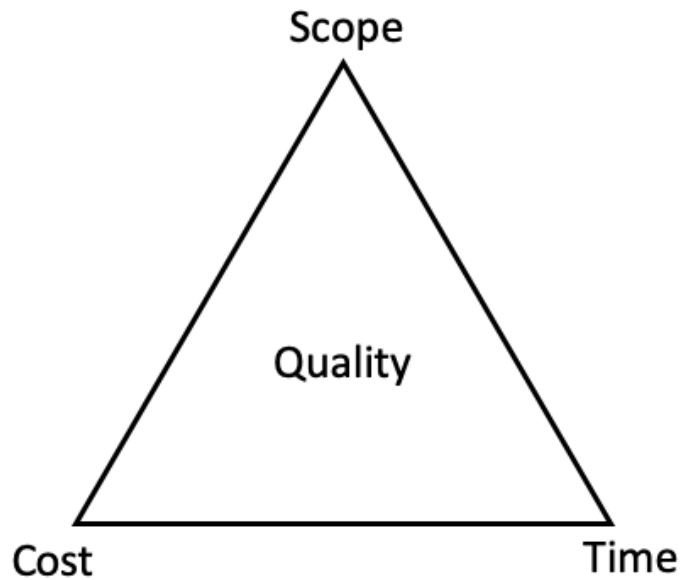
A manager's job is to:

- Plan
  - Motivate
  - Control
- 
- The skills involved are interpersonal, not technical; but managers must retain respect of technical staff
  - Growing software managers a perpetual problem! (Managing programmers is like herding cats.)
  - Nonetheless there are some tools that can help

239

We need this, even with Agile development, since dev work often takes place in large organisations.

# Project management triangle

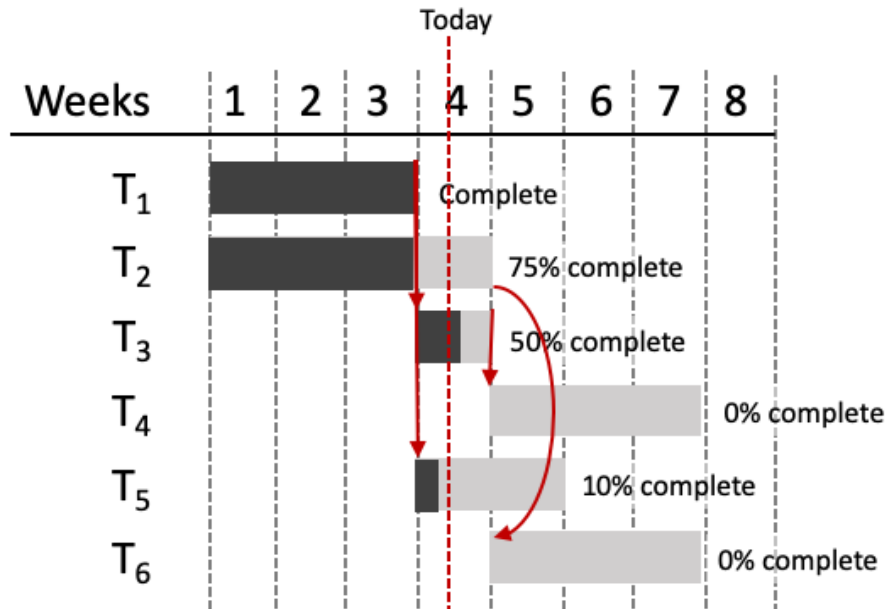


240

The project management triangle, or triple constraint, iron triangle, models the constraints of project management. The main point is that the quality of work is constrained by budget, deadline and scope. The job of the project manager is look at the trade-offs and aim for a suitable one.

[https://en.wikipedia.org/wiki/Project\\_management\\_triangle](https://en.wikipedia.org/wiki/Project_management_triangle)

## Gantt charts: tasks and milestones



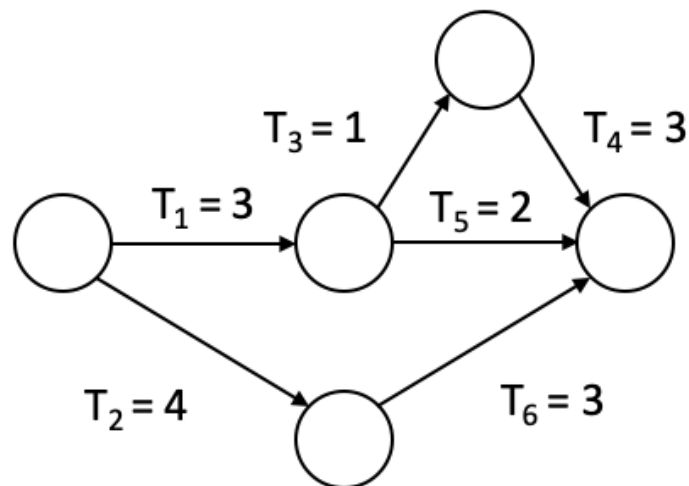
Can be hard to visualise dependencies in large charts

