Aims

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

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
  - Understand development lifecycle
  - Understand bugs, vulnerabilities and hazards

Books

Software and Security

Alastair R. Beresford

With many thanks to Ross Anderson
What is Security Engineering?

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

The Design Hierarchy

- Policy
- Architecture, protocols, ...
- Hardware, crypto, access control, ...

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

Course Outline – Key topics

- Software as a service
- Testability
- Security issues
- Development lifecycle
- Software crisis
- Safety case
- Security policy
- Bugs
- User behaviour

Additional Reading

- F.P. Brooks, *The Mythical Man Month*
- J. Reason, *The Human Contribution*
- S. Maguire, *Writing Solid Code*
- H. Thimbleby, *Improving safety in medical devices* and systems
- O. Campion-Awawd el al., *The National Programme for IT in the NHS – A Case History*
- J.P. Brooks, *The Mythical Man Month*
Security vs Dependability

Dependability = Reliability + Security

- Malice is different from error
- Reliability and security are often strongly correlated
- Security vs Dependability

A system can be...

Electric bike should not propel bicycle when speed exceeds 15.5 mph

Common failure: policy drawn too narrowly

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

Principal:
Person: a subject or a legal person (firm)
Subject: a physical person

Subjects and Principals

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

Anonymity, integrity, authenticity

Anonymity: restrict access to metadata.

Integrity: an object has not been altered since the last authorized modification.

Authenticity: an object has integrity plus freshness.

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.

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. A trusted system controls the keys

Errors, failures, reliability, accidents

Confidentiality: an obligation to protect someone else’s secrets.

Privacy: control of your own secrets.

Security: mechanism to control which principals can access information.
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
- Risk is often combined with unit of exposure; e.g. a micromort
- Uncertainty is where the risk is not quantifiable

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 policy is a succinct statement of protection goals applied to a particular system

Traditional Government approach

- Safety is simple: freedom from accidents
- Uncertainty is where the risk is not quantifiable
- A critical system, process or component is one whose failure will lead to an accident
- A security target is a detailed statement of protection goals applied to a particular system

What's wrong with this?

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.

Basic idea since 1940: a clerk with 'Secret' clearance can read documents at 'Confidential' and 'Secret', but not at 'Top Secret'.

What's wrong with this?

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.
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

• Principals have clearance.
  • Official, secret, top secret, or high and low.

• Classify all documents and data with a level, such as official, secret, top secret, or high and low.

21 Bell-LaPadula formal model

• Bell-LaPadula (1973): simple security policy (no read down)
• Bell-LaPadula (1973): simple security policy (no read down)
• Bell-LaPadula (1973):
  * policy
  * policy

• With these two rules, one can prove that a system with a simple enforcement property of a testable system will remain in one high assurance MLS system.

• The pump simplifies the problem: replace the complex emergent property of the whole system with a simple testable component.

• Nevertheless, often harder than it looks.

22 The pump: cover channel bandwidth is a complex.

Moral: cover channel bandwidth is a complex.

Delayed and unusable signal?
How can you let message traffic pass from low to high? If you let message traffic pass from low to high...

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, just not delayed and unusable again?

23 High assurance MLS system

• The pump simplifies the problem: replace the complex emergent property of the whole system with a simple testable component.

• Nevertheless, often harder than it looks.

• Aim is to minimise the Trusted Computing Base.

• With these two rules, one can prove that a system with a simple enforcement property of a testable system will remain in one high assurance MLS system.

Covert channels cause havoc

• BLP lets malware move from Low to High, just not delayed and unusable 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, just not delayed and unusable again?
Multilateral Security

Stop lateral flow, examples:

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

Biba formal model for integrity

- Biba (1975)
- *-integrity policy (no write up)
- Simple integrity policy (no read down)

Biba’s formal model for integrity

Architecture matters

- Defence in depth:
  - Separate subnets
- Chrysler Jeep recall

- Trust all network nodes
- Lots of legacy protocols

Safety policies

- Failure modes and effects analysis
- Electrically redundant with safety at the highest level
- Medical devices with calibrate and operate modes
- Dual of the Bell-LaPadula model

Two basic ways to evolve:
- Mature industries adopt standards, but in more complex systems, new standards, but in more
- Competitive design environments, embedded in architectural assumptions, etc., often

- Industry have their own standards, cultures, etc.

Failure modes and effects analysis

- Electrically redundant with safety at the highest level
- Medical devices with calibrate and operate modes
- Dual of the Bell-LaPadula model

- *-integrity policy (no write up)
- Simple integrity policy (no read down)

- Biba (1975)

Safety policies

- Defence in depth:
  - Separate subnets
- Chrysler Jeep recall

- Trust all network nodes
- Lots of legacy protocols

Safety policies

- Failure modes and effects analysis
- Electrically redundant with safety at the highest level
- Medical devices with calibrate and operate modes
- Dual of the Bell-LaPadula model

- *-integrity policy (no write up)
- Simple integrity policy (no read down)

- Biba (1975)

Safety policies

- Defence in depth:
  - Separate subnets
- Chrysler Jeep recall

- Trust all network nodes
- Lots of legacy protocols

Safety policies

- Failure modes and effects analysis
- Electrically redundant with safety at the highest level
- Medical devices with calibrate and operate modes
- Dual of the Bell-LaPadula model

- *-integrity policy (no write up)
- Simple integrity policy (no read down)

- Biba (1975)
Failure modes and effects analysis

- 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

Example: nuclear bomb safety

- Developed by NASA
- Interactions
- Use secondary mechanisms to deal with
  - Redundancy?
  - Reduce risk by overdesign?
  - Figure out what to do about each failure

Fault tree analysis

- Work backwards from bad outcome we must avoid to
- Identify critical components

Bookkeeping, 8–4th millennium BCE
Bookkeeping, circa 1100 AD

- Double-entry bookkeeping: each entry in one ledger is matched by opposite entries in another ledger.
- 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 receivables account, debit the cash account. Customer subsequently pays, so credit the cash account, debit the receivables account.

Separation of duties in practice

- Serial:
  - Lecturer gets statement of money left on grant
  - Accounts checks delivery and tells Old Schools to pay
  - Stores receives goods, accounts gets invoice
  - Gets stores to sign order form and send to supplier
  - Lecturer gets Old Schools to register supplier
  - Lecturer gets money from EPSRC, charity, ...

- Parallel: authorization from two distinct subjects

Role-based access control (RBAC)

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Roles</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Exams</td>
<td>Future exam, Past exam</td>
</tr>
<tr>
<td>Bob</td>
<td>Exams</td>
<td>Future exam, Past exam</td>
</tr>
<tr>
<td>Charlie</td>
<td>Exams</td>
<td>Future exam, Past exam</td>
</tr>
<tr>
<td>Lecturer</td>
<td>Exams</td>
<td>Future exam, Past exam</td>
</tr>
<tr>
<td>Student</td>
<td>Past exam</td>
<td>Future exam</td>
</tr>
</tbody>
</table>

In the Genizah Collection

Double-entry bookkeeping found in the Genizah Collection.
Swiss Cheese Model

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

Summary of security and safety

- What are we trying to do?
- Security: threat model, security policy
- Safety: hazard analysis, safety standard
- Typical mechanisms: usability engineering, firewalls, protocols, access controls, etc.

Do not ignore user behaviour

- Many systems fail because users make mistakes
- Most car crashes are user error; yet we now build cars with crumple zones
- Banks routinely tell victims of fraud "our systems are secure so it must be your fault"
How can we protect against these attacks?

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

Many abuses of mechanism

Harm hierarchy:

- Targeted attacks
- Generic malware
- Bulk password compromise
- Abuse of mechanism

Volume of harm

Sophistication

Usable privacy is also hard

Traditional approaches – anonymisation and consent are really hard to deliver

Automated data collection (e.g. from sensors) makes the situation more difficult again

Problem gets harder as systems get larger
Medical device safety

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

Avionics are safer, as incentives are more concentrated
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
- Many errors arise from our highly adaptive mental processes

Cognitive factors

- Many errors arise from our highly adaptive mental processes
- Over time, the rules evolve, and are partly automatic
- Frequently encountered problems are dealt with using well-evolved, automatic processes
- We deal with novel problems in a conscious way

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

Phishing and social engineering

- What can we do about password reuse?
- How can we prevent many such exploitations?
- Reused passwords were used on those third-party sites
- Passwords were reused on other sites, from mail
- Method: SQL injection (seelater)
- Phished from a Russian forum
- Example: in June 2012, 6.5m LinkedIn passwords

John Podesta email compromise

- 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

Phishing and social engineering

- What can we do about password reuse?
- How can we prevent many such exploitations?
- Reused passwords were used on those third-party sites
- Passwords were reused on other sites, from mail
- Method: SQL injection (see later)
- Phished from a Russian forum
- Example: in June 2012, 6.5m LinkedIn passwords

Cognitive factors

- Many errors arise from our highly adaptive mental processes
- Over time, the rules evolve, and are partly automatic
- Frequently encountered problems are dealt with using well-evolved, automatic processes
- We deal with novel problems in a conscious way

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

Phishing and social engineering
Risk misperception

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

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

<table>
<thead>
<tr>
<th>Actual</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gain</td>
<td></td>
</tr>
<tr>
<td>Loss</td>
<td></td>
</tr>
</tbody>
</table>

Rational

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

Framing decisions about risk, or the Asian disease problem.

