Software Engineering and Ethics: when code goes bad

Once upon a time, estimates for the Strategic Defense Initiative (SDI or ``Star Wars") claimed that there would be *30 million lines of code*, all bug free. This is at least three orders of magnitude greater than ever has been achieved. ... What have we learned since then?

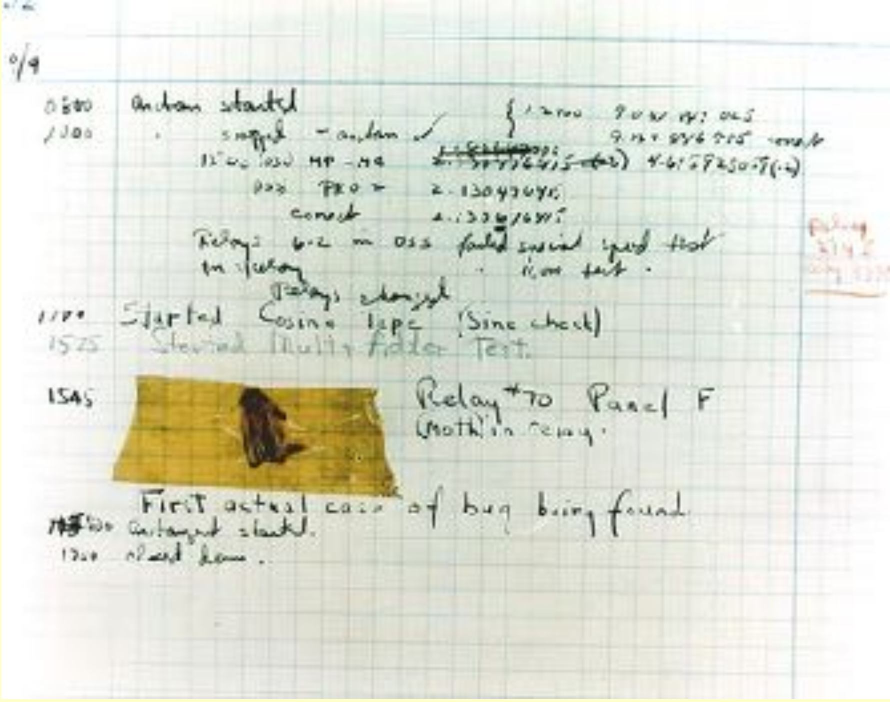
Lecture Plan

> *It is a bad plan that admits of no modification*
>
> Publilius Syrus

- Software Crisis
- Ethics
- Software Engineering
- Software Engineering and Ethics
- The root of the problem - computer science boundaries?
- When code goes bad - education through classic examples
- Proposals for the future

Software Crisis

Do you recognise this?



9/9

0800 Antan started

1000 " stopped - antan ✓ {1.2700 9.037 847 025
9.037 846 995 conect

13" UC (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ (-2) 4.615925059(-2)

(033) PRO 2 2.130476415

conect 2.130676415

Relays 6-2 in 033 failed special speed test

in relay " " 10.000 test.

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Multi Adder Test.

1545 Relay #70 Panel F

(moth) in relay.

First actual case of bug being found.

1630 Antangent started.

1700 closed down.

Relay 2145
Relay 3376

Software Engineering and “Bugs”

When everyone is wrong everyone is right

Nivelle de La Chaussee

QUESTION: What’s the difference between hardware and software ?…

> *buy some hardware and you get a warranty, buy some software and you get a disclaimer*

The *software crisis:*

Does this really exist?

- *always late*
- *always over-budget*
- *always buggy*
- *always hard to maintain*
- *always better the next time round … but never is!*

This doesn’t seem *right* … where are our ethics?

Is there really a crisis? …

… look at the advances we have made

To avoid crisis, just hire the best people

Success in software development depends most upon the quality of the people involved.

There is more software to be developed than there are capable developers to do it.

