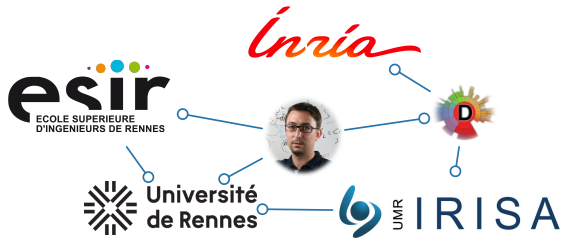
VALIDATION & VERIFICATION

GENERAL INTRODUCTION

UNIVERSITY OF RENNES, ESIR, 2023-2024

BENOIT COMBEMALE
FULL PROFESSOR, UNIVERSITY OF RENNES, FRANCE

HTTP://COMBEMALE.FR
BENOIT.COMBEMALE@IRISA.FR
@BCOMBEMALE



WHAT ARE WE LOOKING FOR?

We look for bugs

source: http://www.itpro.co.uk/security/20050/microsoft-pay-out-bounties-windows-81-hunters

Local bugs

- Some bugs are very local
  - redundant code
  - wrong condition
  - omission
  - lack of checks
  - divide by zero
  - approximations

Zune bug

```
while (days > 365)
    {
        if (IsLeapYear(year))
        {
            if (days > 366)
            {
                days -= 366;
                year += 1;
            }
        }
        else
        {
            days -= 365;
            year += 1;
        }
    }
```

(days >= 366)

source: http://pastie.org/349916#; http://www.zuneboards.com/forums/showthread.php?t=38143

Zune bug

- Zune 30 was the first portable media player released by Microsoft
  - release date: november 2006
- On Dec 31, 2008 all Zune stop working
- Software bug in the firmware: infinite loop when dealing with leap years
- Huge loss of business

Heartbleed bug

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

            err = sslRawVerify(ctx,
                               ctx->peerPubKey,
                               dataToSign,                                /* plaintex
                               dataToSignLen,                        /* plaintext length *
                               signature,
                               signatureLen);
            if(err) {
                        sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                        "returned %d\n", (int)err);
                        goto fail;
            }

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

source: http://opensource.apple.com/source/Security/Security-55471/libsecurity_ssl/lib/sslKeyExchange.c

Heartbleed bug

```
53  -     if ((err = ReadyHash(&SSLHashSHA1, &hashCtx, ctx)) != 0)
54  +     if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
55            goto fail;
56        if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
57            goto fail;
58        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
59            goto fail;
60        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
61            goto fail;
62  +         goto fail;
63        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
64            goto fail;
65
66           err = sslRawVerify(ctx,
67                              ctx->peerPubKey,
68                              dataToSign,                             /* plaintext */
69                              dataToSignLen,                  /* plaintext length */
70                              signature,
71                              signatureLen);
72           if(err) {
73                   sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
74                        "returned %d\n", (int)err);
75                   goto fail;
76           }
77
78    fail:
79  -     SSLFreeBuffer(&signedHashes, ctx);
80  -     SSLFreeBuffer(&hashCtx, ctx);
81  +     SSLFreeBuffer(&signedHashes);
82  +     SSLFreeBuffer(&hashCtx);
83        return err;
```

source: https://gist.github.com/hongrich/9176925

Heartbleed bug

- Bug introduced March 2012
- Bug revealed in April 2014
- Without using any privileged information, it is possible to retrieve
  - secret keys used for X.509 certificates
  - user names and passwords
  - instant messages
  - emails and business critical documents

More local bugs examples

USS Yorktown (1998) :
Division par zéro stoppa les moteurs

Mars Climate Orbiter (1998) :
Comparaison de valeurs dans des unités de mesure différentes

Bug de l'an 2000 :
Format date en 2 chiffres au lieu de 4

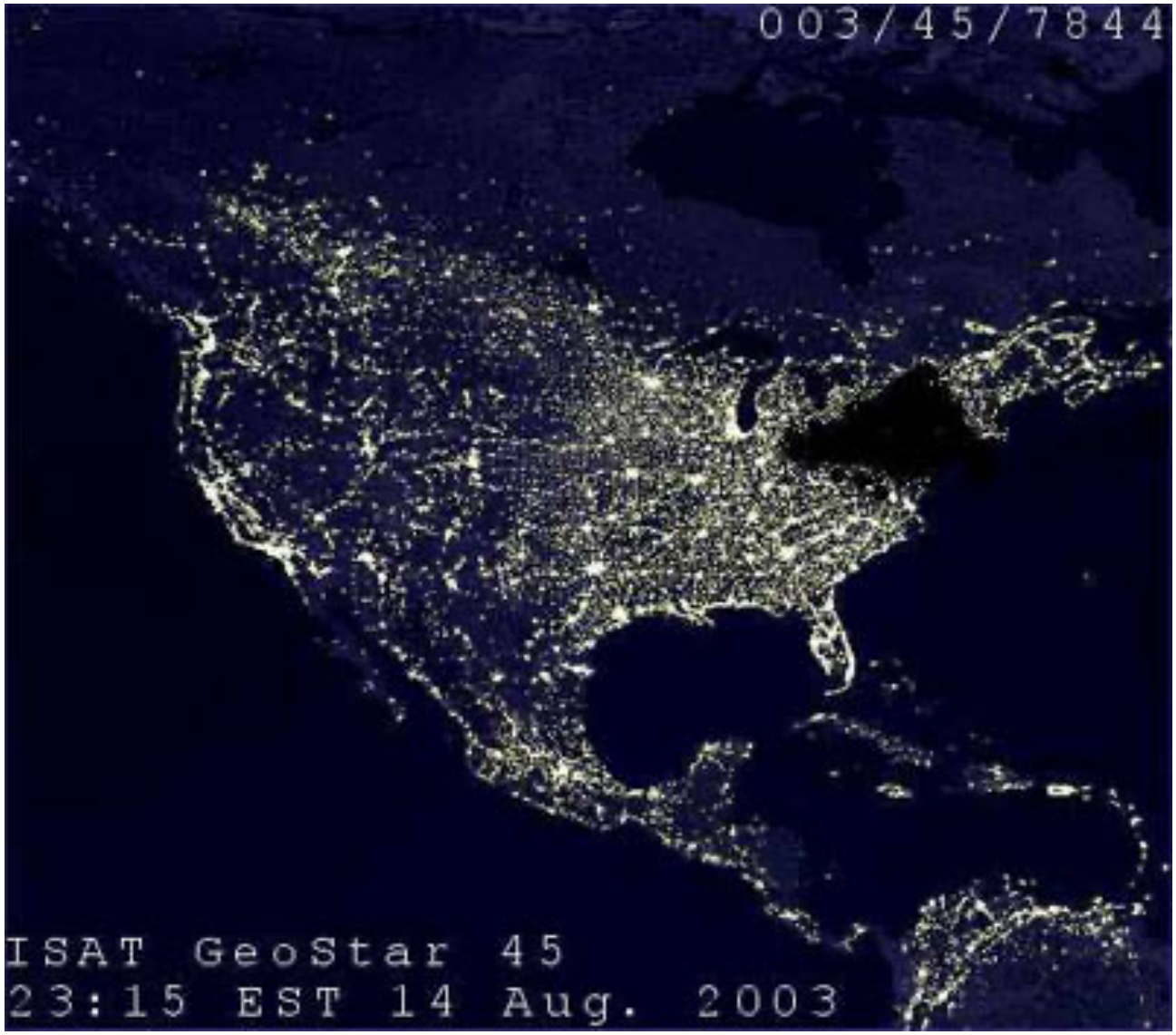
Global bugs

- Some bugs emerge from interactions
  - wrong assumptions about third parties
  - error in reuse
  - concurrency bugs
  - hardware/software/user improbable interactions

