LISP
LABORATÓRIO DE INFORMÁTICA, SISTEMAS E PARALELISMO

Software Reliability and Security
The Formal Methods perspective

Simão Melo de Sousa

UAlg
UNIVERSIDADE DO ALGARVE

UNIVERSIDADE
BEIRA INTERIOR

UNIVERSIDADE
DE ÉVORA

what a nice car!

Ford F150 Raptor V6 Turbo

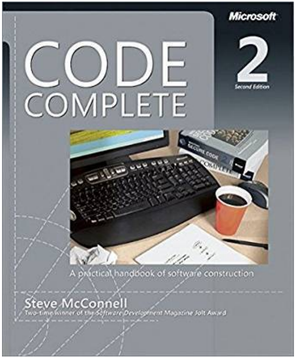
450 HP, 100 km/h in 6,1 seg, 2580 Kg

on a totally different matter

industry average: about **15 - 50 bugs** per **1000 lines of code**

(from Code Complete - Steve McConnell)

fortunately this ratio tends to go under this lower bound in industry areas considered sensitive or critical


Microsoft
CODE
COMPLETE
2
Second Edition
A practical handbook of software construction
Steve McConnell
Two-time winner of the Software Development Magazine Jolt Award

returning to our appetizer

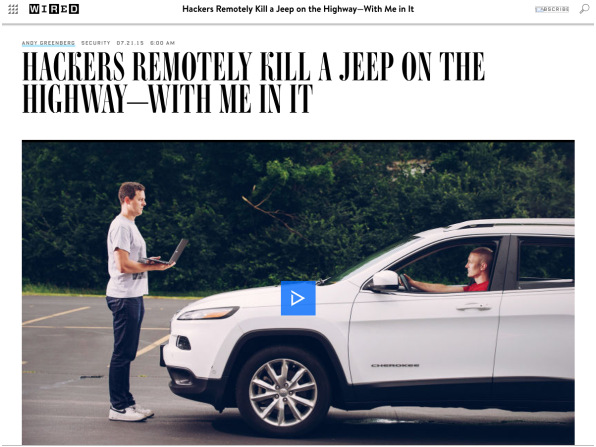
Ford F150 Raptor V6 Turbo
450 HP, 100 km/h in 6,1 seg, 2580 Kg

contains more than 150 millions of lines of code!

(source - link)

this is an actual trend in the automotive industry

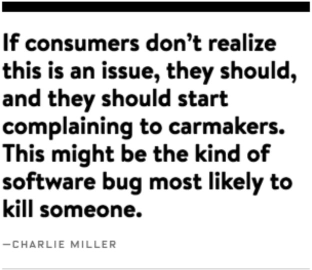
predictable outcome



WIRED Hackers Remotely Kill a Jeep on the Highway—With Me in It SUBSCRIBE

ANDY GREENBERG SECURITY 07.21.15 6:00 AM

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



If consumers don't realize this is an issue, they should, and they should start complaining to carmakers. This might be the kind of software bug most likely to kill someone.

—CHARLIE MILLER

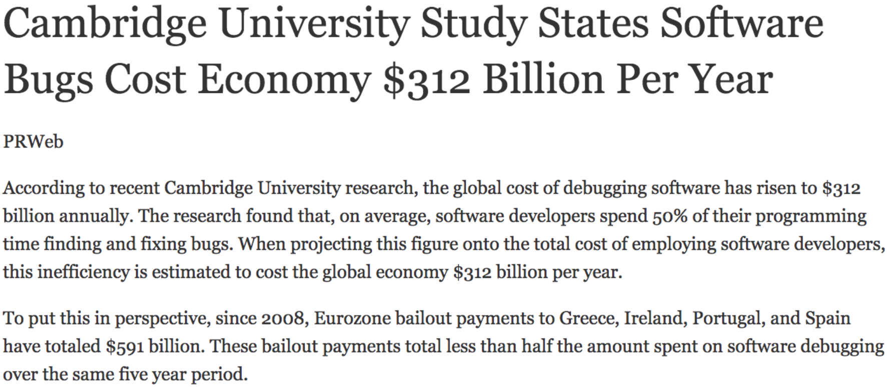
(source - link)

consequences of bugs

bugs can cost

- lives (e.g. a bug in a pacemaker)
- money (e.g. a bug in a transaction system)
- business, customer trust (e.g. a bug in a voting system, or a third party business system)
- missions (e.g. a bug in the control system of a satellite)
- privacy (e.g. a bug in a sensible information system)
- etc...