241

“A Gantt chart is a type of bar chart that illustrates a project schedule, named after its inventor, Henry Gantt (1861–1919), who designed such a chart around the years 1910–1915. Modern Gantt charts also show the dependency relationships between activities and current schedule status.” ([https://en.wikipedia.org/wiki/Gantt\\_chart](https://en.wikipedia.org/wiki/Gantt_chart))

Here is one example. [Describe the example.] This kind of chart helps maintain the ‘hustle’ in a project and warns of approaching trouble. Warning: if you can't split a job down into units of work which are ~2 weeks in length, then you can't actually produce a reliable estimate. You can transform your Gantt chat into a PERT chart...

## PERT charts: show critical paths



Which paths are critical?

242

Project Evaluation and Review Technique (PERT): draw as a graph with dependencies. Highlights critical paths: those paths in the process which if delayed lengthen the delivery of the entire project.

Which edges are on a critical path?

The critical paths here are,  $T_1, T_3, T_4$  and  $T_2, T_6$ . Conversely there is some slack in  $T_5$  since it may start as early as the start of week 4 or as late as the start of week 6.

# Motivating people in groups

- People can slack in groups (*free rider, social loafing*)
- Competition no good: people who don't think they will 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

243

Don't abuse minorities and women by demeaning them and only ever assigning them roles such as writing the documentation or testing. (This used to be a problem in industry -- it never was the right approach.)

Many other factors: acknowledgement, attribution, equity, leadership, and 'team building' (shared food / drink / exercise; scrumming). Acknowledgement -- make sure that everyone gets credit publicly if possible; e.g. on the website. Equity: make sure that people get their own picks of tasks first in rotation. Team building is important -- make sure they know and trust one another with activities such as paint balling, etc.

Acknowledgement and gender discrimination remain a problem which we need to accept is happening and fix. The onus is on all of us to change attitudes and actively fight discrimination. Katie Bouman is one recent example.

<https://www.theguardian.com/commentisfree/2019/apr/17/katie-bouman-black-hole-image-online-trolls>

## Testing: half the effort (and cost)

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!**

244

Testing is often neglected in academia.

Bill Gates is quoted as saying “are we in the business of writing software, or test harnesses?”

Andy Rice talked about this in terms of software engineering. Spiral or Agile development in the early phase is an important form of testing (e.g. design validation, UX prototyping) since you're testing your spec and requirements. There's now the phrase "dog fooding" or "eat your own dog food" where you trial software on your own employees. There is also dark launch or A/B testing where you roll out to a small number (see SaaS). Development economics says you need to get rid of as many bugs as possible early on.

## Design for testability, use CI and automate regression testing

**Regression 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 does not work
- 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, tests *are* the spec

245

Huge advances in modern software systems. Design for testing, use continuous integration and automate regression testing.

In the old days, 20% of bug fixes would reintroduce old bugs! Another thing that you can do is collect lots of test data before you've even built it. For example, you can drive one billion kilometres on the roads to collect environmental parameters such as how good the cameras are, when dogs run into the road, etc. You can then use this data to test the system, but also in your defence in your court case later to show a good, sensible strategy.

## A MTBF of $x$ requires testing for $x$

- Reliability growth models help us assess MTBF, number of bugs remaining, use in further testing, ...
- Probability,  $p_i$ , that a particular defect remains after  $t$  tests is:

$$p_i = e^{-E_i t}$$

where  $E_i$  is the virility of the defect

- Yet in large systems, likelihood that the  $t$ -th test fails is proportional to  $k/t$ , where  $k$  is a constant.

**Take away: for  $10^4$  hours MTBF, must test  $>10^4$  hours**

246

[Note: this slide has been updated since printing to give a clearer explanation.]

Researchers have shown that systems with a single bug, or a small number of bugs, are governed by Poisson survival statistics [1], namely that the probability  $p_i$  that a particular defect remains undetected after  $t$  tests is given by  $p_i = \exp(-E_i t)$  where  $E_i$  is the virility of the defect and depends on the proportion of the input space that it affects.