Northeast blackout of 2003

- Root cause of the outage was linked to a variety of factors, including FirstEnergy's failure to trim back trees encroaching on high-voltage power line
- Software bug in the alarm system at a control room of the FirstEnergy corp.
  - When triggered, race condition caused alarm system to stall for over an hour
  - backup server kicked in, it could not keep up with unprocessed data
  - warnings and alarms were not sounded because the systems were struggling to process old data.
  - employees did not take action
  - black-out spread to a huge region

Northeast blackout of 2003



Northeast blackout of 2003

- Widespread power outage on Aug 14, 2003
- Affected an estimated 10 million people in Ontario and 45 million people in eight U.S. states.

source: https://en.wikipedia.org/wiki/Northeast_blackout_of_2003#Computer_failure
http://www.securityfocus.com/news/8016

Race condition

```
public class SimpleApplet extends java.applet.Applet{
    java.awt.Image art;

    public void init() {
        art = getImage(getDocumentBase(),
                                    getParameter("img"));
    }

    public void paint(java.awt.Graphics g) {
        g.drawImage(art, 0, 0, this);
    }
}
```

an Applet's `paint()` method can be called before its `init()` method.

Check input

```
public class SimpleApplet extends java.applet.Applet{
    java.awt.Image art;

    public void init() {
        art = getImage(getDocumentBase(),
                                                    getPa
    }

    public void paint(java.awt.Graphics g) {
        if (art!=null){
                g.drawImage(art, 0, 0, this);
            }
    }
}
```

Ariane 501

- H0 -> H0+37s : nominal
- Dans SRI 2 (Inertial Reference System) :
  - BH (Bias Horizontal) > 2^15
  - convert_double_to_int(BH) fails!
  - exception SRI -> crash SRI2 & 1
- OBC (On-Board Computer) disoriented
  - Angle attaque > 20°,
  - charges aérodynamiques élevées
  - Séparation des boosters

Ariane 501

- H0 + 39s: auto-destruction (coût: 500M€)

Why? (cf. Jézéquel et al., *IEEE Comp. 01/97*)

- Ariane 5 reused a component from Ariane 4, which had an implicit assumption!
  - Assumes a constraint on input domain
    - Précondition : abs(BH) < 32768.0
  - OK for Ariane 4 but not Ariane 5
- *Need to specify exact contracts*

Log4Shell (Nov. 2021)

- Takes advantage of Log4j's allowing requests to arbitrary LDAP and JNDI servers
- Allows attackers to execute arbitrary Java code on a server or other computer, or leak sensitive information
- Existed unnoticed since 2013

https://www.cve.org/CVERecord?id=CVE-2021-44228

More global bug examples

- London Ambulance System (1992) – delays in medical emergencies
  - bad data checks, memory leaks, GUI issues, bad HW reuse, etc.
- Mars orbiter (1999)
  - Comparing inches with meters makes the probe crash on landing
- Orange (2012)
  - bug in the replicated, brand new HLR, no alarm triggered
- Facebook IPO glitch (2012)
  - race condition

The CWE top 25

Below is a list of the weaknesses in the 2022 CWE Top 25, including the overall score of each. The KEV Count (CVEs) shows the number of CVE-2020/CVE-2021 Records from the CISA KEV list that were mapped to the given weakness.

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

https://cwe.mitre.org/top25

Even more global bugs

- Therac-25 (official report)
  - The software code was not independently reviewed.
  - The software design was not documented with enough detail to support reliability modelling.
  - The system documentation did not adequately explain error codes.
  - AECL personnel were at first dismissive of complaints.
  - The design did not have any hardware interlocks to prevent the electron-beam from operating in its high-energy mode without the target in place.
  - Software from older models had been reused without properly considering the hardware differences.
  - The software assumed that sensors always worked correctly, since there was no way to verify them. (see open loop)
  - Arithmetic overflows could cause the software to bypass safety checks.
  - The software was written in assembly language. While this was more common at the time than it is today, assembly language is harder to debug than high-level languages.
  - ….

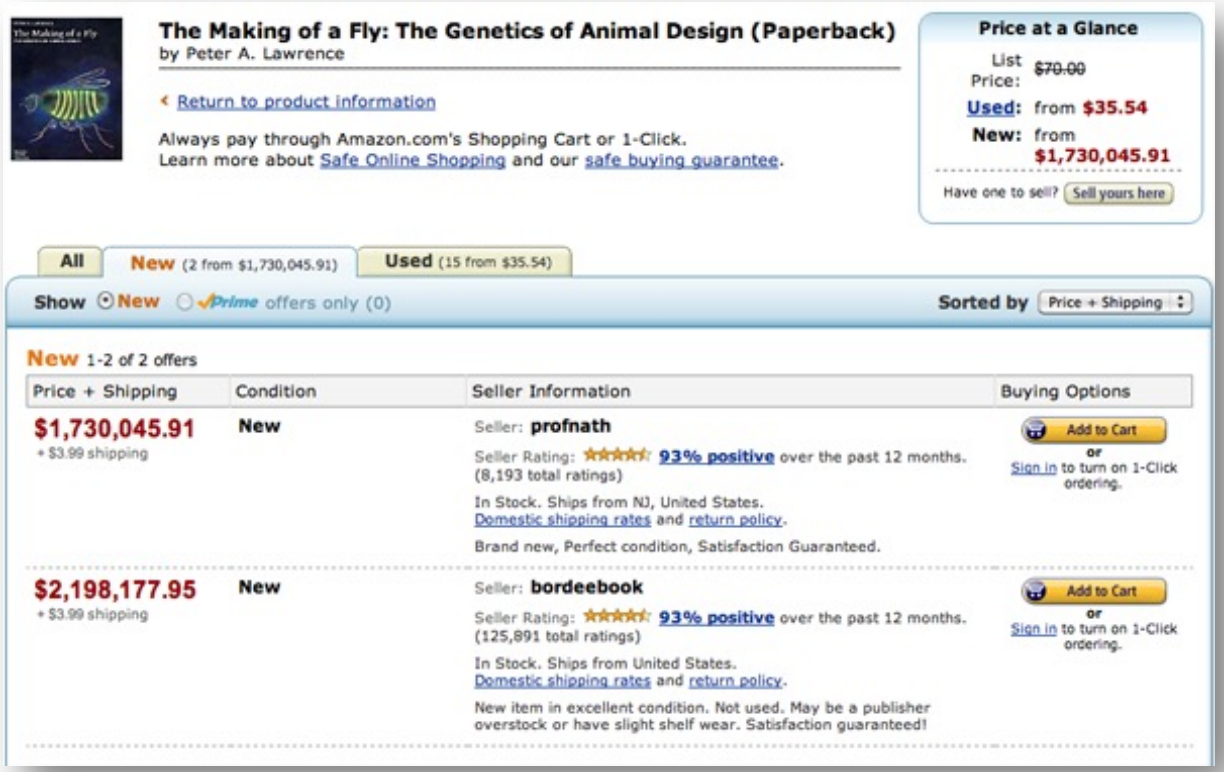
Even more global bugs

- Système d'information du FBI
  - abandonné en avril 2005 : coût 170 M $
  - mauvaise spécification, exigences mal exprimées
  - réutilisation dans un contexte inadapté
  - trop d'acteurs concurrents (hommes politiques, agents secrets, informaticiens)

Software fails

- Multiple causes
  - various sources
  - various levels
  - various reasons
- True for every domain
- Has all sorts of consequences

Amazon's $23,698,655.93 book about flies



The Making of a Fly: The Genetics of Animal Design (Paperback)
by Peter A. Lawrence

Return to product information

Always pay through Amazon.com's Shopping Cart or 1-Click.
Learn more about Safe Online Shopping and our safe buying guarantee.