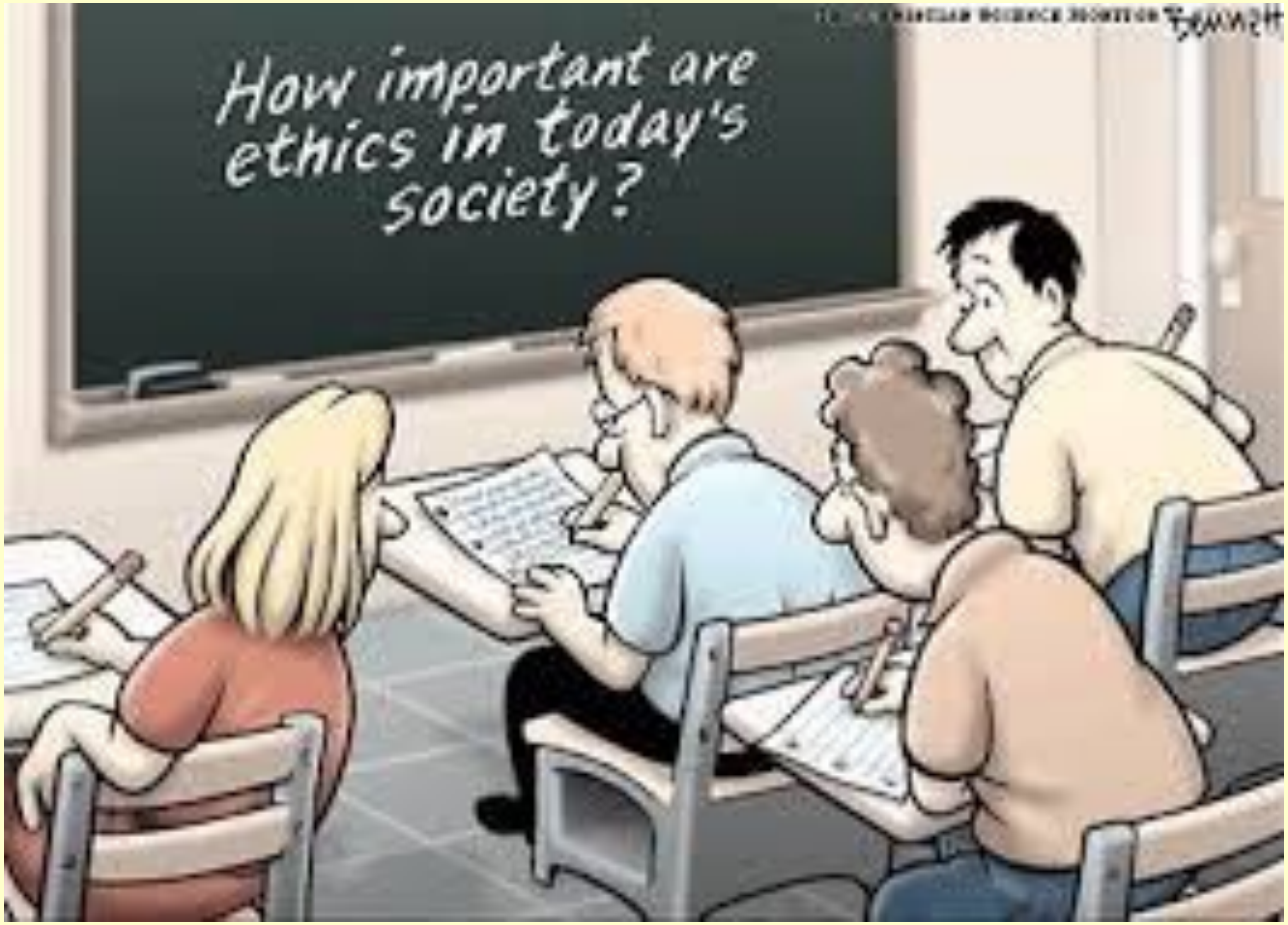
Demand for engineers will continue to outstrip supply for the foreseeable future.

Complacency has already set in … some firms acknowledge that many of their engineers make negative contribution. Some engineers don’t care.

Hence, more and more software development will be in crisis.

What can we do about this? … Try and make software engineering a true engineering discipline. Try and make better (ethical) engineers.

Ethics



What are ethics and what can they do for me?

> *A man who moralises is usually a hypocrite*
>
> Oscar Wilde

The doctrine of morals; A moral philosophy; A system of moral principles

Morals - generally to do with right and wrong

Ethics in software engineering would define an acceptable code-of-practice.

We already have such codes-of-practice in law, medicine, architecture, etc..

Usually, they are defined and enforced by a particular professional body…

Such structures are in place for engineers.

However, for software engineers there is only a voluntary code of practice which is ill-defined, imprecise, ambiguous and not specific to problems particular to software engineering.

Typical attitude: *I've got better things to think about*

This attitude is not unique to software engineers … the Challenger Disaster provides a good example.

The Challenger Disaster...

A 1-in-a-billion bit of bad luck

According to the R*eport of the Presidential Commission on the Space Shuttle Challenger Accident,* evidence pointed to the right solid rocket booster as the source of the accident… this is well known because of **Richard Feynman**

In January of 1987, nearly a full year after the Challenger exploded, **Roger Boisjoly** (A NASA scientist) spoke at MIT about his attempts to avert the disaster during the year preceeding the Challenger launch:

> In 1985 **Boisjoly** began work to improve the O-ring seals which connect segments of the solid rocket booster. [...] He repeatedly warned them of potential dangers! Yet, a flawed design went into production as the scientists' issues were continually overlooked.

For his honesty and integrity leading up to and directly following the shuttle disaster, **Boisjoly** was awarded the Prize for Scientific Freedom and Responsibility by the American Association for the Advancement of Science.

FOOTNOTE: None of his superiors was ever taken before a court-of-law (some of them were even promoted a few months after the enquiry!)

Where is the science in software engineering?

> *Computer Science can't demonstrate to the software engineering people on a sufficiently large scale that what it is doing is of interest or importance to them.*

Christopher Strachey

The Challenger example illustrates the need for the engineers to listen to the scientists… this is even more of a problem in software engineering.

Problem 1: software engineering is technology driven and therefore our fundamental understanding of computers, programs, algorithms, information, etc … *would seem to be* aiming at a moving target

Problem 2: computer scientists and software engineers speak different languages, and are unsympathetic to each other's problems

Problem 3: efforts to bring mathematical rigour to programming quickly reach a level of complexity that makes the techniques of verification subject to the very concerns that prompted their development in the first place!

Software Engineering and Ethics

> *Morality is a private and costly luxury*
>
> Henry B Adams

The study of ethics is a necessary part of the education of every software engineer.