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.
- Reciprocal, built-in: Give a gift to increase your chance of receiving one.
- Distraction, so they act automatically.
- Appeal to the marks, dishonesty.
- Appeal to the marks, kindness.

### Fraud psychology

- Appeal to the mark’s kindness.
- Appeal to the mark’s dishonesty.
- Distract them so they act automatically.
- Arouse them so they act viscerally.

All the above plus:

Framing decisions about risk, or the Asian disease problem.
People only follow advice that 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 approaches. Users have different mental models. Explore how people only follow advice which confirms their own world view.

Affordances: Johnny can’t encrypt.

The power of default.

Most people don’t worry enough about computer security, and worry too much about terrorism. Therefore defaults may be contentious.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.

The power of default.
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

Where should the path be?

Differences between people

- Ability to perform certain tasks varies widely across subgroups of the population, including by age, gender, and education
- Yet all customers receive complex password rules and anti-phishing advice

More accidents with Volvos?

Volvo ÖV 4, April 1927

The compliance budget
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
- Procedure errors from action sequences (using the ATM)
- Retrieval failures ("its on the tip of my tongue")
- Misidentifying objects, signals
- Forgetting plans, intentions (strong habit intrusion)
- Slips and lapses
- Significant psychology research into errors

Training and practice reduce errors

<table>
<thead>
<tr>
<th>Time Short</th>
<th>Stress High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creative thinking</td>
<td>Unfamiliar complex operations</td>
</tr>
<tr>
<td>1.0</td>
<td>High stress complex task, much stress</td>
</tr>
<tr>
<td>1.0</td>
<td>Unfamiliar task dependent on situation, memory</td>
</tr>
<tr>
<td>1.0</td>
<td>Complex tasks, little time, some cues needed</td>
</tr>
<tr>
<td>1.0</td>
<td>Regularly performed simple tasks, low stress</td>
</tr>
<tr>
<td>1.0</td>
<td>Inexplicable errors, stress free, right cues</td>
</tr>
</tbody>
</table>

User studies are important

<table>
<thead>
<tr>
<th>Expected strength</th>
<th>Y &gt; G &gt; C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected resets</td>
<td>Y &gt; G &gt; C</td>
</tr>
</tbody>
</table>

Passwords are cheap, but...

- Will users enter passwords correctly?
- Will they remember them?
- Will they choose a strong password?
- Will the write them down?
- Will they be tricked into revealing passwords?

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
10% non-compliance

Knowledge-based mistakes, heuristics and biases

Can the user be tricked into revealing passwords?
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 attempts on card; 3 online)
  - iPhone PIN (timeouts)
  - Login attempts to web services (timeouts; care required)

Password recovery often involves basic information which doesn’t change:

Password recovery is a weak point.

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

Mitigate worst effects of a stolen password file:

• Use key stretching techniques such as PBKDF2:

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

• Establish breach reporting laws

• Externalise the problem with Oauth

Password recovery often involves basic information which doesn’t change:

• What was the name of your first pet?
• What was the name of your first school?

Password recovery is a weak point.

Password recovery often involves basic information 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.

...
Exteralities 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
  • Iterative guessing of card details with botnet on websites works

Externalities need consideration

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

Twitter: Find personal website, then Gmail, home address
Amazon: Call with name, address, credit card number and associate, new email address with the account
Amazon: Use web password reset to new email address; get credit card number (fake) to the account
Amazon: Call with name, address, credit card number and associate a new email
Amazon: Call with name, address, credit card number and associate, new email
Gmail: Account recovery, give “m••••m@me.com”
Twitter: Reset password sent to “m••••m@me.com”
Gmail: Reset password sent to Gmail
Twitter: Reset password sent to Twitter
Amazon: Use web password reset to new email address; get credit card number (fake) to the account
Amazon: Call with name, address, credit card number and associate a new email
Amazon: Call with name, address, credit card number and associate, new email
Gmail: Account recovery, give “m••••m@me.com”
Twitter: Reset password sent to “m••••m@me.com”
Gmail: Reset password sent to Gmail
Twitter: Reset password sent to Twitter

And all they wanted was his three letter Twitter handle!

Iterative guessing of card details with botnet on websites works

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

• 291 ask for PAN, expiry date and CVV2
• 291 ask for PAN, expiry date and CVV2
• 37 use PAN + postcode (numeric digits only for others)
• 291 ask for PAN, expiry date and CVV2
• Of Alexa top 500 websites, 6 use primary account number (PN) and expiry date
• Of Alexa top 500 websites, 6 use primary account number (PN) and expiry date

How Apple and Amazon
Security flaws led to my Twitter Hacking

Epic Hacking

Many firms train customers in unsafe behaviour from clicking on external links or redirecting the browser to third-party domains for payment

Everyone wants recovery questions too

Password sharing a conspicuous example; we have

One firm’s action has side-effects for others

Iterative guessing of card details with botnet on websites works

There is enough variation in requirements across websites that you can iteratively generate valid credentials.
Security protocols are another intellectual core of security engineering. They introduce an important abstraction, and illustrate adversarial thinking. They often implement policy directly, and they are much older than computers.

### Car unlocking protocols

- **Static**
  - E: K
  - T: N

- **Interactive**
  - E: T, {T, N}
  - T: {T, N}

<table>
<thead>
<tr>
<th>Non-interactive</th>
<th>Interactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>N: nonce; a sequence number, random number or timestamp</td>
<td>K: secret key shared between E and T</td>
</tr>
<tr>
<td>T: car key fob or transponder</td>
<td>E: engine unit</td>
</tr>
<tr>
<td>F: B: N</td>
<td>E: B: F</td>
</tr>
</tbody>
</table>

### Identity Friend or Foe (IFF)

- Basic idea: Fighter challenges bomber.
- Different possibilities:
  - What can go wrong?
  - B: E, F: {N}
  - F: E, B: N

### 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:

- Host orders wine; then it's served to guests.
- Host chooses wine; sommelier fetches it.
- Sommelier presents wine list to host.

Security protocols are another intellectual core of security engineering.
Person-in-the-middle attack...

- Basic idea: fighter challenges bomber

F®B: N
B®F: {N}

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

F®B: N
B®F: N
F®B: {N}
B®F: {N}

Card authentication protocol

Two-factor authentication (2FA)

- Users compute codes for access, authentication
- Allows EMV cards to be used in online banking
- Two-factor authentication protocol

Card authentication protocol

- Allows EMV cards to be used in online banking
- Users compute codes for access, authentication
- Two-factor authentication protocol

Card authentication protocol

- Allows EMV cards to be used in online banking
- Users compute codes for access, authentication
- Two-factor authentication protocol

Card authentication protocol

- Allows EMV cards to be used in online banking
- Users compute codes for access, authentication
- Two-factor authentication protocol
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 freshness?

Kerberos uses tickets to support communication between parties.

Replaces insides of the terminal with your own electronics.

Replace terminals with your own electronics.

How might you attack this?

How can they check the protocol's freshness?

Alice and Bob want to talk. They each share a key with Sam. How?
The relay attack: unstoppable but unrealistic – too hard to scale

PIN

$2000

$20

PIN attackers can be on opposite sides of the world

$89

Magstripe fraud is scalable

The no-PIN attack: simpler

Protocol required

Fixing the no-PIN attack: simpler

Either physically or wirelessly collect data

Install fake terminal and collect card data and PINs

Photo credit: Brian Krebs, krebsonsecurity.com

The no-PIN attack (2010)

Barclays introduced a fix July 2010; removed December 2010. Banks asked for student thesis to be taken down from web instead.

In practice has to be the issuer since incentives for terminal and acquirer are poor.

In theory might compare card data with terminal data at terminal, acquirer, or issuer.

Real problem: EMV spec now far too complex.

Evolutionarily fixed for UK transactions in 2016.