Price at a Glance

List Price: $70.00
Used: from $35.54
New: from $1,730,045.91

Have one to sell? Sell yours here

All | New (2 from $1,730,045.91) | Used (15 from $35.54)

Show: New | Prime offers only (0) — Sorted by: Price + Shipping

New 1-2 of 2 offers

Price + Shipping	Condition	Seller Information	Buying Options
$1,730,045.91 + $3.99 shipping	New	Seller: profnath Seller Rating: 93% positive over the past 12 months. (8,193 total ratings) In Stock. Ships from NJ, United States. Domestic shipping rates and return policy. Brand new, Perfect condition, Satisfaction Guaranteed.	Add to Cart or Sign in to turn on 1-Click ordering.
$2,198,177.95 + $3.99 shipping	New	Seller: bordeebook Seller Rating: 93% positive over the past 12 months. (125,891 total ratings) In Stock. Ships from United States. Domestic shipping rates and return policy. New item in excellent condition. Not used. May be a publisher overstock or have slight shelf wear. Satisfaction guaranteed!	Add to Cart or Sign in to turn on 1-Click ordering.

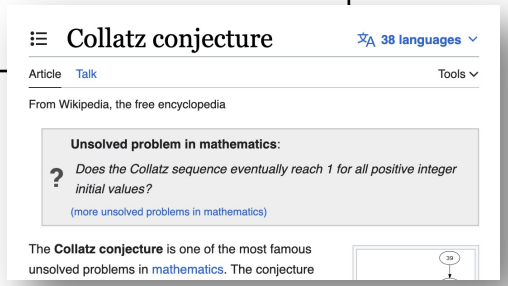
- Algorithmic pricing:

  Once a day profnath set their price to be 0.9983 times bordeebook's price, then bordeebook "noticed" profnath's change and elevated their price to 1.270589 times profnath's higher price.

WHY IS IT SO HARD TO BUILD CORRECT SOFTWARE?

Programming-in-the-small

```
Acquérir une valeur positive n
Tant que n > 1 faire
        si n est pair
        alors n := n / 2
        sinon n := 3n+1
Sonner alarme;
```

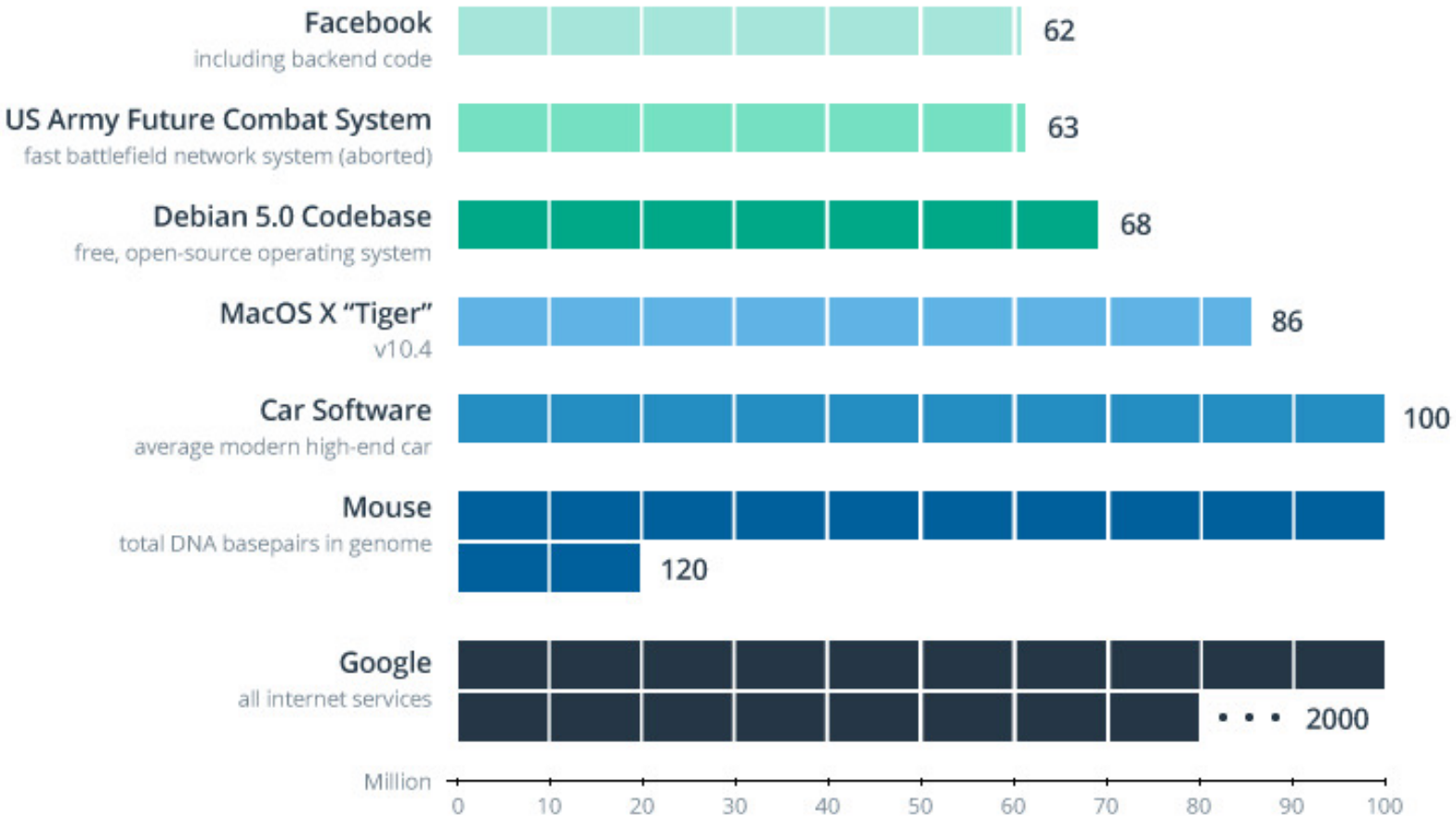


- Prouver que l'alarme est sonnée pour tout n?
- Indécidabilité de certaines propriétés
  - problème de l'arrêt de la machine de Turing...

➢ Recours au test

- ici, si machine 32 bits, 2^31 = 10^10 cas de tests
- 5 lignes de code => *10 milliards de valeurs* !