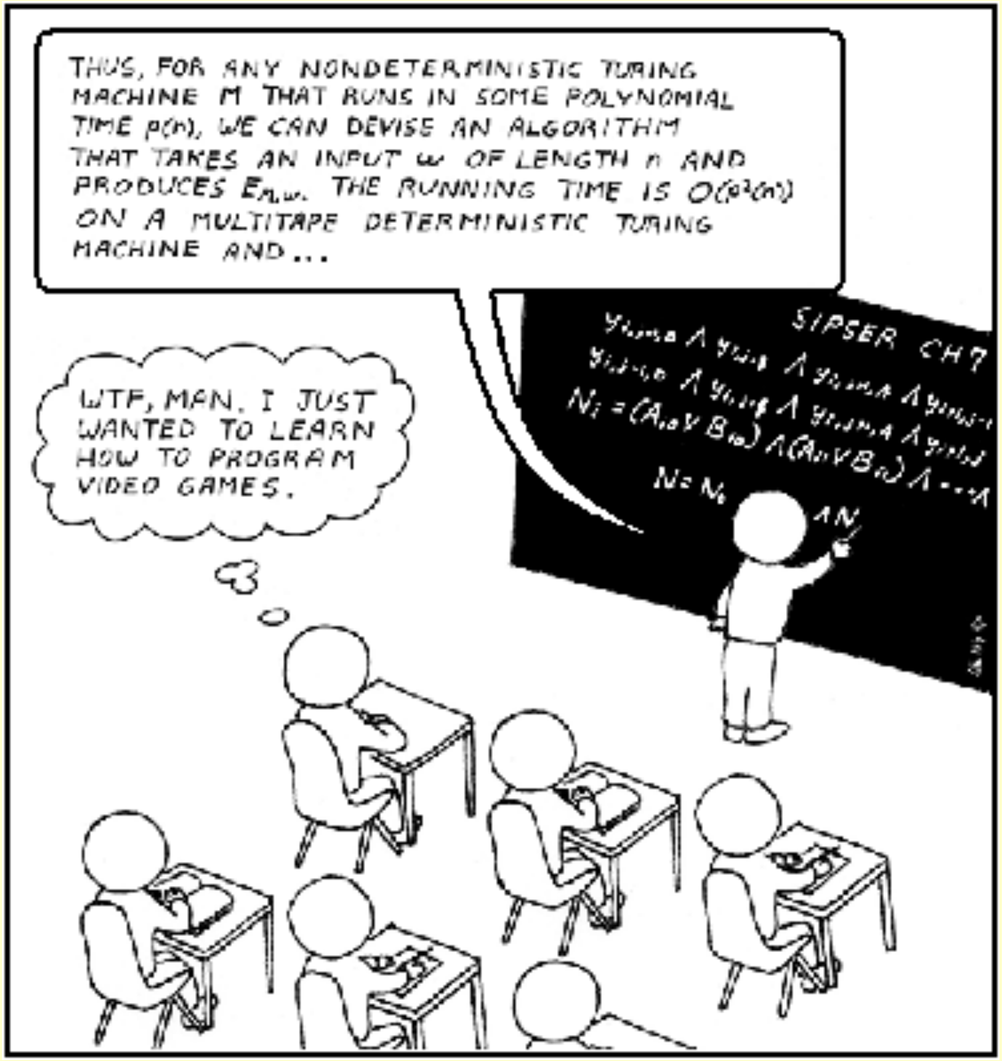
Software Engineering Ethics involves any decision made by a software engineer during the design, development, construction and maintenance of computing artifacts. Learning how to make these decisions is an essential part of the technical education of a software engineer.

•**Base level** - don't lie, cheat, steal, hurt etc..- an ethical commitment to minimal morality!

•**Professional level** - like any other professional ethics in requiring a special commitment to the public served and affected by the profession - an ethical commitment to public well being.

•**Engineering level** the responsibilities within software engineering, which are closely related to the state of the art: an ethical commitment to quality work.

The root of the problem - computer science boundaries?


THUS, FOR ANY NONDETERMINISTIC TURING MACHINE M THAT RUNS IN SOME POLYNOMIAL TIME p(n), WE CAN DEVISE AN ALGORITHM THAT TAKES AN INPUT w OF LENGTH n AND PRODUCES $E_{M,w}$. THE RUNNING TIME IS $O(p^2(n))$ ON A MULTITAPE DETERMINISTIC TURING MACHINE AND...
WTF, MAN. I JUST WANTED TO LEARN HOW TO PROGRAM VIDEO GAMES.
SIPSER CH7
$N_i = (A_{i0} \vee B_{i0}) \wedge (A_{i1} \vee B_{i1}) \wedge \cdots \wedge$
$N = N_1 \wedge N$

Fundamental Boundaries of Computation

Man is still the most extraordinary computer of them all

JFK

To be *as good a software engineer as you can be* requires you to understand the boundaries of computation as defined by computer science.

To ignore the theoretical basis should open you to accusations of malpractice… no other engineering/professional discipline would let you get away with it.

I have spoken to senior, experienced, well-respected software engineers whose projects have got into a crisis because they didn't understand the following fundamentals:

- **Computability**
- **Complexity**
- **Correctness**
- **Common Sense**

The overall malaise is one of **complacency**

Computability

The Feynman Problem-Solving Algorithm:
(1) write down the problem; (2) ***think very hard;*** *(3) write down the answer.*

Murray Gell-Mann

The **Church-Turing** thesis proposes that each one of a variety of different formal systems adequately define the intuitive concept of (effectively) *computable*

The complementary nature of the work by Godel, Church, Turing, Rosser, Kleene and Post is a great illustration of the way in which mathematics and TCS (sometimes) walk along the same paths.