Real problem: EMV spec now far too complex.

Evolutionarily fixed for UK transactions in 2016.

Fixing the no-PIN attack: simpler

In theory might compare card data with terminal data at terminal, acquirer, or issuer.

Real problem: EMV spec now far too complex.

Evolutionarily fixed for UK transactions in 2016.

Fixing the no-PIN attack: simpler

In theory might compare card data with terminal data at terminal, acquirer, or issuer.

Real problem: EMV spec now far too complex.

Evolutionarily fixed for UK transactions in 2016.

Fixing the no-PIN attack: simpler

In theory might compare card data with terminal data at terminal, acquirer, or issuer.

Real problem: EMV spec now far too complex.

Evolutionarily fixed for UK transactions in 2016.

Fixing the no-PIN attack: simpler

In theory might compare card data with terminal data at terminal, acquirer, or issuer.

Real problem: EMV spec now far too complex.

Evolutionarily fixed for UK transactions in 2016.

Fixing the no-PIN attack: simpler

In theory might compare card data with terminal data at terminal, acquirer, or issuer.

Real problem: EMV spec now far too complex.

Evolutionarily fixed for UK transactions in 2016.
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 $d$.

- If I choose secret integer $a = 4$, then $A_{\text{Alice}} \overset{\text{g}}{\rightarrow} B: g^a \mod p = 5 \mod 23 = 10$

- If I choose secret integer $b = 3$, then $B_{\text{Bob}} \overset{\text{g}}{\rightarrow} A: g^b \mod p = 5^3 \mod 23 = 10$

Alice and Bob now agree the secret integer is $18$.

---

**Symmetric key cryptography** requires careful sharing of keys.

**Public key cryptography** allows two parties with no prior knowledge of each other to jointly establish a shared secret key over an insecure channel.

**Diffie-Hellman**

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

1. Alice chooses secret integer $a = 4$, then $A_{\text{Alice}} : g^a \mod p = 5^4 \mod 23 = 4$

2. Bob chooses secret integer $b = 3$, then $B_{\text{Bob}} : g^b \mod p = 5^3 \mod 23 = 10$

3. Alice computes $10^4 \mod 23 = 18$

4. Bob computes $4^3 \mod 23 = 18$

Alice and Bob now agree the secret integer is $18$.

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.

Asymmetric public-key crypto

Separate keys for encryption and decryption:

- Publish encryption key ("public key")
- Separate keys for encryption and decryption
- Digital signatures are the other way around: only you can sign but anyone can verify
- You can sign but anyone can verify
- Brute attack found 15 years later.

MITM attack found 15 years later.

Is this secure?

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

MITM attack found 15 years later.

Asymmetric public-key crypto

• Physical public key crypto with locks

97

• Anthony sends a message in a box to Brutus. Since the messenger is loyal to Caesar, Anthony puts a padlock on it.

98

• Asymmetric public-key crypto

99

• Separate keys for encryption and decryption

• Digital signatures are the other way around: only you can sign but anyone can verify

• You can sign but anyone can verify

• Brute attack found 15 years later.

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

Physical public key crypto with locks

Public-key Needham-Schroeder

Proposed in 1978.

Is this okay?

The idea is to use $N_A \oplus N_B$ as a shared key.

$K_A$ and $K_B$ are public keys for $A$ and $B$ respectively.

$N_A$ and $N_B$ are nonces generated by $A$ and $B$ respectively.

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

You can sign but anyone can verify.

Digital signatures are the other way around: only decoder the message and read it.

Brutus, who can now unlock it.

Anthony removes his padlock and sends it to Anthony.

Anthony adds his own padlock and sends it back to Anthony.

The messenger is loyal to Caesar, Anthony puts a padlock on it.

Anthony sends a message in a box to Brutus. Since

Anthony sends a message in a box to Brutus. Since
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 = $Sig(A, K, A, Timestamp, Length)$

**Transport Layer Security (TLS)**
- Uses public key cryptography and certificates to establish a secure channel between two machines
- Uses a large number of root certificate authorities
- Yet, the protocol is broken annually
- Protocol proven correct (Paulson, 1999)

**TLS security landscape is complex**

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

**SSL Labs** (SSL report: www.ssllabs.com/ssltest/)
- Used by some smartphone apps
  - Use certificate pinning inside an app
  - Based on Transport Layer Security (TLS) as used in HTTPS
  - Same signs certificate to bind Alice's key with her name
  - Use certificates with trusted certificate authority
  - Trust on first use; optionally verify later
  - IPSec SSL
  - Physically install binding on machines
The Mafia asks people to sign a random challenge as proof of age for porn sites!

Patriot missile failures in Gulf War I

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

Caused by arithmetic bug

- System measured time in 1/10 sec, truncated from 0.00100110011... to 0.00100110011...
- Accuracy upgraded as system upgraded from air-defense 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.
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 strings. For example, in Java:

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

Therefore, a malicious client could read:

The contents of server memory allows clients to read:

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

Heartbleed allows clients to read:

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

Therefore, a malicious client could read:

The attack left no trace of use in server logs.

Heartbleed bug (2014)
Notification and clean-up difficult

- Intel AMT Bug
  - ATG allows sysadmins remote access to a machine, even when turned off (but mains power on)
  - A sketch of the protocol for authentication
  - Provides full access to machine, independent of OS

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 Spanish or English
a recorded message in chosen language and read out
4. Phone then dialed the number and read out appended inmate name to the end:
   • An inmate from Clallam jail wishes to speak “Eve”

Intel AMT Bug
- AMT allows sysadmins remote access to a machine, even when turned off (but mains power on)
- A sketch of the protocol for authentication
- Provides full access to machine, independent of OS
- Even when turned off (but mains power on)
- ATG allows sysadmins remote access to a machine,

12th March 2012
Bug introduced (OpenSSL 1.0.1)

1st April 2014
Google secretly reports vuln

3rd April 2014
Codenomicon reports vuln

7th April 2014
Fix released

7th April 2014
Public announcement

9th May 2014
57% of websites still using old TLS certificates

20th May 2014
1.5% of 800,000 most popular websites still vulnerable to time of check to time of use failute (TOCTOU)

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

an inmate from Clallam jail wishes to speak “Eve”
Okay Google, what's a Whopper?

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 Abuse Act.
- 3 year suspended sentence
- 400 hr community service
- 6,000* machines infected
- Used a list of common weak passwords
- Exploited vulnerabilities in sendmail, fingerd, rsh

SQL Injection attack: failure to sanitize untrusted inputs

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

SQL Injection attack: failure to sanitize untrusted inputs

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

* Estimate

Burger King®

Okay Google, what's a Whopper?
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
  - API matching
    - Challenging for APIs accessible over the Internet

We cannot write code without latent vulnerabilities. Link OS versions to database of vulnerabilities:
- versions of LG handsets
- OS versions of LG handsets
- Match OS version information to OS and Build Number to put each handset into one group:
  - Insecure
  - Maybe secure
  - Secure

Why is software security important?

MIL-STD-188-125 MTI Tactical Environment Smart Environment MIL-STD-188-229

MIL-STD-188-229 MTI Tactical Environment Smart Environment
On average, 85% are vulnerable. 11% are secure.

The Software Crisis

- Some combine the above (i.e., some expensive scares, Y2K, Pentium).
- Some expensive scares (Y2K, Pentium).
- Some failures cost lives (Theerc25) or billions (Ariane 5, NPI, DWP, Amdahlbooks).
- Many large projects are late, over budget, or abandoned.
- Software still lags behind hardware's potential.

Project Background

- Public concern over service quality.
- Industrial relations poor; pressure to cut costs.
- Load test would fail.
- Consultant study said this might cost £1.9m and take 15 months, provided a packaged solution.
- South West Thames Regional Health Authority decided on fully automated system: responder would “email” ambulance.
- Some failures cost lives (Theerc25) or billions (Ariane 5, NPI, DWP, Amdahlbooks).
- Many aspects of the failure widely repeated since 1992.
- Widely cited example of project.
- Number estimated deaths ran as high as 20.
- Result left London without service for a day.
- London Ambulance Service disaster.