Programming-in-the-large



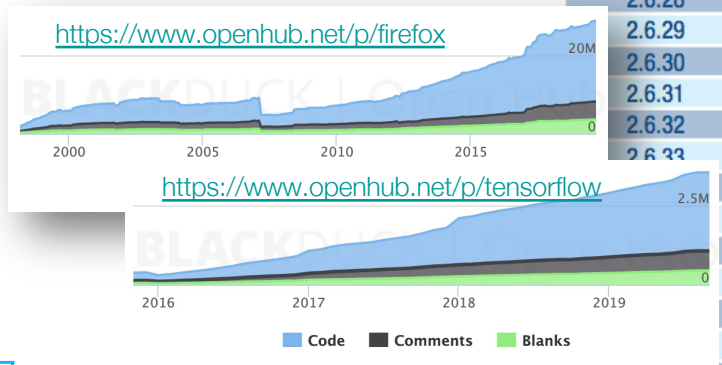
See https://informationisbeautiful.net/visualizations/million-lines-of-code/

Programming-in-the-large

- “Windows XP is compiled from 45 million lines of code.”

  See http://windows.microsoft.com/en-US/windows/history

- Example*:
  - Linux Kernel 2.6.17 - 4,142,481
  - Firefox 1.5.0.2 - 2,172,520
  - MySQL 5.0.25 - 894,768
  - PHP 5.1.6 - 479,892
  - Apache Http 2.0.x - 89,967

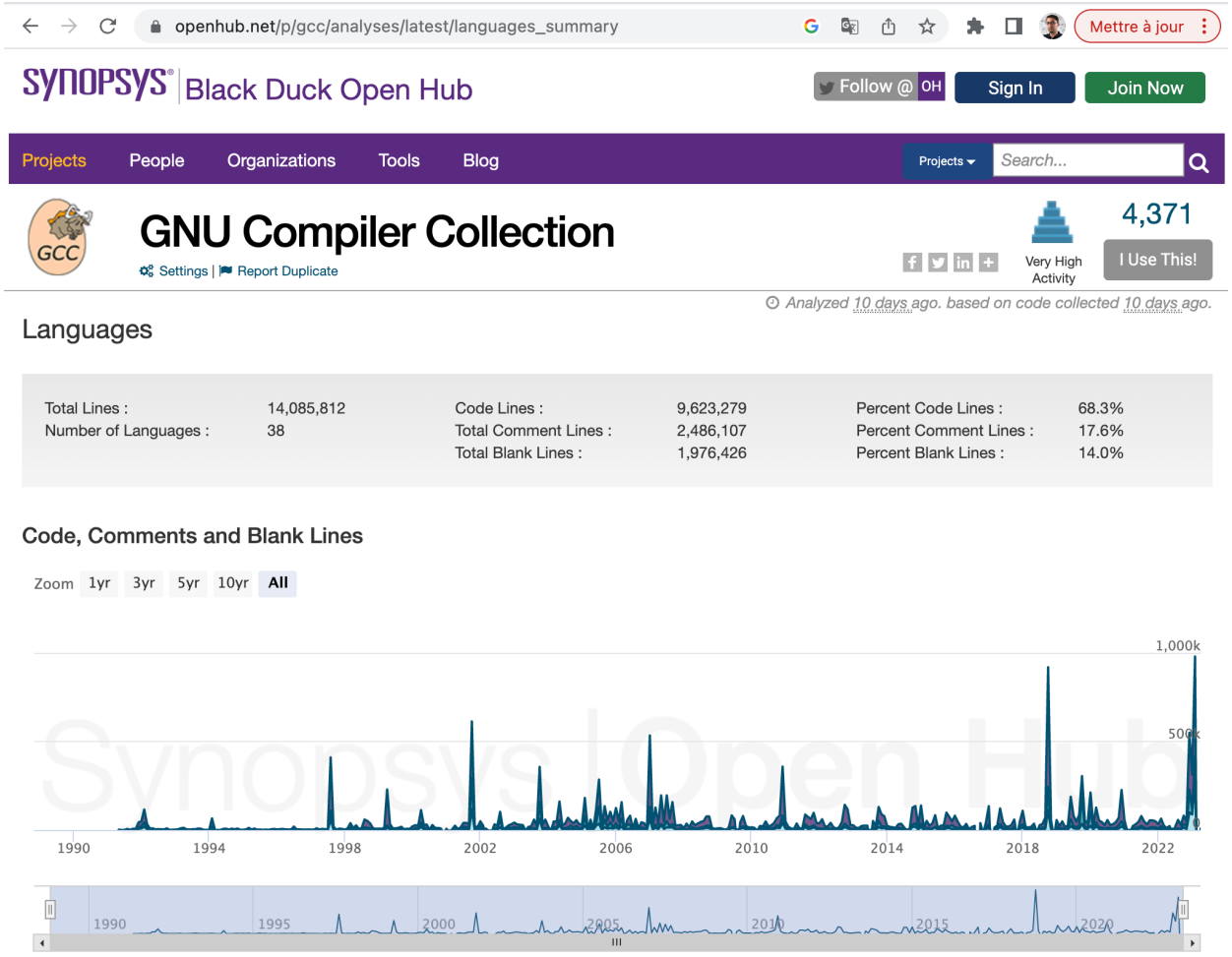


Kernel Version	Files	Lines
2.6.11	17,090	6,624,076
2.6.12	17,360	6,777,860
2.6.13	18,090	6,988,800
2.6.14	18,434	7,143,233
2.6.15	18,811	7,290,070
2.6.16	19,251	7,480,062
2.6.17	19,553	7,588,014
2.6.18	20,208	7,752,846
2.6.19	20,936	7,976,221
2.6.20	21,280	8,102,533
2.6.21	21,614	8,246,517
2.6.22	22,411	8,499,410
2.6.23	22,530	8,566,606
2.6.24	23,062	8,859,683
2.6.25	23,813	9,232,592
2.6.26	24,273	9,411,841
2.6.27	24,356	9,630,074
2.6.28	25,276	10,118,757
2.6.29	26,702	10,934,554
2.6.30	27,911	11,560,971
2.6.31	29,143	11,970,124
2.6.32	30,504	12,532,677
2.6.33	31,584	12,912,684
2.6.34	32,316	13,243,582
2.6.35	33,335	13,468,253
2.6.36	34,317	13,422,037
2.6.37	36,189	13,919,579
2.6.38	36,868	14,211,814
2.6.39	36,713	14,537,764
3.0	36,788	14,651,135
3.1	37,095	14,776,002
3.2	37,626	15,004,006

Greg Kroah-Hartman, Jonathan Corbet, Amanda McPherson. "Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It" (March 2012). The Linux Foundation. Retrieved 2012-04-10.

* *eLOC (effective line of code)* is the measurement of all lines that are not comments, blanks or standalone braces or parenthesis (see http://msquaredtechnologies.com/m2rsm/rsm_software_project_metrics.htm)

Programming-in-the-large



See https://www.openhub.net/p/gcc. Retrieved 2023-05-01.