Undecidability is, IMHO, the most important lesson for all computer scientists.

The Turing machine model of computation is fundamental in that it identifies a set of machines which, through historical evidence, almost certainly correspond to modern electronic computers.

COMMENT: it is surprising how many software engineers waste their time trying to solve the *halting problem*

Complexity

> *Attempt the end, and never stand to doubt;*
> *Nothing's so hard, but search will find it out.*
>
> Robert Herrick

A different, yet related, problem is that of *complexity* – not all computational problems which can be solved in principle can be solved in practice: the computational resources required may be prohibitive.

In the standard (Turing) model of computation, complexity theory identifies time and space as fundamental resources; and within this framework the most challenging area of study is precisely the boundary between ‘easy’ and ‘difficult’ problems (P vs NP).

Recent research has applied evolutionary models of computation to stretch the boundary between P and NP, in specific subsets of classes of difficult problems.

COMMENT: it is surprising how many software engineers waste their time by not understanding intractability.

Common Sense

> *Logic is one thing and common sense another*
>
> Elbert Hubbard

It is accepted that no current computer system exhibits *intelligent* behaviour comparable to that seen in most living organisms.

Intelligence - like computer science and software engineering - is about problem solving. We must ask why some problems appear to be more complex than others.

In every other engineering discipline, estimations of complexity are built upon scientific foundations… not ours!

- Be wary of any SE model which claims to exhibit common sense when judging complexity
- Be wary of any SE model which claims to remove the need for common sense when judging complexity

Many *great results* in software engineering research are just common sense

COMMENT: it is surprising how many software engineers waste their time by trying to build intelligence when they don't know what it is

Correctness

> *I have a good idea why it's hard to verify programs. They're usually wrong.*
>
> Manuel Blum

How to prove that a system is correct (bug free)

Define correctness as a mathematical relationship between a specification and an implementation.

The specification may contain a set of properties that need to be verified.

Verification is not validation .. where we check correctness of the original specification ‘informally’… and formally (consistency and completeness)

Systems may be incorrect if they are developed from an incorrect specification -- this is a requirements modelling problem

Systems may be incorrect if a design decision introduces unwanted behaviour.

COMMENT: it is surprising how many software engineers do not understand that the ends of this development chain are the weakest points… and it is even more surprising how many software engineers do not know what correctness is!

Complacency

> *He that is too secure is not safe*
>
> Thomas Fuller

Developers:
Anyone can build and sell software

Educators:
Software development is not a profession because it is not taught like one

Researchers:
Much of their work is camouflaged theory or poor project management

Qualifications:
No qualification in SE guarantees an acceptable minimum standard of engineer

Customers:
Quality of life depends on quality of software, yet we accept faulty software as a matter of course

Everyone writes software!

> *The secret of all good writing is sound judgement*
>
> Horace

Who programs? -

Engineers, scientists, computing graduates, psychologists, mathematicians, businessmen, teachers, gardeners, school children ….

Who knows the science of software?-where is the sound judgement?-

Few of the above, even the graduates!

An unthinkable solution:

Stop everyone from programming

A better (more ethical) solution:

Provide a clear, 2-tier system of software engineers where the qualified engineers continually work to attack the root of the problem.

The root of the software crisis:

Current software standards are weak, superficial, and not based on software science

When code goes bad - education through classic examples

When code goes bad … learning by example

Learn as though you would never be able to master it

Confucius

There are 1000s of software horror stories (many of them untrue)

They are often memorable because of their comic nature

They can also be tragic

Some are trivial to explain to a non-engineer

Some are very complex to understand, even for software engineers

The most important thing is that they make us think, and that we can learn from them.

APOLLO 11

Ignorance is the only universal constant that is universally ignored

Problem: Apollo 11's main navigation system crashed and secondary system had to be installed (on the fly) .. reducing the number of scheduled tests by 50%, at an estimated cost of 120 million dollars.

Reason: The Apollo 11 software had a bug that made gravity repulsive instead of attractive …

Analysis: The chief engineer asked the programmer involved did he learn nothing from Sir Isaac Newton?

Who was to blame: ??

Apollo 11 - would unsigned integer types have helped?

```
void foo(void)
{
 unsigned int a = 6;
 int b = -20;
 (a+b > 6) ? puts("> 6") : puts("<= 6");
}
```

This example C code illustrates 1 of the many problems with unsigned integers, if you don’t know your language very well!

Gemini V

You mean Pi isn't exactly twenty two over seven? Nobody told me

Problem: Gemini V landed 100 miles off course

Reason: some programmer was a bit weak on physics. The correct elapsed distance (the key variable in this case) should have been calculated by using the Sun as a fixed reference point and not a point on the Earth. The programmer instead used elapsed time, thinking the reference point on Earth returns every 24 hours. (The value of 24 is ~ 1.00273790935 of the correct value). This difference results in <u>only</u> a few hundred miles in our solar system!

Analysis: the chief software engineer stated that none of his team knew about the inaccuracy of the 24-hour day … and had coded it as an exact integer!

Who was to blame: ??

Gemini V - rounding errors are universal

```
public class Main {
  public static void main(String[] args)
{
    double a = 0.7;
    double b = 0.9;
    double x = a + 0.1;
    double y = b - 0.1;
    System.out.println(x == y);
  }
}
```

Consider this Java code, what would you expect it to give as output?