On average, 85% are vulnerable. 11% are secure.
**Original dispatch system worked**

- Worked on paper with regional control
- Resource mobilisation
- Call taking
- Resource identification
- Resource management
- Control
- Assistant
- Map
- Book
- Resource controller
- Incident form
- Resource reallocators
- Allocations
- Box
- Radio
- Operator
- Dispatcher
- Incident form
- Incident form''

**Many problems with original system**

- It took 3 minutes to dispatch an ambulance
- It required 200 staff (out of 2700 in total)
- Queues and bottlenecks, especially with the radio
- There were errors, especially in deduplication
- It took 3 minutes to dispatch an ambulance
- £700K cheaper than next lowest bidder
- Tender awarded to consolidation of Systems Options
- Tender awarded at tender, 19 proposed; most said possible by early 1992
- 35 firms looked at tender, 19 proposed; most said director hired

**Tender process was poor**

- £700K cheaper than next lowest bidder
- Tendered on 7th Feb 1992; completion due Jan 1992
- Tenders of a packaged solution forgotten; new IS
- Idea of £1.5m system stuck; idea of AVLS added

**Computer-aided dispatch system**

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

**Components**

- Controller
- Management
- Resource
- Mobilisation
- Resource
- Book
- Map
- Incident
- Call

**Tender was poor**

- Tendered on 7th Feb 1992; completion due Jan 1992
- 35 firms looked at tender, 19 proposed; most said director hired
- Idea of £1.5m system stuck; idea of AVLS added

**Computer-aided dispatch system worked**

- £700K cheaper than next lowest bidder
- Tender awarded to consolidation of Systems Options
- Tender awarded at tender, 19 proposed; most said possible by early 1992
- 35 firms looked at tender, 19 proposed; most said director hired

**Many problems with original system**

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

**Tender process was poor**

- £700K cheaper than next lowest bidder
- Tendered on 7th Feb 1992; completion due Jan 1992
- 35 firms looked at tender, 19 proposed; most said director hired
- Idea of £1.5m system stuck; idea of AVLS added

**Computer-aided dispatch system**

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

**Tender was poor**

- Tendered on 7th Feb 1992; completion due Jan 1992
- 35 firms looked at tender, 19 proposed; most said director hired
- Idea of £1.5m system stuck; idea of AVLS added

**Many problems with original system**

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

**Tender process was poor**

- £700K cheaper than next lowest bidder
- Tender awarded to consolidation of Systems Options
- Tender awarded at tender, 19 proposed; most said possible by early 1992
- 35 firms looked at tender, 19 proposed; most said director hired

**Computer-aided dispatch system worked**

- £700K cheaper than next lowest bidder
- Tender awarded to consolidation of Systems Options
- Tender awarded at tender, 19 proposed; most said possible by early 1992
- 35 firms looked at tender, 19 proposed; most said director hired
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-booking, gazetteer, docket printing

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 26th Oct 1992
• Independent review had called for volume testing, implementation strategy, change control, …all ignored
• CEO: “No evidence to suggest that the full system is fit for purpose, when commissioned, will not prove reliable”

Independent review had called for volume testing, implementation strategy, change control, …all ignored

On 26 Oct, room was reconfigured to use terminals, not paper. There was no backup...

CEO: “No evidence to suggest that the full system is fit for purpose, when commissioned, will not prove reliable”

Management decided to go live on 26th Oct 1992

Independent review had called for volume testing, implementation strategy, change control, …all ignored

Phase one: design work ‘done’ in July and contract signed in August

Phase two: full automation

System slowdown and congestion leading to collapse

Circle of disaster on 26/7th October

• System progressively lost track of vehicles
• Exception messages scrolled off screen and were lost

133

Taking exception, docket printing automation by January deadline – front end for call-booking, gazetteer, docket printing

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

No full-time LAS users providing domain knowledge

A lack of methodology

A 6-month timescale for an 18-month project

Minutes of a progress meeting in June recorded: July and contract signed in August
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.

Operational mistakes

• Specification mistakes
  • Specification was inflexible but incomplete: it was drawn up without adequate consultation with staff.
  • Specification mistakes were directly related to the collapse.
  • Specification was disregarded by the paramedics.
  • Ambulance service staff skills and experience were insufficient.
  • Poor interface for ambulance crews.
  • Slow response times.
  • System went live with known serious faults.

Project management mistakes

• Chief executive resigns
  • Poor control room interface.
  • Poor interface for ambulance crews.
  • Poor data communications, with effects not foreseen.
  • Procurers insufficiently qualified and experienced.

• Specification mistakes
  • Specification was inflexible but incomplete: it was drawn up without adequate consultation with staff.
  • Specification mistakes were directly related to the collapse.
  • Specification was disregarded by the paramedics.
  • Ambulance service staff skills and experience were insufficient.
  • Poor interface for ambulance crews.
  • Slow response times.
  • System went live with known serious faults.

• Project management mistakes
  • Confusion over who was managing it all.
  • Poor change control, no independent QA, suppliers misled on progress.
  • Confusion over who was managing it all.
  • Inadequate software development tools.
  • Poor change control, no independent QA, suppliers misled on progress.
  • Confusion over who was managing it all.

• Operational mistakes
  • System went live with known serious faults.
  • Specification mistakes were directly related to the collapse.
  • Specification was disregarded by the paramedics.
  • Ambulance service staff skills and experience were insufficient.
  • Poor interface for ambulance crews.
  • Slow response times.
  • System went live with known serious faults.

• Specification mistakes
  • Specification was inflexible but incomplete: it was drawn up without adequate consultation with staff.
  • Specification mistakes were directly related to the collapse.
  • Specification was disregarded by the paramedics.
  • Ambulance service staff skills and experience were insufficient.
  • Poor interface for ambulance crews.
  • Slow response times.
  • System went live with known serious faults.

• Project management mistakes
  • Confusion over who was managing it all.
  • Poor change control, no independent QA, suppliers misled on progress.
  • Confusion over who was managing it all.
  • Inadequate software development tools.
  • Poor change control, no independent QA, suppliers misled on progress.
  • Confusion over who was managing it all.

• Operational mistakes
  • System went live with known serious faults.
  • Specification mistakes were directly related to the collapse.
  • Specification was disregarded by the paramedics.
  • Ambulance service staff skills and experience were insufficient.
  • Poor interface for ambulance crews.
  • Slow response times.
  • System went live with known serious faults.

• Specification mistakes
  • Specification was inflexible but incomplete: it was drawn up without adequate consultation with staff.
  • Specification mistakes were directly related to the collapse.
  • Specification was disregarded by the paramedics.
  • Ambulance service staff skills and experience were insufficient.
  • Poor interface for ambulance crews.
  • Slow response times.
  • System went live with known serious faults.

• Project management mistakes
  • Confusion over who was managing it all.
  • Poor change control, no independent QA, suppliers misled on progress.
  • Confusion over who was managing it all.
  • Inadequate software development tools.
  • Poor change control, no independent QA, suppliers misled on progress.
  • Confusion over who was managing it all.

• Operational mistakes
  • System went live with known serious faults.
  • Specification mistakes were directly related to the collapse.
  • Specification was disregarded by the paramedics.
  • Ambulance service staff skills and experience were insufficient.
  • Poor interface for ambulance crews.
  • Slow response times.
  • System went live with known serious faults.

• Specification mistakes
  • Specification was inflexible but incomplete: it was drawn up without adequate consultation with staff.
  • Specification mistakes were directly related to the collapse.
  • Specification was disregarded by the paramedics.
  • Ambulance service staff skills and experience were insufficient.
  • Poor interface for ambulance crews.
  • Slow response times.
  • System went live with known serious faults.

• Project management mistakes
  • Confusion over who was managing it all.
  • Poor change control, no independent QA, suppliers misled on progress.
  • Confusion over who was managing it all.
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’
2013 (then: Sep 2014 ‘...’)
- Coalition Government: wanted deployment by 2015
- Coalition Government: wanted deployment by 2015
- 2010: Experts said we just can’t change 47m meters
- 2009: EU Electricity Directive for 80% by 2020
- 2009: EU Electricity Directive for 80% by 2020
- 2009: EU Electricity Directive for 80% by 2020
- 2002: Labour IT centralised project to save the planet and help fix supply crunch in 2017

145

146

Smart meters: more centralisation

Idea: fix poverty trap

Universal Credit: fix poverty trap

• Wrong, changing, or contested requirements
• Most failures of really large systems are due to
  reactions to new functionality
  technical interactions mean we can’t predict
  the system, isn’t just the code; complex socio-
  components flow at O(n) or even O(2^n)
  as programs get bigger, interactions between
  bugs arise at micro level in challenging components

Managing complexity at many levels

Software engineering is about

Descented into chaos; see NAO report

Depended on firms

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

There is often little incentive to get a job
Idea: hundreds of welfare benefits which means

Spec still quick, tech getting obsolete, despair...