Programming-in-the-large

But also…

Programming-in-the-large

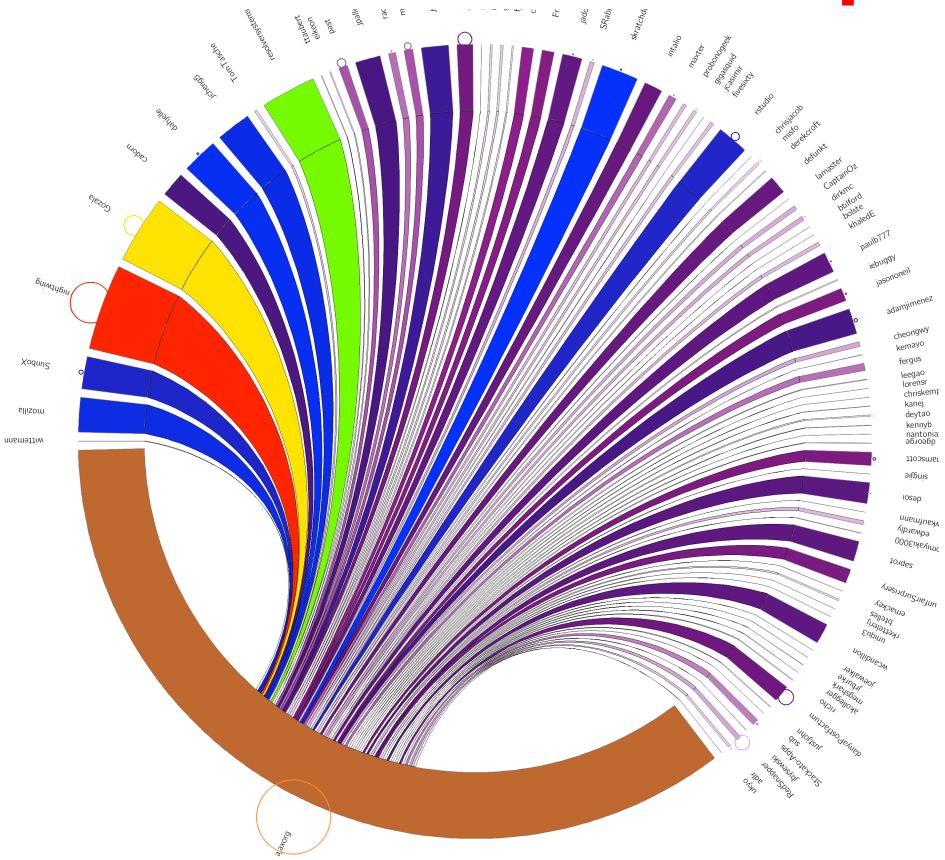
Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
C++	3,460,458	869,918	20.1%	739,205	5,069,581	36.0%
C	2,389,131	684,958	22.3%	475,972	3,550,061	25.2%
Ada	831,230	396,568	32.3%	290,062	1,517,860	10.8%
Go	793,870	169,011	17.6%	101,033	1,063,914	7.6%
D	675,619	157,575	18.9%	116,462	949,656	6.7%
Autoconf	531,266	2,363	0.4%	82,745	616,374	4.4%
Fortran (Free-format)	239,471	71,199	22.9%	45,037	355,707	2.5%
Modula-2	151,218	77,965	34.0%	51,802	280,985	2.0%
HTML	128,638	778	0.6%	12,645	142,061	1.0%
Make	121,979	4,616	3.6%	14,574	141,169	1.0%
Assembly	77,779	20,729	21.0%	13,448	111,956	0.8%
XML	65,934	550	0.8%	6,331	72,815	0.5%
shell script	39,165	8,469	17.8%	5,849	53,483	0.4%
Objective-C	28,618	5,603	16.4%	8,369	42,590	0.3%
Fortran (Fixed-format)	23,571	2,080	8.1%	1,992	27,643	0.2%
Python	11,584	3,633	23.9%	2,798	18,015	0.1%
Rust	10,497	1,004	8.7%	2,317	13,818	0.1%
Automake	9,420	1,610	14.6%	1,724	12,754	0.1%
Postscript	8,674	157	1.8%	234	9,065	0.1%
TeX/LaTeX	8,668	3,611	29.4%	1,053	13,332	0.1%
Perl	6,122	1,840	23.1%	1,074	9,036	0.1%
AWK	4,833	785	14.0%	661	6,279	0.0%
Pascal	1,083	141	11.5%	219	1,443	0.0%
Mathematica	1,061	47	4.2%	242	1,350	0.0%
C#	879	506	36.5%	230	1,615	0.0%
DCL	756	162	17.6%	12	930	0.0%
JavaScript	490	21	4.1%	90	601	0.0%
CMake	310	44	12.4%	59	413	0.0%
OCaml	285	29	9.2%	44	358	0.0%
Haskell	208	0	0.0%	27	235	0.0%
CSS	203	42	17.1%	48	293	0.0%
Vim Script	193	71	26.9%	48	312	0.0%
AMPL	34	0	0.0%	9	43	0.0%
Brainfuck	10	4	28.6%	3	17	0.0%
Emacs Lisp	7	12	63.2%	4	23	0.0%
XSL Transformation	6	6	50.0%	4	16	0.0%
Matlab	5	0	0.0%	0	5	0.0%
DOS batch script	4	0	0.0%	0	4	0.0%
Totals	9,623,279	2,486,107		1,976,426	14,085,812	

- Interoperability

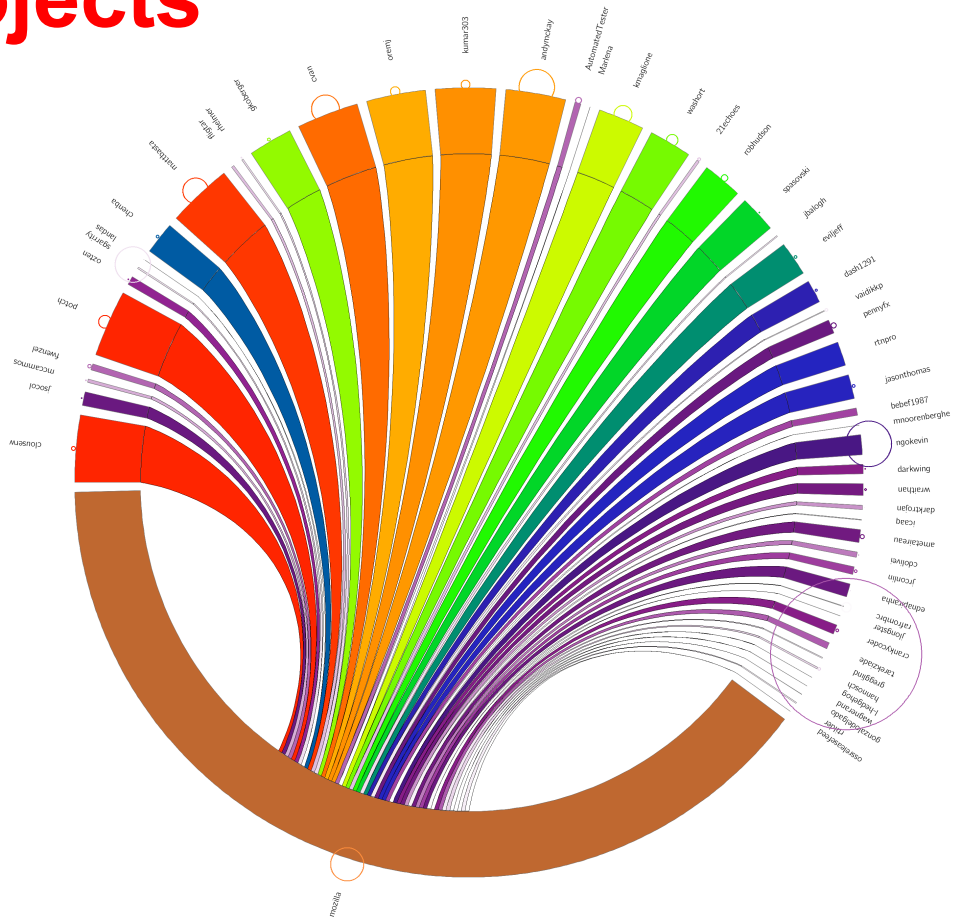
See https://www.openhub.net/p/gcc. Retrieved 2023-05-01.