Comparing ‘reals’ should always include a precision/error value -

```
System.out.println(Math.abs(x - y) < 0.0001);
```

NORAD - Modelling the moon

Why would we want to shoot at the moon?

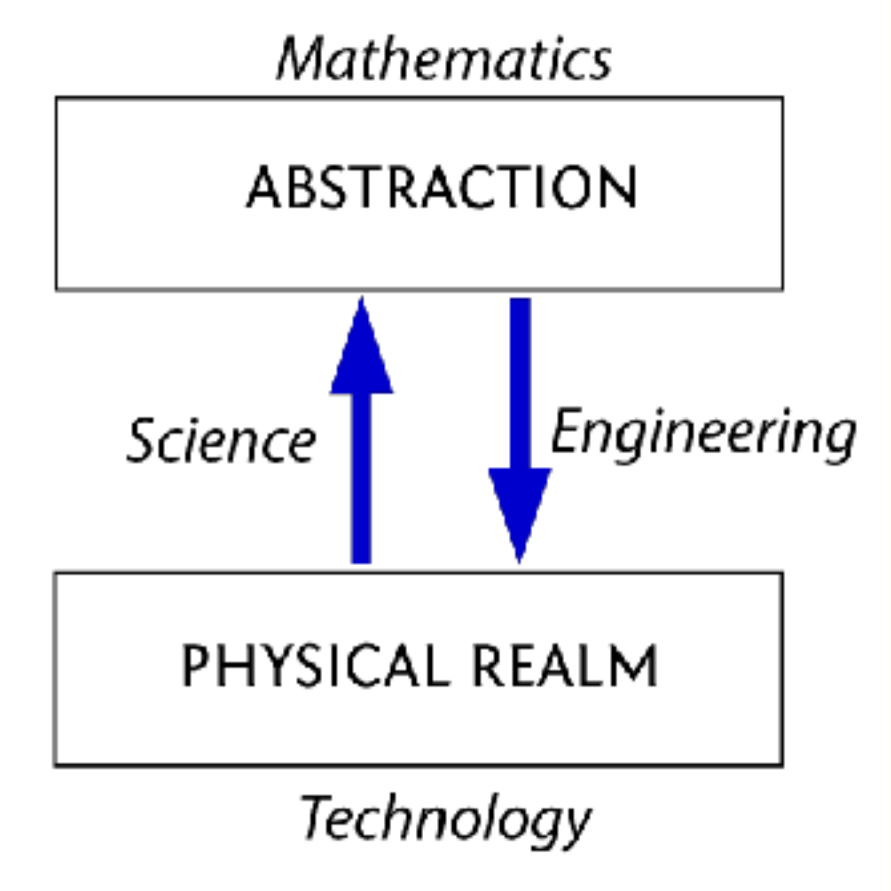
Problem: On October 5, 1960, the North American Defense Command (NORAD) went to 99.9% alert … just minutes from a *defensive counter strike*

Reason: programmers forgot that the Moon rises and would show on radar.

Analysis: the moon did not even appear in the requirements model … *why would we want to shoot at the moon?*

Who was to blame: ??

NORAD - Modelling the moon



Finding the right level of abstraction is very difficult

NORAD - training for disaster

> *The quickest way of ending a war is to lose it*
>
> George Orwell

Problem: On June 3, 1980, and again on June 6, 1980, NORAD (again!) went to full alert and tried to launch everything.

Reason: Training tapes had been loaded onto the live system.

Analysis: there was no way physically or electronically to distinguish training tapes from recording tapes without actually running the tapes!

Who was to blame: ??

NORAD - training for disaster

Exactly the same type of problem has been reported for e-voting machines

HCI Problems & 'big' data

Without knowledge, life is no more than the shadow of death

Moliere

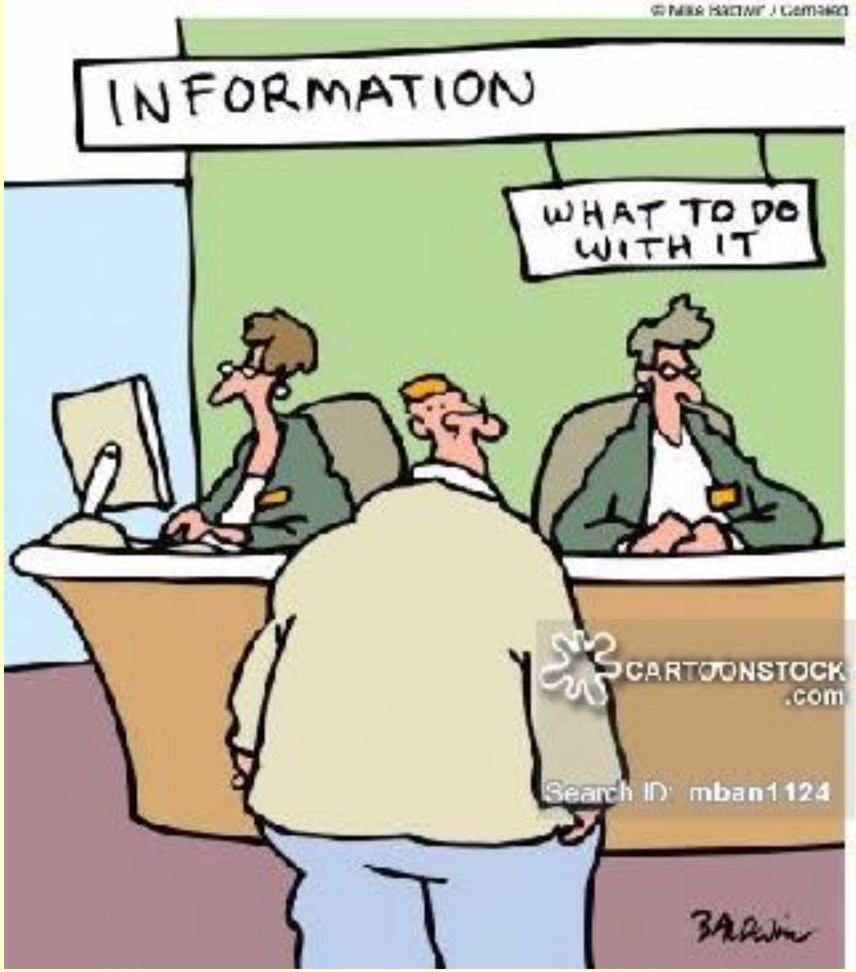
Problem: On July 3, 1988, the U. S. Aegis cruiser Vincennes - jammed to the gunnels with computers, radars and the fanciest equipment afloat - shot down an Iranian airliner that had complied with every restriction placed on a civilian aircraft in the area.