“The problem is that extensive empirical investigations have shown that in large systems, the likelihood that the  $t$ -th test fails is not proportional to  $\exp(-Et)$  but to  $k/t$  for some constant  $k$ ” [2]. This can be shown to be the case through an analysis of statistical thermodynamics [2], the details of which are beyond the scope of this course.

The take home message is if we need a mean time to failure of 10,000 hours for a piece of software, then we need to test it for at least 10,000 hours (see [3]). This is of importance for public policy, as it impinges on the use of software in applications where very high mean times to failure are required, such as nuclear plant control and aircraft flight systems. It turns out that this is the *best* mean time to failure that we can expect given such a level of testing, and that the reliability will usually be less [2].

[1] Adams E. N., Optimising preventive maintenance of software products, IBM

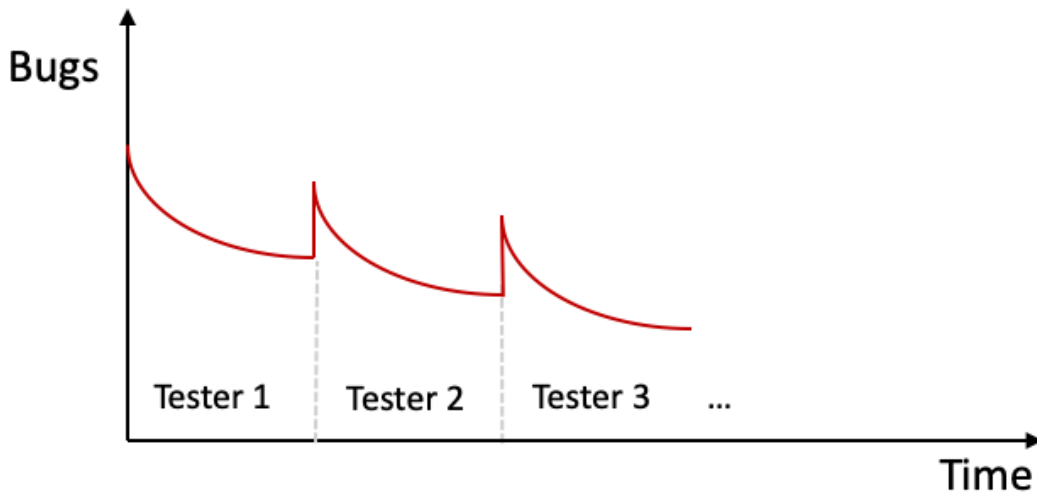


Journal of Research & Development , Vol. 28, issue 1 pp 2–14 (1984)

[2] Robert M. Brady, Ross J. Anderson, Robin C. Ball, Murphy', Murphy's law, the fitness of evolving species, and the limits of software reliability. University of Cambridge Computer Lab Technical Report 471, September 1999.  
<https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-471.pdf>

[3] Butler R. W., Finelli G. B. The infeasibility of experimental quantification of life-critical software reliability, ACM Symposium on Software for Critical Systems, New Orleans ISBN 0-89791-455-4 pp 66–76 (Dec 1991)

## Changing testers finds more bugs



247

Tester 2 has a different area of focus than Tester 1, so they explore a new part of the space of possible bugs. This suggests that you want many testers in parallel who either naturally or by design have a different test focus.

# 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

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

### Our automated check suggests

- we can't find the outline of your head



248

Cat Hallam. Used with permission. See:  
<https://twitter.com/CatHallam1/status/1114590857397788673>

## Tests should exercise the conditions when system is in use

- Many failures result from unforeseen input or environment conditions (e.g. Patriot)
- Random testing – *fuzzing* – now good practice
- Incentives matter: commercial developers look for friendly certifiers, while military, NASA, DoE arrange hostile review
- So: to whom do you have to prove what?

249

Remember to think about requirements -- Patriot was a problem because the spec said work for 4 hours, but it was running for 100s of hours in the field. Random testing: feed in many millions of inputs until you get the system to crash. Hostile review: pay reviewers \$5000 for each bug found; bug bounty programmes are also hostile in this sense.