Programming-in-the-large

- Collaborative projects



137 contributors, 5000 commits, 1300 forks
https://github.com/ajaxorg/ace



97 contributors, 10000+ commits, 173 forks
https://github.com/mozilla/zamboni

Programming-in-the-large

- **Reusability**
- **Durability**
- **Variability**

Programming-in-the-large

- **Critical**
- **Real-time**
- **Embedded**

Phaeton

- 61 networked ECUs
- 3 bus systems + optical bus + sub busses
- 2500 signals in 250 CAN-messages
- more than 50 MByte memory
- more than 2000 individual wires
- more than 3800m cables

Herbert Diess
@Herbert_Diess · **Follow**

Software will be THE differentiator to decide how successful Volkswagen will be in the #NEWAUTO-world!

9:12 AM · Dec 10, 2021

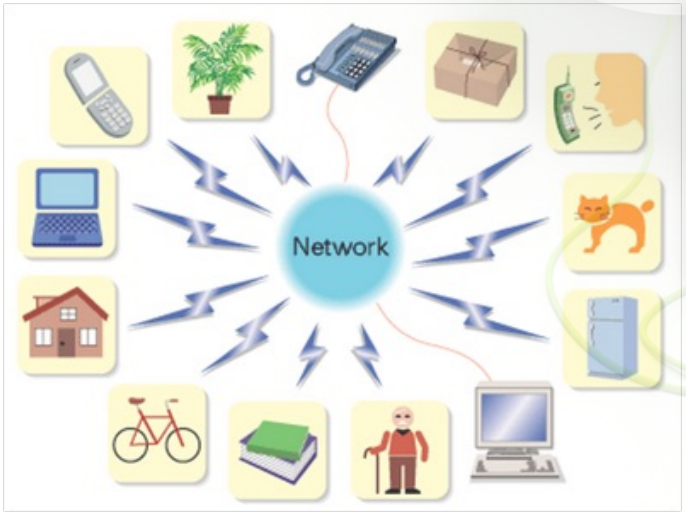
Programming-in-the-large

- ***"The avionics system in the F-22 Raptor […] consists of about 1.7 million lines of software code."***
- ***"F-35 Joint Strike Fighter […] will require about 5.7 million lines of code to operate its onboard systems."***
- ***"Boeing's new 787 Dreamliner […] requires about 6.5 million lines of software code to operate its avionics and onboard support systems."***
- ***"if you bought a premium-class automobile recently, it probably contains close to 100 million lines of software code. […] All that software executes on 70 to 100 microprocessor-based electronic control units (ECUs) networked throughout the body of your car."***
- ***"Alfred Katzenbach, the director of information technology management at Daimler, has reportedly said that the radio and navigation system in the current S-class Mercedes-Benz requires over 20 million lines of code alone and that the car contains nearly as many ECUs as the new Airbus A380 (excluding the plane's in-flight entertainment system)."***
- **"IBM claims that approximately 50 percent of car warranty costs are now related to electronics and their embedded software"**

"*This Car Runs on Code*", By Robert N. Charrette, IEEE Spectrum, Feb. 2009, see http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code

Programming-in-the-large

- Autonomic Computing,
- Cloud Computing
- PaaS, SaaS, IoS, IoT, CPS...



Google's innovation factory

- Google, Building for Scale:
  - 6,000 developer / 1,500+ projects
  - Each product has custom release cycles
    - *few days to few weeks*
  - 1(!!) code repository
  - No binary releases
    - *everything builds from HEAD*
  - 20+ code changes per minute
    - *50% of the code changes every month*

- Distributed
- Large Scale

Source: http://googletesting.blogspot.com/search/label/Copeland

Software Engineering at Google. Fergus Henderson, 2019. https://arxiv.org/abs/1702.01715

Programming-in-the-Duration (maintenance)

- Etalement sur 10 ans ou plus d'une "ligne de produits"
- Près de 80 ans dans l'avionique !
- Age moyen d'un système : 7 ans
- 26% des systèmes ont plus de 10 ans

(Cf. Application banquaire et Cobol)

Long term availability…

AIRBUS A300 Life Cycle

Program began in 1972, production stopped in 2007

2007-1972 = 35 years...

Support will last until 2050

2050-1972 = 78 years !!

On board software development for very long lifecycle products

AIRBUS

Innovation factory

SCRUM

V-cycle

processes

CMMI

spiral

distributed

embedded

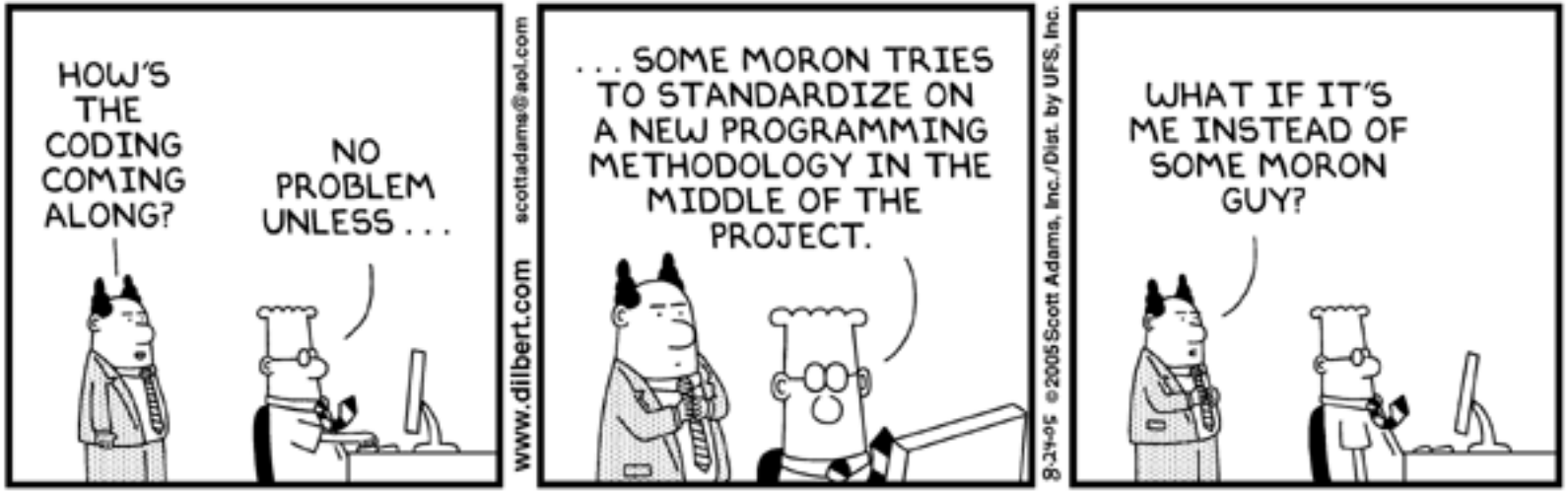
security

critical

constraints

scalability

end-user



cloud

lean

innovation

economics

SOA

outsourcing

agility

OOP

AOP

Procedural

CP

languages

UML

CBSE

DSL

end-user

DSML

A question of perspective

Verification:

"Are we building the product right"

The software should conform to its specification

Validation:

"Are we building the right product"

The software should do what the user really requires

A question of perspective

- Stakeholder
  - customer, developer, sales
- Qualitative
  - functionality, usability, safety-critical, etc.
- Application kind
  - embedded, adaptive, reactive, etc.