Reason: it appears that the crew panicked and misinterpreted the information presented -- in effect, they drowned in information. The one man who needed the information, Captain Rogers, could not get a clear picture of what was going on because there was no one computer station that had the complete picture.

Analysis: Reverse engineering the scenario showed that the data across different machines was inconsistent … no wonder the poor man made a bad decision!

Who was to blame: ??

HCI Problems: information overload



Boeing 767 - the biggest glider in the world

We thought, because we had power, that we had wisdom

Stephen Vincent Benet

Problem: In July of 1983, Air Canada Flight 143, a brand-new Boeing 767, made an emergency landing at an abandoned RCAF airfield at Gimli, Manitoba.

Reason: Their problems began when a microprocessor that monitors fuel supply malfunctioned. This cut off the engines and the electrical power.

Analysis: Boeing engineers thought it would be impossible to lose both engines and therefore electrical power. But Flight 143 did.

Who was to blame: ??

Australian Commonwealth Bank

A bank is a place that will
lend you money if you
can prove that you don't need it

Bob Hope

Problem: 1.5 million bank accounts had data corrupted and a whole day's transaction had to be re-entered by hand … some complaints were still not resolved 5 years later and court cases cost millions of dollars.

Reason: On the night of February 25, 1988, the Australian Commonwealth Bank doubled all debits and credits.

Analysis: A simple spurious 0 in a data file was not picked up and resulted in credits being multiplied by 2 … to compensate, debits were also multiplied. This prompted the manager to make the now famous-in-folk-lore comment: `

The effects of software errors are limited only by the imagination..

Who was to blame: ??

Ariane 5

Everything in space obeys the laws of physics … except the software

Wernher von Braun

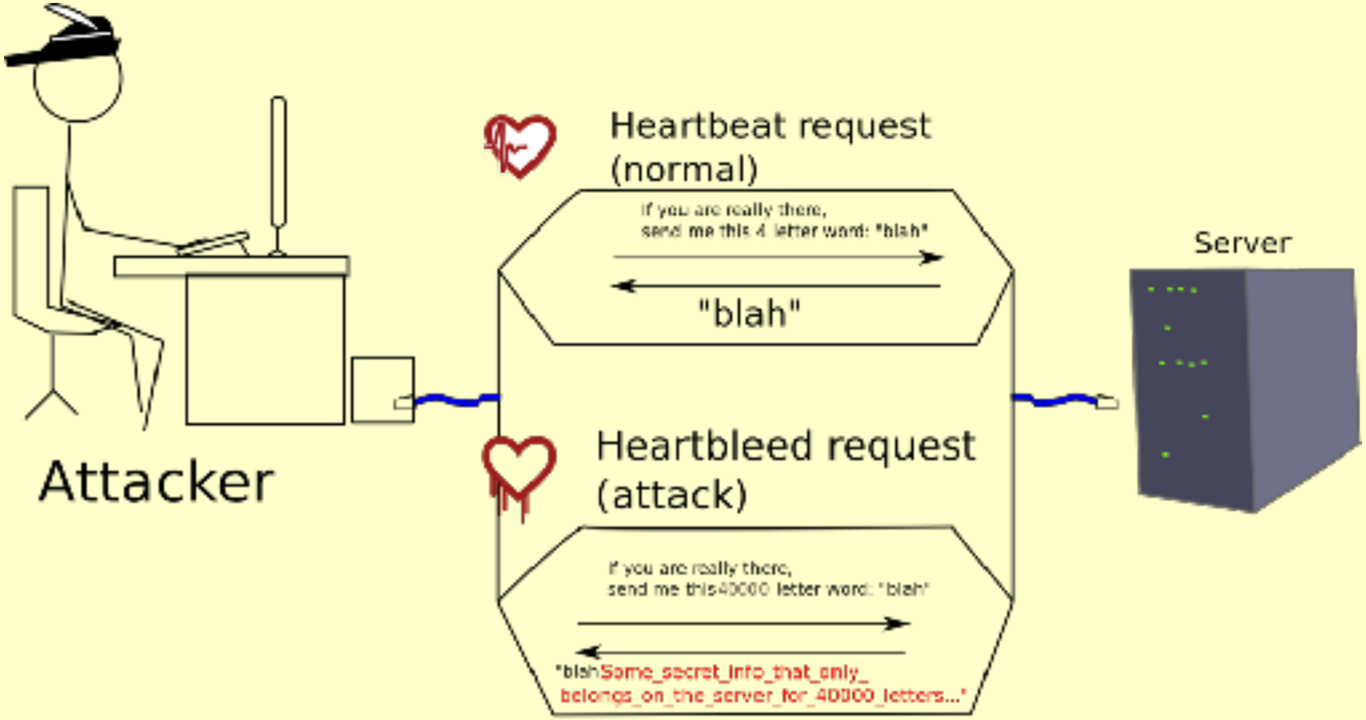
Problem: the rocket exploded on take-off