Coalition Government: NPfIT ‘abolished’
Most systems years late or never worked
Five LSPs plus national contracts: £12bn
The February 2002 Blair meeting
Earlier failed attempt in the 1990s
Change working practices
Like LAS, an attempt to centralise power and

NHS National Programme for IT
On contriving machinery

Charles Babbage

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

Dun, Barlow & Co, 1876

Bank of England, 1870

Project failure, circa 1500 BCE
Sears, Roebuck and Company, 1906

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

First National Bank of Chicago, 1940

• Documentation and testing
  by using disciplines such as project planning,
  control, and specializations
• The term software engineering coined in 1966
• People started asking why project overruns and failures were so much more common than in mechanical engineering, shipbuilding, etc.
• The hope was that we could bring things under control
• Large, powerful mainframes made complex systems possible

The software crisis, 1960s

• Joy of solving puzzles and building things from interlocking parts
• Satisfaction of making stuff that's useful to others
• Complete flexibility – you can base the output on the inputs in any way you can imagine
• Pleasure of working with a tractable medium, pure thought stuff
• Stimulation of a non-repeating task with continuous learning
• Joy of solving puzzles and building things from interlocking parts
• Satisfaction of making stuff that’s useful to others

Those things which make writing software fun also make it complex.
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 visualize 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.

Software economics can be nasty!

Measuring cost of code is hard.

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

AT&T measures
- 2.2 KLOC per developer-year (switch)
- 0.6 KLOC per developer-year (compiler)

IBM measures (1960s)
- 10 KLOC per developer-year (app)
- 5 KLOC per developer-year (compiler)
- 1.5 KLOC per developer-year (operating system)

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

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)
KLOC is a poor measure

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

Example project with 3 developers and 9 months,

```
for (int i = 0; i < 4; i++) {
    System.out.println("Hello, world");
}
```
... • UML • Yourdon • Jackson • SSADM
Lots of methodologies based on this idea:
• Define clear APIs between them
• Divide a complex system into small components
• Only practical way forward is modularization

Take a structured, modular approach

Some projects still fail:
• Hardly any projects succeed at ¾.
• With less time, costs rise sharply.
• With more time, costs rise slowly.

\[ T = 2.5\sqrt{D} \]

\( T \) is time to first shipment and \( D \) is developer-months (Boehm, 1982).

Time to first shipment is cube root of developer-months.

The Software Tar Pit

Take a structured, modular approach

• Only practical way forward is modularization

• Divide a complex system into small components

• Define clear APIs between them

• Lots of methodologies based on this idea:
  • SSADM
  • Jackson
  • Yourdon
  • UML

We try to catch up:

By 6 developers in 1 – interaction costs may be O(n^2).

But: work of 3 developers in 2 months can’t be done by 6 developers in the next month.

Train 3 more developers in the first month, then use all 6 developers in the next month.

Mythical Man-Month: “adding manpower to a late project makes it later”
The Waterfall Model (1970)

- Requirements
- Specification
- Implementation & Unit Testing
- Integration & System Test
- Operations & Maintenance

1. validate
2. verify

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 is viable it’s usually the best approach
- The really critical factor is whether you can define the requirements in detail in advance. Sometimes you can (Y2K bugfix); sometimes you can’t (HCI)

Waterfall fails where iteration is required, such as:

- Requirements not yet understood by developers
- The environment (legal, competitive) is changing
- The technology is changing
- Not yet understood by the customer
- Requirements not yet understood by developers
- Requirements needed, but not yet defined
- …
Iterative development

- Develop outline spec
- Build system
- Use system
- Deliver system

OK?

No

Problem: this algorithm might not terminate!

Evolutionary model

- By the 1990s some codebases had become so big that, don’t yet understand!
- Each iteration is done top-down
- Decide in advance on a fixed number of iterations
- Driven by risk management (i.e., prototype bits you don’t yet understand)

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)

Spiral model

1. Determine objectives
2. Identify and resolve risks
3. Development and test
4. Plan next iteration

Evolutionary model

- By the 1990s some codebases had become so big that, don’t yet understand!
- Each iteration is done top-down
- Decide in advance on a fixed number of iterations
- Driven by risk management (i.e., prototype bits you don’t yet understand)
The Integrated Development Environment (IDE) includes:

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

Content-heavy apps benefit from

<table>
<thead>
<tr>
<th>Prod</th>
<th>Staging</th>
<th>Stable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>Latest</td>
<td>Stable</td>
</tr>
</tbody>
</table>

Tacoma Narrows, 7th Nov 1940

Content: https://www.youtube.com/watch?v=j-zczJSxnw

Critical computer systems have much in common with mechanical systems (bridges, brakes, locks).

Certified systems must study how things fail:

- Critical software avoids certain classes of failures with high assurance.
- Safety-critical systems: failure could cause death, injury or property damage.
- Security-critical systems: failure could allow leakage.
- Real-time systems: software must accomplish certain tasks on time.
- Dev/Test/Prod deployment (Webserver)
- Continuous integration (Jenkins)
- Automated build system (Maven)
- Code review (Git)
- Code and documentation under version control (Git)

The Integrated Development Environment (IDE) includes...

Your host types:

- Prod
- Staging
- Stable
- Latest
- Content
Hazard elimination

• Which motor reversing circuit is the safe one above?
• Some architecture and tool choices can eliminate whole classes of software hazards, e.g. using a garbage collector to eliminate memory leaks.
• But usually hazards involve more than just software hazards. For example, designers don’t consider human factors such as usability and training.

Multi-factor failure

• Testing is often really hard. Techniques such as bug detection in software are not getting the scope right.
• Criticality of timing makes many simple verification techniques inadequate.
• The same goes for security, and real-time systems used in monitoring or control.
• Exception handling is often tricky.

Emergent properties

• A very common error is not getting the scope right.
• In general, safety is a system property and has to be dealt with holistically.
• The backup inertial navigation set core dumped, which was interpreted as flight data. An error in float-to-integer conversion was involved, faster than Ariane 4, causing Airplane 5’s boost separation.
• Full nozzle deflection → 20° angle of attack. Which was interpreted as flight data.
• Hazards such as usability and training don’t consider human factors such as usability and training.

Ariane 5, 4th June 1996

• Some architecture and tool choices can eliminate whole classes of software hazards. E.g. using a garbage collector to eliminate memory leaks.
• Which motor reversing circuit is the safe one above?
Therac-25: radiotherapy machine

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

Therac-25 caused injuries

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

Therac had two operating modes:

- Don't fire focused beam at humans

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

- Error in programming
- Example of fatal accidents
dead in six

Three people died in six accidents

Marietta, GA, June 1985: woman's shoulder burnt

Ontario, July 1985: woman's hip bumed. acet knob
Settled out of court. FDA not told

Yakima, WA, Dec 1985: woman's hip bumed. could have been software error but could not reproduce fault

Ontario, July 1985: woman's hip bumed. acet knob
Settled out of court. FDA not told

Therac-25: radiotherapy machine
Therac-25: Therapeutic X-ray generator, manufactured by Nordion, was marketed as a safer alternative to linear accelerators.

- **Therac-25: East Texas deaths due to editing beam type too quickly**
  - Three people were killed:
    - 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.

- **Therac-25: Ontario accident**
  - Yakima, WA, January 1987: man burned on the chest and died due to different bug now thought to have caused Ontario accident.

- **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.

- **Software safety myths**
  - Computers are cheaper than analogue devices.
  - Software is easy to change.
  - Computers are cheaper than analogue devices.
  - Software safety myths: cheaper, easy to change, reliable.

- **Increasing reliability increases safety**
  - Shuttle software had 16 potentially fatal bugs found since 1988.
  - Shuttles are more reliable than analogue devices.

- **Manufacturer left the medical equipment business**
  - East Texas Cancer Centre, March 1986: man burned in neck and died five months later of complications.

- **East Texas Cancer Centre, March 1986: 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**
  - Hospital physicist managed to reproduce flaw: if parameters changed too quickly from X-ray to electron beam, the safety interlock failed.

- **Ontario accident**
  - Same place, three weeks later: another man burned on the face and died three months later of complications.

- **Therac-25: Easy Texas deaths due to editing beam type too quickly**
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
  - Counter examples: Ariane 5, Patriot and Therac-25

- Testing can make software arbitrarily reliable
  - Not even for 100-line programs

- Redundant hardware does not solve software engineering issues
  - Hardware can still fail; backup inertial navigation
  - Redundant hardware creates additional software engineering issues
  - Redundant software (multi-version programming)

- Redundant hardware for non-stop processing

Stratus computer: redundant hardware for non-stop processing

Redundancy in the Boeing 737

Also an opportunity for new types of failure

- Automation can reduce risk
  - For MTBF of 10 hours you must test >10^9 hours

- Testbeds can make software arbitrarily reliable
  - Not even for 100-line programs
  - Formal verification can remove all errors

- Reuse increases safety
  - Methods, testing and automation

Software safety myths: reuse, formal
Panama crash with 47 fatalities
6th June 1992

Kegworth crash, 47 fatalities
8th January 1989

Understanding and prioritising hazards

Example from the motor industry:

Air traffic control design
Initially blamed writing.
Approach: no power.
Opened throttle on final.
East rudder.
Emergency landing at
engines.
Crew shutdown wrong.
Fan blade broke.