the economical cost of bugs



Cambridge University Study States Software Bugs Cost Economy $312 Billion Per Year

PRWeb

According to recent Cambridge University research, the global cost of debugging software has risen to $312 billion annually. The research found that, on average, software developers spend 50% of their programming time finding and fixing bugs. When projecting this figure onto the total cost of employing software developers, this inefficiency is estimated to cost the global economy $312 billion per year.

To put this in perspective, since 2008, Eurozone bailout payments to Greece, Ireland, Portugal, and Spain have totaled $591 billion. These bailout payments total less than half the amount spent on software debugging over the same five year period.

(source: link)

the actual picture, on the reliability side

we are living times that demand **mass production** of software and requires **software everywhere**

but producing high-quality software is a **serious matter** and require **strong skills**

software making tools (e.g. programming languages, etc.) classically establish a compromise between **efficiency** and **safety/security**

this compromise usually tends to **favor efficiency**

on the reliability side: the actual picture

this compromise usually tends to **favor efficiency**

- trust on the **assumption** that the programmer **knows very well what he is doing**, he is in charge of the safety and security (e.g. safe memory management)

great power comes with great responsibility

- from the programming language research community perspective, this trade-off is nowadays an anachronism: **we can have both!**

the actual picture, on the security side

a computer/software system is only as secure/reliable as its weakest link

generally speaking, **WE** (users, programmers, clients etc.) are this link...

but, putting aside the social engineering perspective, the security of computer systems is now much less challenged by poor cryptographic mechanisms ...

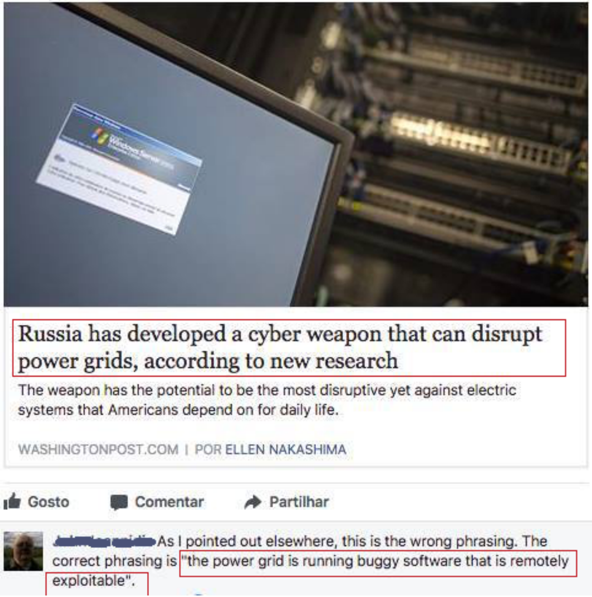
... than by **software failures**

claim

$$\boxed{\textbf{every bug is an attack waiting to be exploited}}$$

$\Longrightarrow$ software security seen as a special case of software reliability

claim.... as seen on a *reliable* social medium



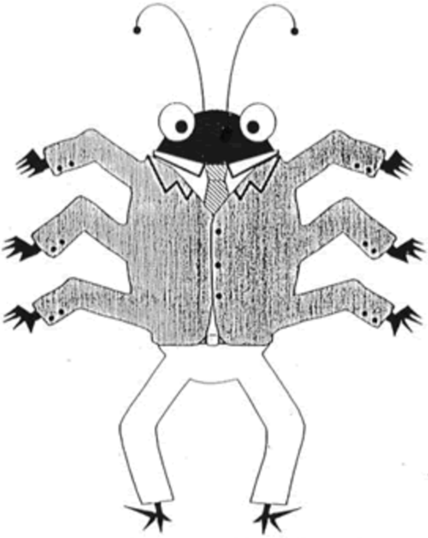
Russia has developed a cyber weapon that can disrupt power grids, according to new research

The weapon has the potential to be the most disruptive yet against electric systems that Americans depend on for daily life.