HOW TO BUILD RELIABLE SOFTWARE ?

Engineering reliable software

- Constructive approach
  - Formal modeling
  - Garantees by construction
- Analytical approach
  - Program analysis
  - Detect and fix errors
- Fault-tolerance
  - Admit the presence of errors
  - Enhance software with fault-tolerance mechanisms

Constructive approach

- Garantee the absence of bugs
- Top-down approach
- Model-driven development + formal analysis
- Formal proof
  - Automatic or manual
  - Offers exhaustive garantees based on logical modeling and reasoning
  - Examples: Isabelle/HOL, B, KeY, Coq
  - Used on specific parts of critical software (e.g., certified C compiler)

Constructive approach

- Model checking
  - Formal behavioral model (transition system)
  - Exhaustive verification of properties on model executions (e.g., absence of deadlock, safety and liveness properties)
  - Examples: SCADE, Java PathFinder
  - Used in hardware and software verification
    - at the ‘system’ level for systems engineering (defense, nuclear plant, transportation, etc.)

Analytical approach

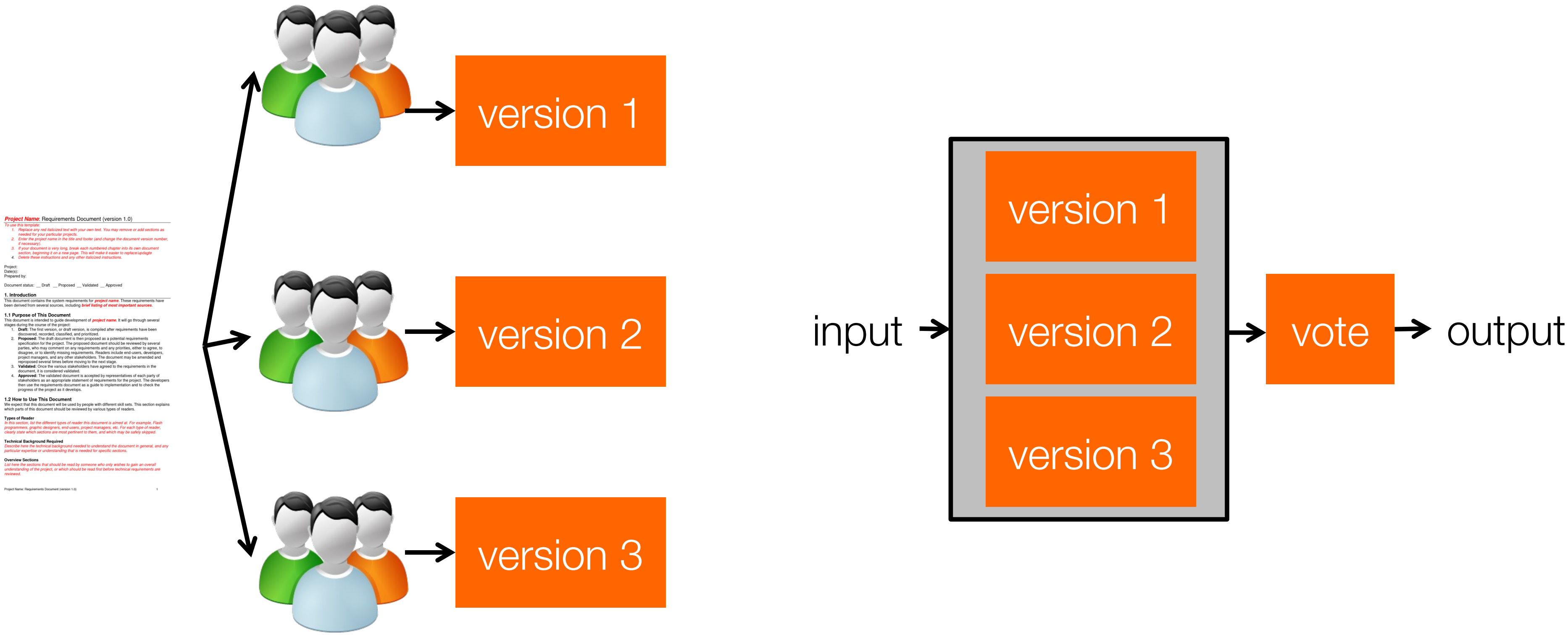
- Look for the presence of bugs
- Heuristic-based
- Analyze all sorts of software artefacts (code, models, requirements, etc.)
- Software testing

Fault-tolerance

- Assume that it is impossible to prevent the occurrence of bugs in production software
- Enhance the system with the ability to deal with it
- Approaches exist at all levels, e.g.:
  - Design diversity at the systems level
  - Exception handling at the source code level
  - Randomization at the machine code level

Fault-tolerance

- N-version programming



N-VERSION PROGRAMMINC: A FAULT-TOLERANCE APPROACH TO RELIABILITY OF SOFTWARE OPERATION.
Liming Chen, A. Avizienis, 25th International Symposium on Fault-Tolerant Computing, 1995.

Netflix's simian army

- Streaming TV network service
  - 200+ million subscribers
  - very high dependence on software and cloud (runs on Amazon EC2)
  - major player in open source
- Induce failure regularly
  - 'break' production code to check the system's ability to react
  - Chaos monkey: randomly terminates an instance in production
  - Chaos kong: take an entire region offline
  - Latency monkey: artificial delay in RESTful clients

Basiri, Ali, et al. "Chaos engineering." IEEE Software 33.3 (2016): 35-41.

Loop perforation: when good enough is better

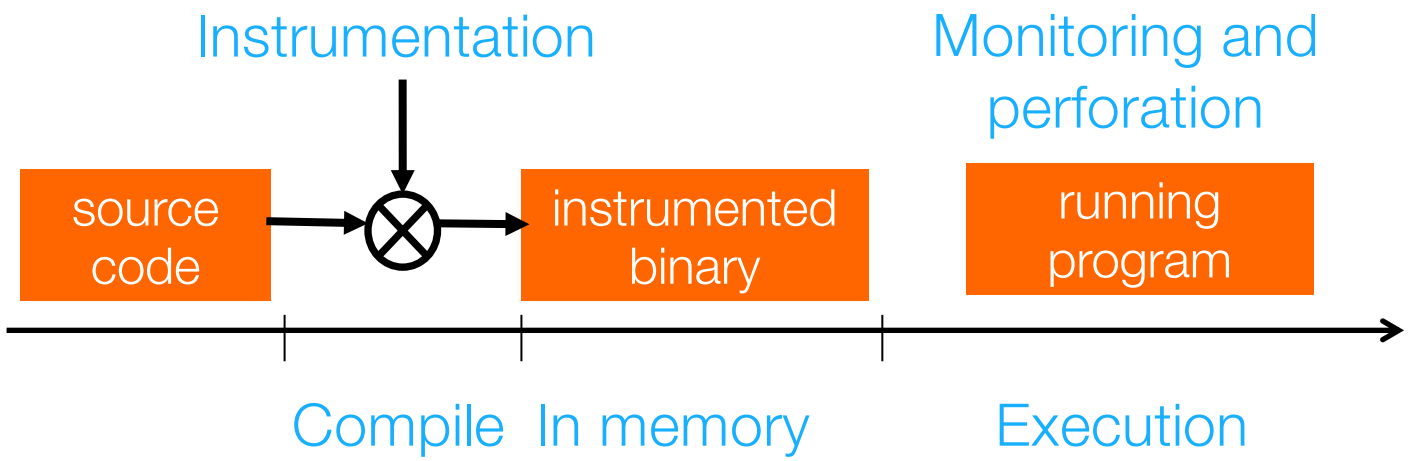
- *"It used to be that people used computers for computations where there was a single, hard, logical right answer"*
- Trade-off between accuracy and performance

```
for (i=0; i<b; i++){…}
```


```
for (i=0; i<b; i+=n){…}
```