Still complex social-technical system that exhibits failure

Excellent regulation and reporting

much better than with medical devices

The whole system has good incentives for learning

Crew capabilities are well known

Interfaces are stable

pilot training, air traffic control...

stable components: aircraft design, avionics design,

It's a mature evolved system

Aviation is actually an easy case

Understand and prioritise hazards

- Normal safety: affects customer satisfaction but not
- More: normal response limits and outcome
- Worst severe: usually controllable, outcome at
- Inconvenient only under favourable circumstances.
- Difficult to control: very severe outcomes.

1. Uncontrollable: outcomes can be extremely

2. Deleterious: usually controllable, outcome at

3. Disturbing: normal response limits and outcome

4. Disturbing: normal response limits and outcome

5. Nuisance: affects customer satisfaction but not

Understand and prioritise hazards

Example from the motor industry:

Air traffic control design
Initially blamed writing.
Approach: no power.
Opened throttle on final.
East rudder.
Emergency landing at
engines.
Crew shutdown wrong.
Fan blade break.

8th January 1989

Kegworth crash, 47 fatalities

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

6th June 1992

Panama crash with 47 fatalities
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, ...)
- Plan for and obtain certification
- Design a management plan
- Develop test plans, procedures, training
- Identify critical components

Most mistakes occur outside the technical phases.

Software engineering tools help manage complexity.

Challenging parts are often:
- Requirements engineering
- Certification
- Operations
- Maintenance

Software complexity?

...?

Heavily object: raise with lever

Tough object: cut with axe

task exceeds our native capacity. So

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

High-level languages remove incidental complexity:
- 2 KLOC per year goes much farther than assembler.
- Appropriate abstraction: data structures, functions, etc.
- Code easier to understand and maintain.
- Structure finds many errors at compile time.
- Branches rather than bits, registers.
- Code may be portable; or at least, the machine-independent.

High-level languages support structure and componentisation:
- Formal methods find bugs, but it is fallible.
- Structure and componentisation.

Huge performance gains possible, now realised:
- Structure reduces many errors at compile time.
- Appropriate abstraction: data structures, functions, etc.
- Code easier to understand and maintain.
- 2 KLOC per year goes much farther than assembler.
- Code more portable; or at least, the machine-independent.

Formal methods finds many errors at compile time.
- Structure reduces many errors at compile time.
- Appropriate abstraction: data structures, functions, etc.
- Code easier to understand and maintain.
- 2 KLOC per year goes much farther than assembler.
- Code more portable; or at least, the machine-independent.

Don’t forget: this is to manage intrinsic complexity.

Formal methods finds bugs, but it is fallible.

Formal methods finds bugs, but it is fallible.

History:
- Goto statement considered harmful (Dijkstra, 1968)
- Structured programming with Pascal (Wirth, 1971)
- “Goto statement considered harmful” (Dijkstra, 1968)

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)
- HOL, Gordon (1988)
- Turing talked about programs correct.

Huge performance gains possible, now realised:
- Code may be portable; or at least, the machine-independent.
- Structure reduces many errors at compile time.
- Appropriate abstraction: data structures, functions, etc.
- Code easier to understand and maintain.
- 2 KLOC per year goes much farther than assembler.
- Code more portable; or at least, the machine-independent.

Formal methods finds bugs, but it is fallible.

Formal methods finds bugs, but it is fallible.

History:
- Goto statement considered harmful (Dijkstra, 1968)
- Structured programming with Pascal (Wirth, 1971)
- “Goto statement considered harmful” (Dijkstra, 1968)

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)
- HOL, Gordon (1988)
- Turing talked about programs correct.

Huge performance gains possible, now realised:
- Code may be portable; or at least, the machine-independent.
- Structure reduces many errors at compile time.
- Appropriate abstraction: data structures, functions, etc.
- Code easier to understand and maintain.
- 2 KLOC per year goes much farther than assembler.
- Code more portable; or at least, the machine-independent.

Formal methods finds bugs, but it is fallible.

Formal methods finds bugs, but it is fallible.

History:
- Goto statement considered harmful (Dijkstra, 1968)
- Structured programming with Pascal (Wirth, 1971)
- “Goto statement considered harmful” (Dijkstra, 1968)

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)
- HOL, Gordon (1988)
- Turing talked about programs correct.
Static analysis tools are a useful result of formal methods

Chief programmers (IBM, 1970s)

Chief programmers (IBM, 1970s)

Aim: avoid loss of great programmers to management

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)

Egoless programming: minimize personal factors (Weinberg, 1971)
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 starting point for use of a new process
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

Extreme Programming (Beck, 1999)

• Iterative development with short cycles
• Automated build and test suites
• Frequent points to integrate new requirements
• Avoid programming a feature until needed
• Solve the worst problem, repeat
• Programming in pairs, one keyboard and screen
• Frequent release, daily meetings, working software as a measure of progress, regular reflection, etc.

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 a measure of progress, regular reflection, etc.

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

Curtis (1988) found causes of failure were:

The specification still matters
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

The specification can kill you

• Spec-driven development of large systems leads to communication problems since \( \frac{N(N-1)}{2} \) channels and \( 2^N \) subgroups

   • Big firms have hierarchies; info flows via ‘least common manager’

   • Big firms have committees; inter-committee communnication problems since \( N \) people means \( N \) people means communication problems

Project management: plan, motivate, control

A manager’s job is to:

motivate, control

Control

Motivate

Plan

...
Many other factors:
- Choice - everyone has a say in what they do
- Content - everyone's task clearly matters
- Collaboration - everyone has a specific task

Dan Rothwell's three Cs of motivation:
- Collaboration no good: people won't think they will win
- Competition no good: people won't think they can slack in groups (free rider, social loafing)

Motivating people in groups:
- People can slack in groups (free rider, social loafing)
- Many other factors:
  - Choice - everyone has a say in what they do
  - Content - everyone's task clearly matters
  - Collaboration - everyone has a specific task

Gantt charts:
- Tasks and milestones can be hard to visualise dependencies in large charts

PERT charts:
- Show critical paths

Project management triangle:
- Scope
- Time
- Cost

Which paths are critical?
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!

Regression Tests:
- Check that new versions of the software give the 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
- Customers more upset by failure of a familiar feature than at a new feature which does not work
- Software gives same answers as old versions
- Regression Tests: check that new versions of the software give the same answers as old versions

Design for testability, use CI and automate regression testing

A MTBF of x requires testing for x

Reliability growth models help us assess MTBF, number of bugs remaining, economics of further testing, etc.

Failure rate due to one bug can be modelled as:

\[ f(t) = \frac{e^{-\frac{t}{\lambda}}}{\lambda} \]

So for 10^9 hours MTBF, must test >10^9 hours

\[ \int_0^x f(t) \, dt = \sum f_i \]

Cost per bug rises dramatically down this list!

Subsequent litigation
- Beta test / field trial
- System test after daily build
- Module test after coding
- Design validation, UX prototyping

Happens at many levels:

Testing: half the effort (and cost)
Think about diversity & inclusion

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

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

Keeping all documents in sync is hard:

— to whom do you have to prove what?
— arrange hostile reviews, while military/NASA, DoE
— incentives matter: commercial developers look for friendly certifiers, while military, NASA, DoE
— random testing; fuzzing — new good practice
— environment conditions (e.g. Failure)
— many failures result from unforeseen input or conditions when system is in use

Release management: from development code to production

Main focus is on stability:

• Critical decision: patch old version or force
• Add copy protection, rights management

Release management:

• Social consensus: style, comments, formatting
• Operational plans and controls, department
• High tech: integrated development environment

Possible partial solutions:

• From specifications, test plans, code?
• Effective documents (requirements, hazard, budgets, PERT charts, staff schedule)?
• How will you deal with management documents?