## Keeping all documents in sync is hard

- How will you deal with management documents (budgets, PERT charts, staff schedules)?
- Engineering documents (requirements, hazard analyses, specifications, test plans, code)?
- Possible partial solutions:
  - High tech: integrated development environment
  - Bureaucratic: plans and controls department
  - Social consensus: style, comments, formatting

250

While the Agile community think that the testing is the documentation, this is not sufficient. Keeping all these different systems together is hard: keeping the spec, safety case, code documentation, testing frameworks, PERT charts, and so on together and cross-reference each other. An IDE is one approach. Some industries use a “plans and controls department” who were empowered to check that backup and recovery actually worked.

# Release management: from development code to production



- Main focus is on stability
- Add copy protection, rights management
- Critical decision: patch old version or force upgrade?

251

Particularly hard when you've got a safety case to do as well since this requires coordination. You may need to destructively test cars! Note also that emergency releases are hard (e.g. Heartbleed) since there's intense time pressure.

## 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
  - Manage backup
  - Recovery
  - Rollback
  - ...
- DevOps integrates development and operations

252

Need an idea centrally for many big systems. For example, with Patch Tuesday, someone needs to check that all the existing software runs on the new version of Windows. So you need to know how all the machines are configured.

In recent years, there's been a move to integrate software development and operations: <https://en.wikipedia.org/wiki/DevOps>

## Vulnerability disclosure: the modern consensus is *coordinated disclosure*

Possible options for discoverer:

1. Disclose without notice: a *zero day*
2. Publicly disclose after a fixed delay: *coordinated or responsible disclosure*
3. Publicly disclose after vendor fix
4. No disclosure, but then vendor can't fix

Vendors use bug bounty programmes to discourage 1.

253

20 years ago lawyers would suggest you deny the existence of bugs; minimise corporate risk. We saw, for example, that researchers at the University of Birmingham were sued by VW over their hacks into VW vehicles. As a result, hackers would boast about them and post exploits on underground forums rather than reporting to vendors.

CERT performs a useful service: you can report to CERT and CERT then contacts the vendor, so they do the "you have 90 days warning". It's good for researchers, as you are not going to get sued. It's better for the ecosystem -- random bugs aren't posted on underground forums.

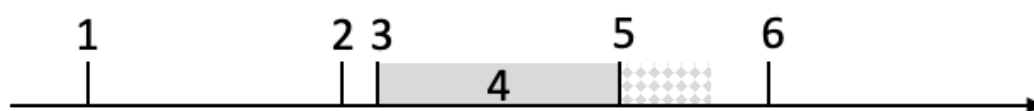
When the discoverer is also an employee of the vendor, then second option may not be possible, but the threat that an external person may find in the future may encourage option 3.

Google Project Zero operates a strict 90-day disclosure period.

[https://en.wikipedia.org/wiki/Project\\_Zero](https://en.wikipedia.org/wiki/Project_Zero)



# Vulnerability lifecycle



1. Engineer introduces a bug
2. Someone discovers it
3. Coordinated disclosure; disclose at once; or exploit
4. Primary exploit window
5. Patch released
6. Public notification of bug

- What about orphaned devices or Mirai?

254

Many devices simply don't get patched. Recall earlier the issues with Android vulnerabilities. Mirai is another example: Internet home hubs and IP cameras are not patched since there is little incentive for the manufacturers to do anything.

Note that there is a long tail of installing updates. WannaCry with the NHS was an example of this, where machines weren't updated since there were concerns about managing updates; it was easier not to (or not to pay Microsoft lots of money).

Solutions? Using the CE mark is may provide one means of fixing the problems: self-certify to say you will provide updates, and then goods on import can be rejected and returned if the vendor has failed to do this on previous products.

## Shared infrastructure provides benefits & implies responsibilities

- 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?
- How do you feed your fixes back to others?
- Do you encourage coordinated disclosure?
- Are you aware of different license terms?

255

Example emergency would be Heartbleed.

## Beware of agency issues

- Employees often optimize their own utility, not project utility (recall London Ambulance Service)
- 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...

## Focus on outcomes over process

- Metrics easier for regular losses (risk)
- But rare catastrophes are hard (uncertainty)
- How reassuring are fatality statistics? E.g. Train Protection Systems, Tesla
- Accidents are random, but security exploits are not
- Product liability for death or injury is strict

257

At first it sounds like we should focus on outcomes rather than process...

Road deaths are coming down in the UK, from over 3,400 in 2000 to just over 1,700 in 2013 (<https://www.gov.uk/government/publications/annual-road-fatalities>).

Nevertheless, any death is something we wish to avoid and we could focus on these outcomes as a measure of success. Replacing robots sounds great if the number of fatalities by 50%; but perhaps people don't like the idea of 1000 deaths per year due to robots?