Reason: a 16-bit integer was used to perform a 32-bit calculation… plus a few other things to do with the polymorphic type system which I won't go into

Analysis: due to over-zealous re-use of code from Ariane 4 (which never exploded!)

Who was to blame: ??

OpenSSL Heartbleed



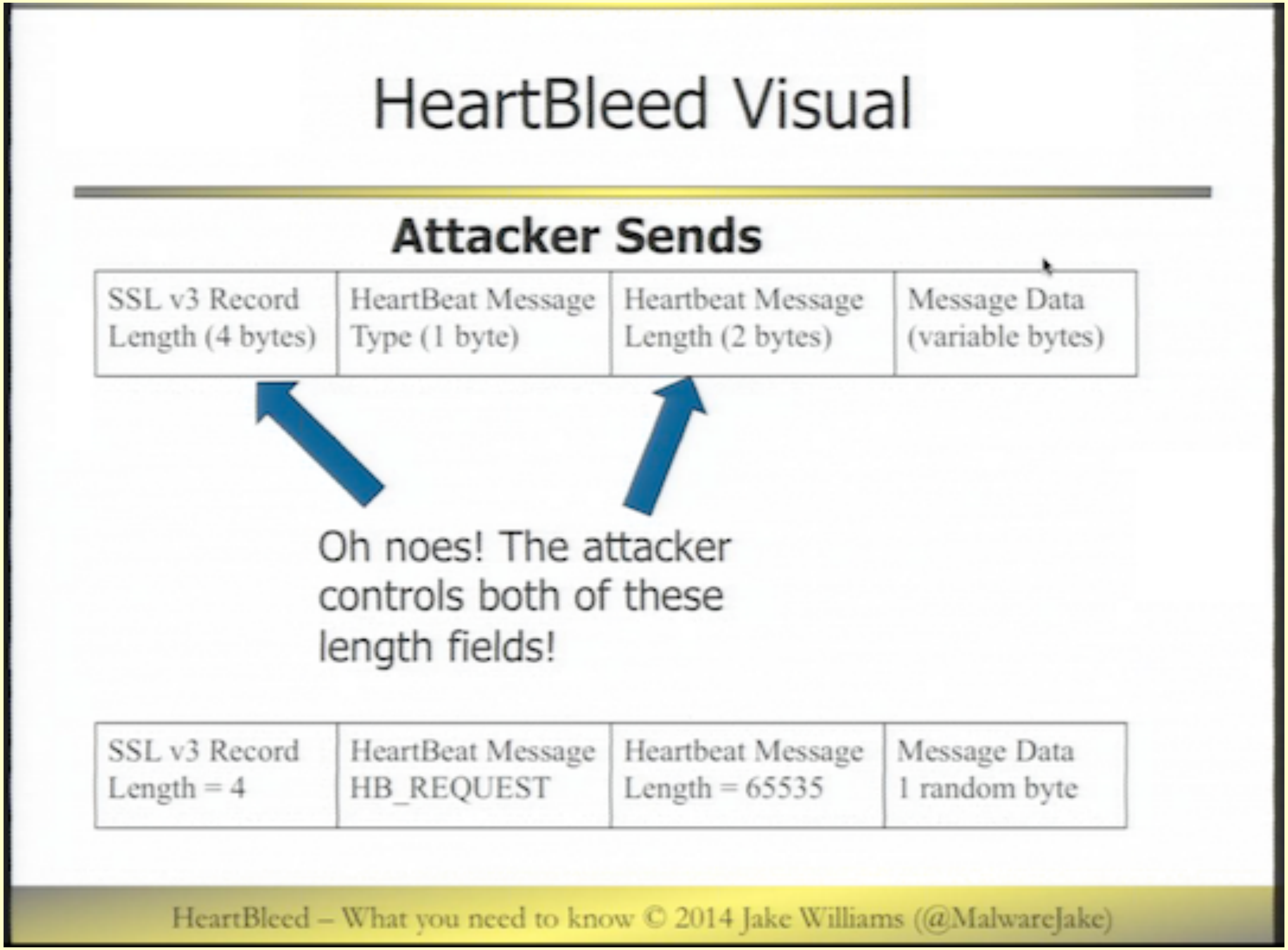
If you are really there, send me this 4 letter word: "blah"

"blah"

If you are really there, send me this40000 letter word: "blah"

"blahSome_secret_info_that_only_belongs_on_the_server_for_40000_letters..."