Keeping all documents in sync is hard:

— to whom do you have to prove what?
— arrange hostile reviews, while military/NASA, DoE
— incentives matter: commercial developers look for friendly certifiers, while military/NASA, DoE
— random testing; fuzzing — new good practice
— environment conditions (e.g. Failure)
— many failures result from unforeseen input or conditions when system is in use

Tests should exercise the
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

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 a fixed delay: coordinated or responsible disclosure
4. No disclosure, but then vendor can't fix

Shared infrastructure provides benefits & implies responsibilities

- We share a lot of code through open source
- Vendors use bug bounty programs to discover I.

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?

- Someone must assess risk and be responsible for:
  - Login:
  - Recovery
  - Manage backup
  - Live running
- Objective: manage testing and deployment
- Change control and config are critical; often poor

Important and can be overlooked
Focus on outcomes over process

- Business avoids risk (regulatory games)
- Sellers must fix bugs for 90 days
- Standard contract of sale for software in Bangalore
- Some use hostile reviewers deliberately
- The world offers hostile review, which we tackle in important and hard to do

Getting incentives right is both

Focus on process over outcomes

- But leaves a gap of residual risk and uncertainty
- Public sector is really keen on compliance
- Blame avoidance is what bureaucracies do
- Safety raising maintenance
- Security development lifecycle is establishments
- Necessary to adapt as environment changes

Getting incentives right is both

Focus on process over outcomes

- hire the big consultants...
- So firms do the checks and use fashionable tools,
- the standards of the industry
- In law negligence breeding negligence judged by
- Risk reduction becomes compliances
- Breaches act as machines for avoiding blame
- Project utilitarian (recall London Ambulance Service)
- Employees often optimize their own utility, not

Beware of agency issues
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 all new source code open
- Understand security & privacy
- Issue public performance indicators
- Test with the minister
- Car manufacturers must do pre-market testing
- What new tools and ideas do we need?
- How will today’s cars get patches in 2039? 2049?
- Top-down approaches can sometimes help, but the intrinsic complexity remains the same direction
- We can cut incidental complexity using tools, but security and safety engineering are going in the same direction

The future is challenging: how to provide safety and security?

- Security and safety are often emergent properties
- Top-down approaches can sometimes help, but really large systems evolve
- Confluence of safety and security may make maintenance the limiting factor
- Confluence of safety and security may make maintenance the limiting factor
- Complex systems are social-technical
- Complex systems are social-technical

Software and security engineering stretches well beyond the technical

Software engineering is about managing complexity

- Remember: all software has latent vulnerabilities
- Security and safety are often emergent properties
- Top-down approaches can sometimes help, but really large systems evolve

The future is challenging: how to provide safety and security?
What is SaaS?

SaaS (Software as a Service) refers to software that is hosted centrally and licensed to customers on a subscription basis. Users access SaaS software via thin clients, often web browsers.
Process

Software Engineering

Impact of SaaS on the

In reality it's a spectrum

Everything's a service


don't hallucinate.

Traditional software distribution

Build software

Release versioned binaries

...
Impact on the 'software company'

Deploy

Manage/
Upgrade

Build
software

Before

Build
software

Release

versioned
binaries

After

Now have to worry about building software and running it

Have to continue evolving/upgrading the software with zero downtime

But the good news:

● 'Software release' no longer an all-or-nothing discrete event
  ○ Provides new ways to manage quality and reduction risk
  ○ Software release no longer as risk associated with discrete event

Continuous Integration (CI):

Continuous Deployment Without Downtime

Managing Continuous
Deployment

Short integration cycles lead to greater throughput

Continuous Integration (CI):

Short integration cycles lead to greater throughput

Continuous Deployment Without Downtime

Managing Continuous
Deployment

Short integration cycles lead to greater throughput

Continuous Integration (CI):

Short integration cycles lead to greater throughput

Continuous Deployment Without Downtime

Managing Continuous
Deployment

Short integration cycles lead to greater throughput

Continuous Integration (CI):
Continuous Deployment (CD): bring 'deploy' into the 'short cycle'

- Built artifacts
- Continuous Integration
- Automated deploy to 'test server' environment
- Run automated acceptance tests
- Automated deploy to production (Ive server's)
- Immediate alerting/feedback on fail condition

Production monitoring / alerting provides immediate feedback but now failures are visible to customers...

- How to do this while reducing risk?
- How to do this while 'always on'?

Rolling deploy

1) Deploy 'canary' (limit exposure/risk)

- Note: these resources are usually running in a cloud platform. So virtual machines, load balancers, storage, network etc. can all be provisioned and configured through the cloud platform's APIs.

- 25% of traffic each

- 24.75% of traffic each to x y instances

- 1% of traffic to x(y+1)
Rolling deploy: 2) Automated monitoring of error rates - OK?

Load Balancer

24.75% of traffic each to x.y instances

1% of traffic to x.(y+1)

Centralised logging

Automated alerts

Rolling deploy: 3) Move traffic from old instance to new etc.

Centralised logging

Automated alerts

Rolling deploy: 4) Upgrade 0% instance

Centralised logging

Automated alerts

Rolling deploy: 5) Move traffic from old instance to new etc.

Centralised logging

Automated alerts
Rolling deploy: Repeat {move traffic old->new; upgrade old}

Load Balancer

Centralised logging

Automated alerts

25%

25%

25%

0%

\(x \cdot (y+1)\)

\(x \cdot y\)

\(x \cdot (y+1)\)

\(x \cdot (y+1)\)

\(x \cdot (y+1)\)

Rolling deploy with service dependencies

CONSTRAINTS:

- \(a \cdot b \) supports \(x \cdot y\)
- \(a \cdot (b+1) \) supports \(x \cdot (y+1)\)

1. Deploy \(a \cdot (b+1)\)

Challenge:

How do we upgrade the dependent service while keeping everything running?

And how do we handle the dependent service when making a ‘breaking change’ to its API?

1. Deploy \(a \cdot (b+1)\)

Rolling deploy: Repeat {move traffic old->new; upgrade old}
1. Deploy a.(b+1)
2. Start rolling out x.(y+1)
3. Finish deploy of x.(y+1)
4. Deploy (a+1).0

We say: a.(b+1)'s API is backwards compatible with a.b's API
(a+1).0's API can introduce a breaking change
On Automation: Infrastructure-as-Code

Problem:
- Manual deployments are time-consuming and error-prone. Subtle environmental differences cause bugs.

Solution:
- Write code to automate deployments, using Cloud APIs etc.
- Put deployment code under version control, just like all other code
- Have development teams write:
  - Application code
  - Code to test the application
  - Code to deploy the application and its associated cloud infrastructure
  - Code to monitor the application and generate alerts

Frameworks like Terraform and CloudFormation help with this.

Other SaaS tools for managing quality:
- Rolling deploy + alerting is a very effective way of managing quality vs. big bang release.
- Insight: as long as we manage user impact, real users become an invaluable part of the QA process.

NB: QA ≠ Quality
and experiments

Behavioral analytics

Traffic mirroring

Synthetic monitoring

Complements regular alerting:
- Deeper testing of end-to-end behavior
- Can test parts of the site that are not actively being used
- Can test important corner case paths that are not sufficiently exercised by real users to show up in aggregate monitoring

Mirroring

New service working?

New SaaS service under test

SaaS service

Alerting based on real usage

Automated alerts

Automated playback of common user actions

Service responses

Alert if response differs from expected result or performance

Synthetic monitoring

Behavioral analytics and experiments
What can we learn from the event logs?

- **User/growth metrics:**
  - Monthly Active Unique Users (MAU); Daily Active Unique Users (DAU)

- **Engagement:**
  - Time spent using the service

- **Feature usage/growth/engagement metrics:**
  - X% of users tried feature F at least once in the last month
  - Y% of users used feature F2 for at least 5 minutes last week
  - Feature F3 usage growing at Z% year-on-year

- **Insights based on user segmentation:**
  - Users who signed up in January 2018 exhibit an average 2% monthly churn
  - Female users aged between 20-25 are X% more likely to use feature F at least once

What else can we learn from the event logs?

- **Correlations:**
  - Usage of feature F2 is correlated with usage of feature F1
  - Daily time spent on the platform is correlated with the number of days since sign-up

- **But NOT cause and effect... At least not without an experiment framework.**

How can we move from correlations to cause/effect?

- **Run controlled experiments:**
  - Determine hypothesis to test
  - Determine level of exposure, E, (% of users that will go into experiment group)
  - Bucket users into either experiment group (E%) or control group (100-E%)
  - Release a change to the experiment group only
  - Measure relevant metric(s) in both control group and experiment group and determine whether the observed difference is statistically significant

By measuring difference between control and experiment groups we can have some confidence that the only meaningful difference is our change under test.