Bureaucracies prefer to focus on process. You see this with the public sector with lots of forms, approval procedures and so on; dealing with the residual risk here is difficult (what do you do about the 1000 deaths per year from robots)?

See Economics, Law and Ethics in Part IB for more on product liability.

## Focus on process over outcomes

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

258

...but there are some good reasons why we should focus on the process too.

## Getting incentives right is both important and hard to do

- The world offers hostile review, which we tackle in stages
- Some use hostile reviewers deliberately
- Standard contract of sale for software in Bangalore: seller must fix bugs for 90 days
- Businesses avoid risk (regulatory games)

259

First, some good examples. Delivery of software is now performed in stages, such as dogfood, alpha, beta, prod which helps us get feedback early from friendly and expert users before trialling on the unsuspecting. The use of hostile review is also a positive step and generally aligns incentives; examples include higher assurance levels of CC, manned spaceflight, nuclear weapons, etc. There is also some alignment of incentives when you sell software if you have to fix any bugs found within 90 days of sale: the buyer will do significant testing early (and perhaps reject if it is of really poor quality) and the seller knows that (assuming quality is reasonable) they are going to get paid and the amount of additional work is bounded. Nevertheless, businesses want to avoid risk wherever possible, and so in the absence incentives much safety and security work will be ignored. Therefore incentives alignment is really important.

## UK's Digital Service Standard: an example pulling it all together

- Understand user needs
- Do ongoing research
- Have a multidisciplinary team
- Use agile methods
- Iterate & improve frequently
- Evaluate tools and systems
- Understand security & privacy issues
- Make all new source code open
- Use open standards and common platforms
- Test the end-to-end service
- Make a plan for being offline
- Make sure users succeed first time
- Make the user experience consistent with GOV.UK
- Encourage everyone to use the digital service
- Collect performance data
- Identify performance indicators
- Report performance data on the Performance Platform
- Test with the minister

260

Here is the UK Government's current attempt at offering advice on how to build Government-funded digital services. Lots of good advice which mirrors much of what we have discussed in this lecture course, yet we have seen recent services still fail to live up to these ideals. Consider the issue of passport renewal we saw earlier as but one example. (Universal Credit is another.)

There are also standards, such as CyberEssentials, which also offer good advice on security.

Further details: <https://www.gov.uk/service-manual/service-standard>

## The future is challenging: how to we provide safety and security?

- Car manufactures must do pre-market testing
- Cars now contain lots of safety critical software
- Security requires us to patch bugs when they are found, yet this might invalidate safety case
- How will today's car get patches in 2039? 2049?
- What new tools and ideas do we need?

261

All code contains latent vulnerabilities. Therefore, particularly Internet-enabled devices require regular patches to keep them secure. Cars are no exception: Tesla delivers monthly patches to their cars. Yet safety regulation requires a safety case which in turn requires testing when significant changes are made. Does the manufacturer need to re-test cars after changes to software? Is there a sensible threshold where the change requires a retest? In the physical world, we can reason about whether a slightly different design of alloy wheel will affect the safety case, but this is not possible for software. Security is not composable. Similarly a single line change can have a dramatic impact on security (e.g. see goto fail).

How will we ensure that manufacturers continue to patch cars in 10, 20 or 30 years from now? We need new tools and ideas. This is your future career.



## Software engineering is about managing complexity

- Security and safety engineering are going in the same direction
- 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
- Remember: all software has latent vulnerabilities

262

Software engineering is about managing complexity, and this is hard. Yet it is our trade. Safety engineering requires software engineering (since everything has a CPU in it) and safety engineering requires security engineering since we are connecting these CPUs to the Internet, thereby exposing them to attacks from the other side of the world.

## Software and security engineering stretches well beyond the technical

- Complex systems are social-technical
- Institutions and people matter
- Confluence of safety and security may make maintenance the limiting factor

263

Remember that safety or security failure often occurs because people do not define the concept of the system broadly enough. The users matter, and secure and safe systems are usable systems.

Institutions and their people, culture and processes matter. About 30% of big commercial projects fail. This figure has been stable for years. Better tools simply let people climb higher up the complexity mountain before they fall off. It's an (admirable) human trait that we always aspire to do better, to step beyond what we have achieved before.

# The End

**Alastair R. Beresford**

arb33@cam.ac.uk

*With many thanks to Ross Anderson*

264

This is the last lecture in the series.