source: http://newsoffice.mit.edu/2010/situationally-award-0513

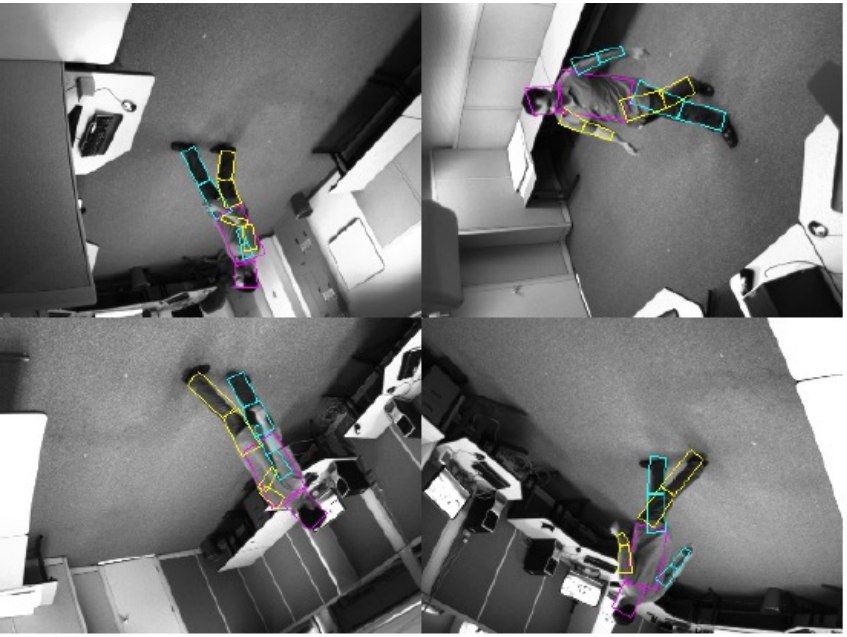
Loop perforation

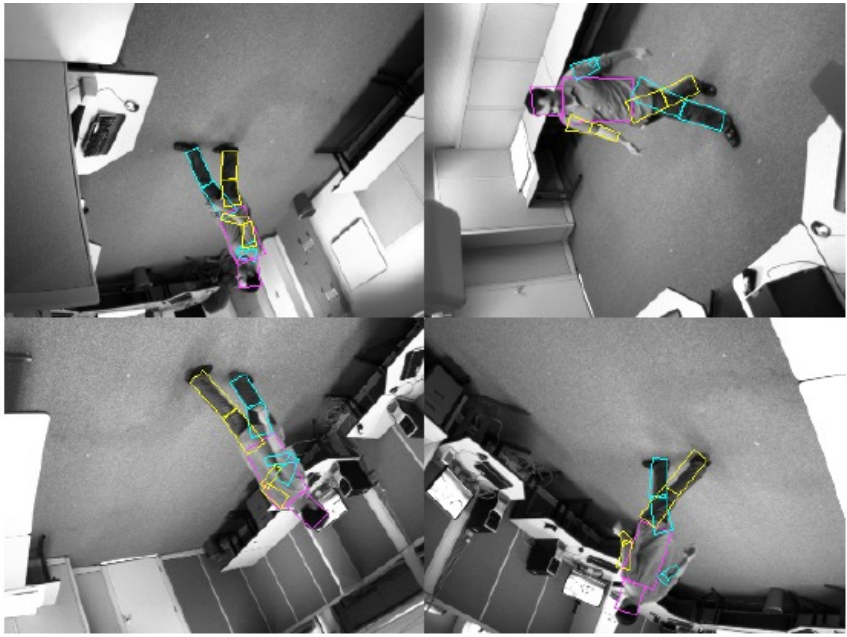


- unsound transformation
- still useful

Sasa Misailovic, Stelios Sidiroglou, Henry Hoffmann, Martin C. Rinard: Quality of service profiling. ICSE (1) 2010: 25-34

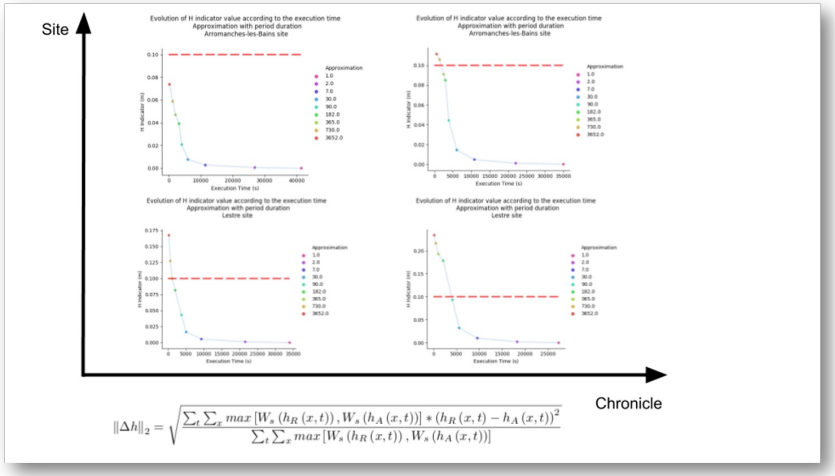
Loop perforation





Approximate Scientific Software

- Reduce the simulation time to better support trade-off analysis and decision making
- Application of approximate computing to scientific computing
- Work on the simulation code (white box) or the input data (black box)
- Require a domain-specific interpolation operator
- Challenge: infer the approximation bound



Loop Aggregation for Approximate Scientific Computing
June Sallou, Alexandre Gauvain, Johann Bourcier, Benoît Combemale, Jean-Raynald de Dreuzy. ICCS 2020: 141-155.

In this lecture

- Software testing
  - most probably the technique you have to use
- for verification
  - validation is essential but requires the involvement of users
- from the developer's perspective
  - you should test your software as developer, and this is an important part of existing positions.

Acknowledment

To make a long story short, all materials are shared since a long time with various friends and colleagues, and are the results of an awesome, along the way, collaborative effort!

Big thanks to, among others, **Benoit Baudry** (KTH, Sweden), **Yves LeTraon** (Univ. Luxembourg), **Jean-Marc Jézéquel** (Univ. Rennes, France) and **Oscar Luis Vera perez** (MediaKind, France).