- Often pick low and ramp up (e.g.: 1% of users, 10%, 25%, 50%)

- Feature usage grows by at least Z% year-on-year

- Feature usage is correlated with usage of feature F1

By measuring difference between control and experiment group we can have the observed difference is statistically significant.
A/B test architecture

SaaS service

IF (hash(UID.EID) mod 100) < E then serve experiment variant
ELSE serve control variant

Where:
UID = User ID
EID = Experiment ID (one per experiment)
E = size of experiment group for experiment EID

Users persistently in a control or experiment group remain in experiment group as E increased
(except for cases of independence assumptions)

Big time-sequence of events for all users

For each experiment e, generate reports for metrics in EID_e, and (ii) not in EID_e. Compare these results for each metric and test statistical significance.

For each experiment e, generate reports for metrics in EID_e, and (ii) not in EID_e. Compare these results for each metric and test statistical significance.

Hybrid apps/SaaS
Modern apps are often a hybrid of native, web, SaaS...

- A mobile app you can download from a store...
  - Native binaries can deliver lower latency, more controlled on-device experience...
- …which accesses web services...
  - For real-time interaction with other users, accessing live information, making payments, requesting services etc...
  - …which may contain webviews...
    - For flexible rendering of content, the structure of which doesn’t have to be specified within the mobile app itself...
- …which can even do phased releases here to manage quality!

---

**Summary**

Putting the manage/deploy/upgrade cycle to the software company is a profound change with far-reaching consequences:

- Economically:
  - Reduces customer TCO and barriers to purchasing
  - Leads to better prediction and less duplication; creates new business models

- Operationally:
  - Enables new ways of doing QA, which changes the economics of testing
  - Philips customer CD and business propagation
  - Enables new ways of doing QA, which changes the economics of testing

- Ensures competitiveness
  - Leads to better prediction and less duplication; creates new business models

- Enables building of higher quality software through increased visibility of user behavior. (N.B. with great power comes great responsibility!)
  - Behavioural analytics
  - Experiments

- Mode app store
  - A mobile app you can download from a store...
  - A mode app is...
An introduction to software testing

Andrew Rice

Some problems can be detected statically

\begin{verbatim}
fun nth 0 (x::_) = x
| nth n (x::xs) = nth (n-1) xs;
\end{verbatim}

Many problems cannot

\begin{verbatim}
var l = nth 10 [1,2,3];
\end{verbatim}

Testing checks how software performs at run-time

Test input values

Input behaviour

Output

Oracle

Pass or Fail?

Testing

An introduction to software

Some problems can be detected statically
Objectives

1. Identify different types of test
2. Be able to write a 'good' unit test
3. Know about some techniques for measuring test quality
4. Understand how testing fits into the software development process

Different types of test

- **Unit tests**
  - check isolated pieces of functionality
  - Complex & Expensive
  - Simplicity

- **Integration tests**
  - check that the parts of a system work together
  - Simple & Cheap
  - Complexity

- **E2E (end-to-end) tests**
  - simulate real-user scenarios
  - Expensive
  - Complexity

These form the 'testing pyramid'

- **Unit tests** 70%
- **Integration tests** 20%
- **E2E tests** 10%

We will consider three kinds of testing
(1) What kind of test is this?
Testing whether clicking the logout button on a website clears the cookie set in the user's browser.

(2) What kind of test is this?
Testing that the `computeShortestPath` function returns a sensible result when there are negative edge-weights in the graph.

(3) What kind of test is this?
Testing whether the room booking system is able to query a user's calendar correctly.

Unit testing demo
```java
class TestCalculateAgeInDays {
    static long calculateAgeInDays(String dateOfBirth) {
        Instant dob = dateFormat.parse(dateOfBirth).toInstant();
        Instant currentTime = new Date().toInstant();
        Duration age = Duration.between(dob, currentTime);
        long ageInDays = age.toDays();
        if (ageInDays < 0) {
            return 0;
        }
        return ageInDays;
    }
}
```
Unit testing takeaway points

Design for test: dependency injection

Test naming

One property per test

Arrange, Act, Assert

Writing assertions

JUnit lifecycle

Using @Before vs constructors

Mocking can be used to simulate a dependency

Testing whether clicking the logout button on a website clears the cookie set in the user’s browser

1. Start up a test instance of the server
2. Start a webdriver
3. Login to the site and collect the session cookie
4. Simulate a click on the button
5. Check the response from the server contains the directive to clear the cookie

% of tests that are flaky

More complex tests tend to be more flaky

Non-hermetic reliance on external systems

A ‘flaky’ test will pass and fail on the same code

Integration and E2E tests are more complicated

Table: Flaky Tests by Technology

<table>
<thead>
<tr>
<th>Technology</th>
<th>Flaky Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android emulator</td>
<td>25.46%</td>
</tr>
<tr>
<td>Java webdriver</td>
<td>10.45%</td>
</tr>
<tr>
<td>All tests</td>
<td>1.65%</td>
</tr>
</tbody>
</table>

Automated test generation can find unnoticed bugs. Many approaches exist:

- **Random testing**
  - Generate inputs at random
  - Use search to refine these inputs to make them more effective
  - Check for ‘bad things’ like a buffer overflow

See [https://github.com/google/oss-fuzz](https://github.com/google/oss-fuzz) - found thousands of security vulnerabilities in open source code.

- Check for bad things like a buffer overflow
- Use search to refine these inputs to make them more effective
- Generate inputs at random
- One example is random testing

How good are my tests? 100% coverage does not mean bug-free:

```java
class BugExample {
    public static void xPlusYMinusZ(double x, double y, double z) {
        double t = x + y;
        return t - z;
    }
}
```

```java
@Test
public void xPlusYMinusZ_correctlyCombines_smallNumbers() {
    double r = xPlusYMinusZ(2.0, 2.0, 2.0)
    // check floating point values with error tolerance...
    assertThat(r).isWithin(0.1).of(2.0);
}
```

This has 100% coverage but the code still has a bug...
Test coverage can use various properties

```java
if (a == 0) {
    ...
} else {
    if (b) {
        ...
    } else {
        if (c) {
            ...
        }
    }
}
```

Statement coverage: all lines were executed
Branch coverage: all decisions were explored at every branch
Path coverage: all paths through the program were taken
Data flow coverage: is every possible definition tested

Mutation testing can tell us how robust our tests are
- Change + to a -
- Change constant term
- Negate a condition
Verify that this causes a test to fail

Defects in software are inevitable

Expect 1-25 errors per 1000 lines for delivered software

Integrating testing into your software engineering process

80% of errors are in 20% of the project's classes

Defects in software are inevitable
Expect 1-25 errors per 1000 lines for delivered software

- When we find a problem we need to know we've fixed it
- Once we fix a bug it needs to stay fixed


80% of errors are in 20% of the project’s classes
- If we can’t test everything then prioritize the error prone parts

Continuous integration automatically runs tests

Regression testing helps with bug fixing

1. Write test that exercises existing functionality
2. Develop new code
3. Run tests to check for regressions
4. Fix bug if test fails
   - Check that test now passes

Regression testing preserves existing functionality

1. Write test that exercises existing functionality
2. Develop new code
3. Run tests to check for regressions
4. Fix bug if test fails
   - Check that test now passes

Expect 1-25 errors per 1000 lines of delivered software
We can't run all the tests on every change. Google has 4.2 million tests and 150 million test executions every day.

Need to deliver results to developers quickly.

Need to manage the execution cost of running tests.

Test suite minimisation

Choose a minimal subset of tests which maximise coverage over the project.

NP-complete problem so use heuristics.

1. Choose the essential tests
2. Choose remaining tests greedily in order of coverage added

If some test is the only test to satisfy a test requirement then it is an essential test.

Example: test suite minimisation

1. Write tests which demonstrate the desired behaviour
2. Implement new functionality
3. Check tests now pass
4. Repeat

Test set selection

Choose a subset of tests which are appropriate for the change submitted.

Test set prioritisation

Choose an ordering such that tests more likely to find a defect are run earlier.

Test Driven Development uses tests as specification

1. Write tests which demonstrate the desired behaviour
2. Implement new functionality
3. Check tests now pass
4. Repeat

Pros: guarantees that you write tests and that your code is testable, tests can be written that directly describe the customer's requirements.

Cons: early commitment to how the project will work, changes in approach are hard, some areas are more important to test than others.

See "The State of Continuous Integration Testing @Google"
Objectives

1. Identify different types of test
2. Be able to write a 'good' unit test
3. Know about some techniques for measuring test quality
4. Understand how testing fits into the software development process

```
--- E. W. Dijkstra
never demonstrate their absence...
```

Program testing may convincingly demonstrate the presence of bugs, but can never demonstrate their absence...