WASHINGTONPOST.COM | POR ELLEN NAKASHIMA

[redacted] As I pointed out elsewhere, this is the wrong phrasing. The correct phrasing is "the power grid is running buggy software that is remotely exploitable".

who is the scariest? the bug or Terminator?



Vint Cerf about the internet (of things)

VINT CERF: BUGGY SOFTWARE IS SCARIER THAN A ROBOT TAKEOVER

(source : link)

Robots won't take over humans, but buggy software might, according to the Google exec known as the "father of the Internet."

(...)

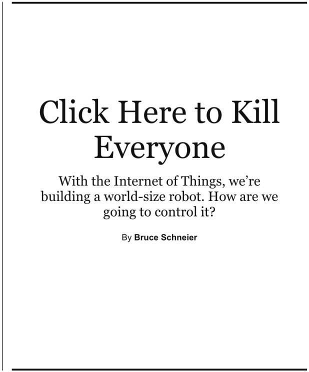
"I'm actually not as worried about artificial intelligence and robots as I am with software and robots," he said during an event Monday at the Italian embassy. "If there are bugs in the software and some device is operating autonomously with regard to that software, the bugs can cause bad things to happen."

(...)

People aren't careful enough about updating their software – especially when such software is increasingly running heating and ventilation, among other functions – and also making sure the software they're updating doesn't include malware.

"The big headline I worry about is, '100,000 Refrigerators Attack Bank of America,'" he said.

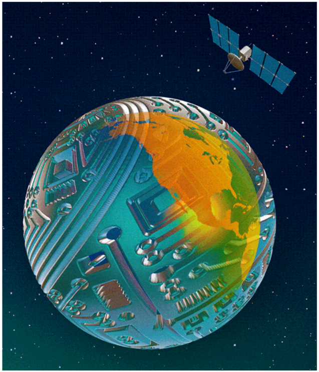
Bruce Schneier about the internet (of things)



Click Here to Kill Everyone

With the Internet of Things, we're building a world-size robot. How are we going to control it?

By **Bruce Schneier**



(source : link)

Truism No. 1: On the internet, attack is easier than defense.

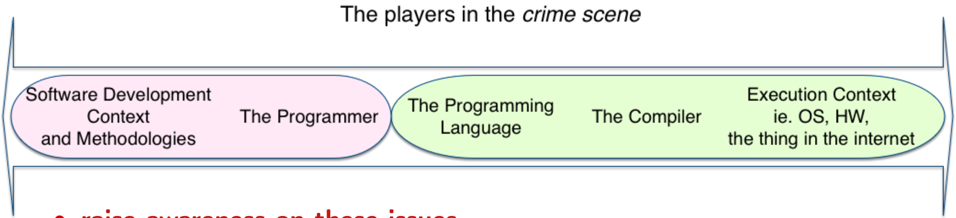
Truism No. 2: Most software is poorly written and insecure.

Truism No. 3: Connecting everything to each other via the internet will expose new vulnerabilities.

Truism No. 4: Everybody has to stop the best attackers in the world.

Truism No. 5: Laws inhibit security research.

What researchers at LISP are doing on this matter?