OpenSSL Heartbleed



HeartBleed Visual

Attacker Sends

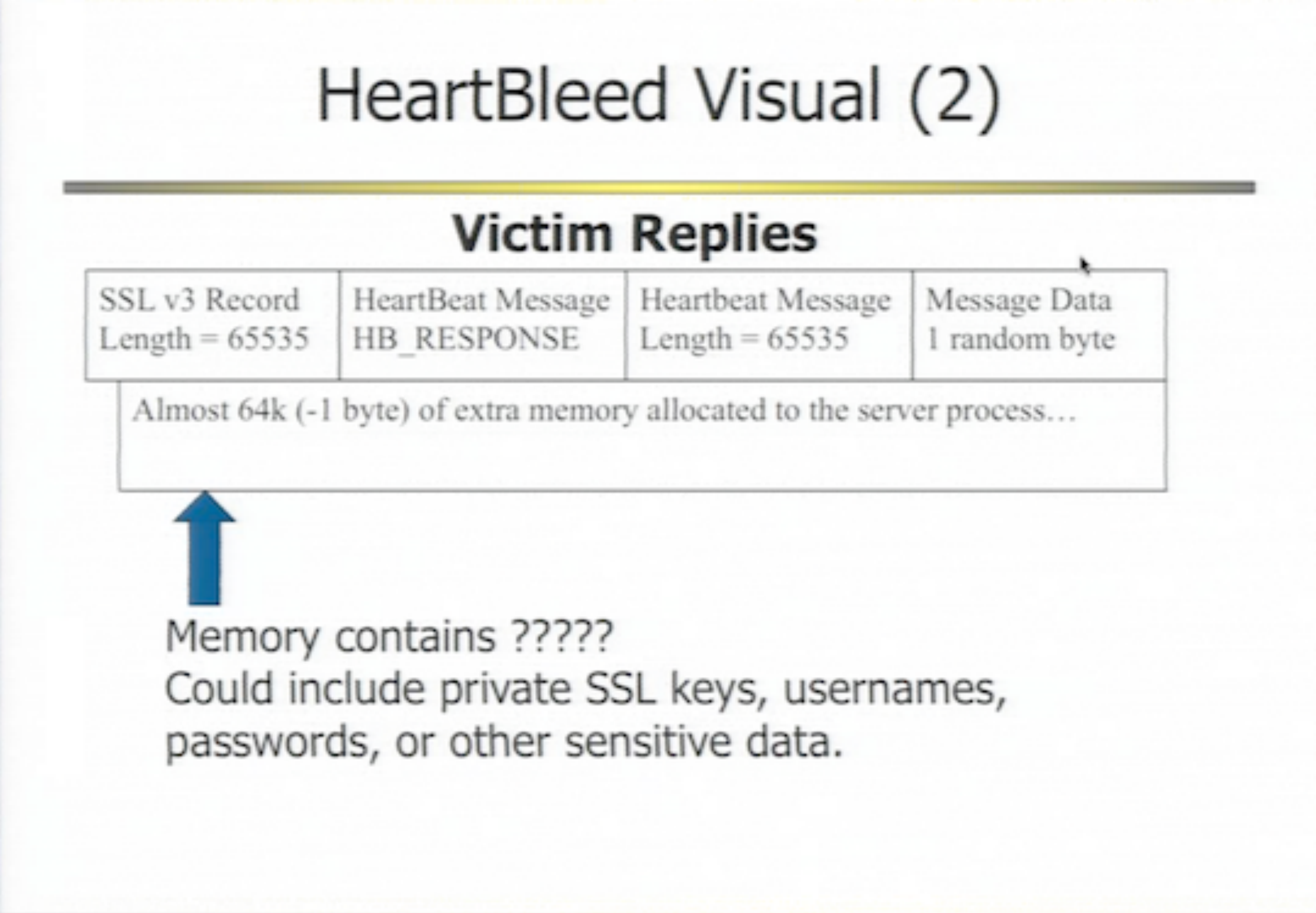
SSL v3 Record Length (4 bytes)	HeartBeat Message Type (1 byte)	Heartbeat Message Length (2 bytes)	Message Data (variable bytes)

Oh noes! The attacker controls both of these length fields!

SSL v3 Record Length = 4	HeartBeat Message HB_REQUEST	Heartbeat Message Length = 65535	Message Data 1 random byte

HeartBleed – What you need to know © 2014 Jake Williams (@MalwareJake)

OpenSSL Heartbleed



HeartBleed Visual (2)

Victim Replies

SSL v3 Record Length = 65535	HeartBeat Message HB_RESPONSE	Heartbeat Message Length = 65535	Message Data 1 random byte
Almost 64k (-1 byte) of extra memory allocated to the server process…			

Memory contains ?????
Could include private SSL keys, usernames, passwords, or other sensitive data.

OpenSSL Heartbleed

See **http://git.openssl.org/gitweb/?p=openssl.git;a=commitdiff;h=96db902** for a fix

See **http://www.seancassidy.me/diagnosis-of-the-openssl-heartbleed-bug.html** for analysis

Apple’s “`goto fail;`” SSL bug

`Consider` the`SSLVerifySignedServerKeyExchange` function, found in the `sslKeyExchange.c` file -

```
. . .
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;  /* MISTAKE! THIS LINE SHOULD NOT BE HERE */
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);
. . .
```

Can you see the problem? Who is to blame?

Proposals for the future

"They shouldn't allow humans to drive!"

Proposals for the future

Not every end is the goal

Friedrich Nietzsche

We need professional software engineers

We need a code-of-practice (enforced by a professional body)

We need acceptance and teaching of the ethical approach

We need a theoretical (formal) foundation

We need to stop being complacent