The players in the *crime scene*
Software Development Context and Methodologies
The Programmer
The Programming Language
The Compiler
Execution Context ie. OS, HW, the thing in the internet

- **raise awareness on these issues**
- on the process/methodology side: better requirement engineering, better software methodologies that make reliability and accountability a central concern
- creating new or enhancing:
  - programming languages, with better safe programming support
  - (static) advanced type systems
  - validation and verification tools
  - static analysis tools
  - programming environments
  - runtime safety/security mechanisms

theoretical limits of Validation and Verification tools

Semantic specification: a rigorous specification of expected behavior (or expected properties, or set of correct executions)

Rice theorem (1953)

Considering a Turing complete language, any non trivial semantic specification is not computable

- *computable* means "can be found/calculated by a computer"
- intuition: there is no hope for a "finding all bugs" algorithm starting with only the program code
- therefore all **interesting properties are not computable** (buffer/arithmetic overflows, information leakage, abscence of crash, etc.)

Towards partial solutions

there is no hope for programming tools that ensure the absence of all bugs

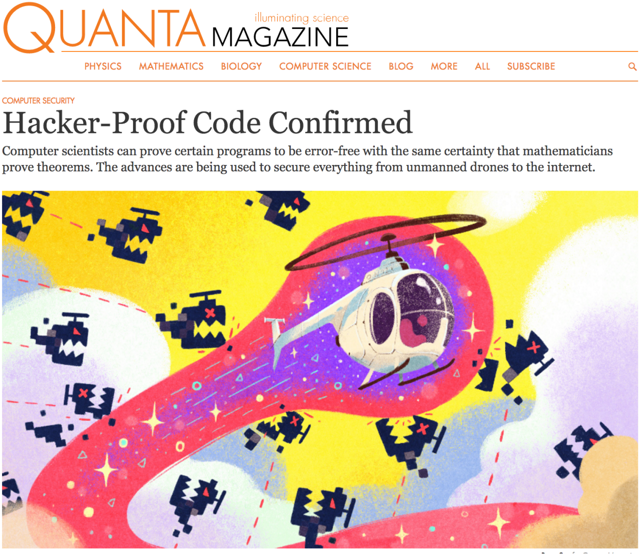
solution: **solve a weaker problem**

several compromises can be made:

- provide interactive tools that take advantage of human ingenuity to find/avoid bugs: **expressiveness, but loss of automation**
- understand the theoretical limits and try to provide tools that go as close as possible to it: **focus and automation, but incompleteness** (i.e. innocuous programs can be erroneously tagged as “possibly problematic” - the converse does not occur!)

nevertheless, interesting results have been reached

do all these have a practical impact? … one example



COMPUTER SECURITY

Hacker-Proof Code Confirmed

Computer scientists can prove certain programs to be error-free with the same certainty that mathematicians prove theorems. The advances are being used to secure everything from unmanned drones to the internet.

(source: link)

Reliability and Security @ LISP, very few examples

Certified Operating Systems or Virtual Machines

Java Card in COQ (JavaCard = multiple applications programmable smart cards like Cartão de Cidadão, eID, mobile SIM card, credit cards, etc.)

Formal verification of the Java-Card Platform in COQ

1. a specification and prototype of JavaCard Execution Platform and
2. the proof that (a R. Milner quote)

   Well-typed (JavaCard) Programs cannot go wrong

3. A provably correct implementation of the ByteCode Verifier (BCV), a crucial security module based on static program analysis.

Upgradable Highly Critical RealTime Operating System

CPU consumption bounds certificate mathematically designed and carried with its associated critical code

this certificate can be safely verified and used as a basis for safe upgrade policies (for instance, in satellites or robotic space missions)

PROSINAL - EFACEC with UBI/LISP

- goal : CENELEC SIL4 Railway Signaling System for the **Metro do Porto** - linha Aeroporto Sá Carneiro
- challenge: Software layer design, validation and certification (SIL4 - the highest) using Formal Methods in a very restrictive normative context (CENELEC). The first of its nature, to the best of our knowledge
- auxiliary Mission: set-up and training of a (formal) Validation team in a industrial context
- extension of the Scade toolset to deal with Function Block based HW
  - a (pencil and paper proved) translation methodology and
  - a (HW level) testing framework with tests generated from SCADE models – allow for a better confidence on the translation process

(Highlight: First Signaling System **in the world!** formally and completely proved from scratch that reach the new CENELEC SIL4 certification)