

VALIDATION & VERIFICATION

GENERAL INTRODUCTION

MASTER 1 ICE, 2017-2018

BENOIT COMBEMALE
PROFESSOR, UNIV. TOULOUSE, FRANCE

[HTTP://COMBEMALE.FR](http://combemale.fr)
[BENOIT.COMBEMALE@IRIT.FR](mailto:benoit.combemale@irit.fr)
[@BCOMBEMALE](https://twitter.com/BCOMBEMALE)



WHAT ARE WE LOOKING FOR?

We look for bugs



Local bugs

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

Many bugs in conditions

1049	1049	<code>public static String chopNewline(String str) {</code>
1050	1050	<code> int lastIdx = str.length() - 1;</code>
1051		<code> if (lastIdx == 0) {</code>
	1051	<code> if (lastIdx <= 0) {</code>
1052	1052	<code> return "";</code>
1053	1053	<code> }</code>
1054	1054	<code> char last = str.charAt(lastIdx);</code>
1055	1055	<code> if (last == '\n') {</code>
1056	1056	<code> if (str.charAt(lastIdx - 1) == '\r') {</code>
1057	1057	<code> lastIdx--;</code>
1058	1058	<code> }</code>
1059	1059	<code> } else {</code>
1060	1060	<code> lastIdx++;</code>
1061	1061	<code> }</code>
1062	1062	<code> return str.substring(0, lastIdx);</code>
1063	1063	<code>}</code>
1064	1064	

Zune bug

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

(days >= 366)

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,
                  dataToSignLen,
                  signature,
                  signatureLen);
/* plaintext */
/* plaintext length */

if(err) {
    sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
               "returned %d\n", (int)err);
    goto fail;
}

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


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;
```

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 by zero stops the engine of the submarine
- Guiding system(2002)
 - Wrong initialization
- The Patriot and the Scud (1991)
 - approximation in subtraction

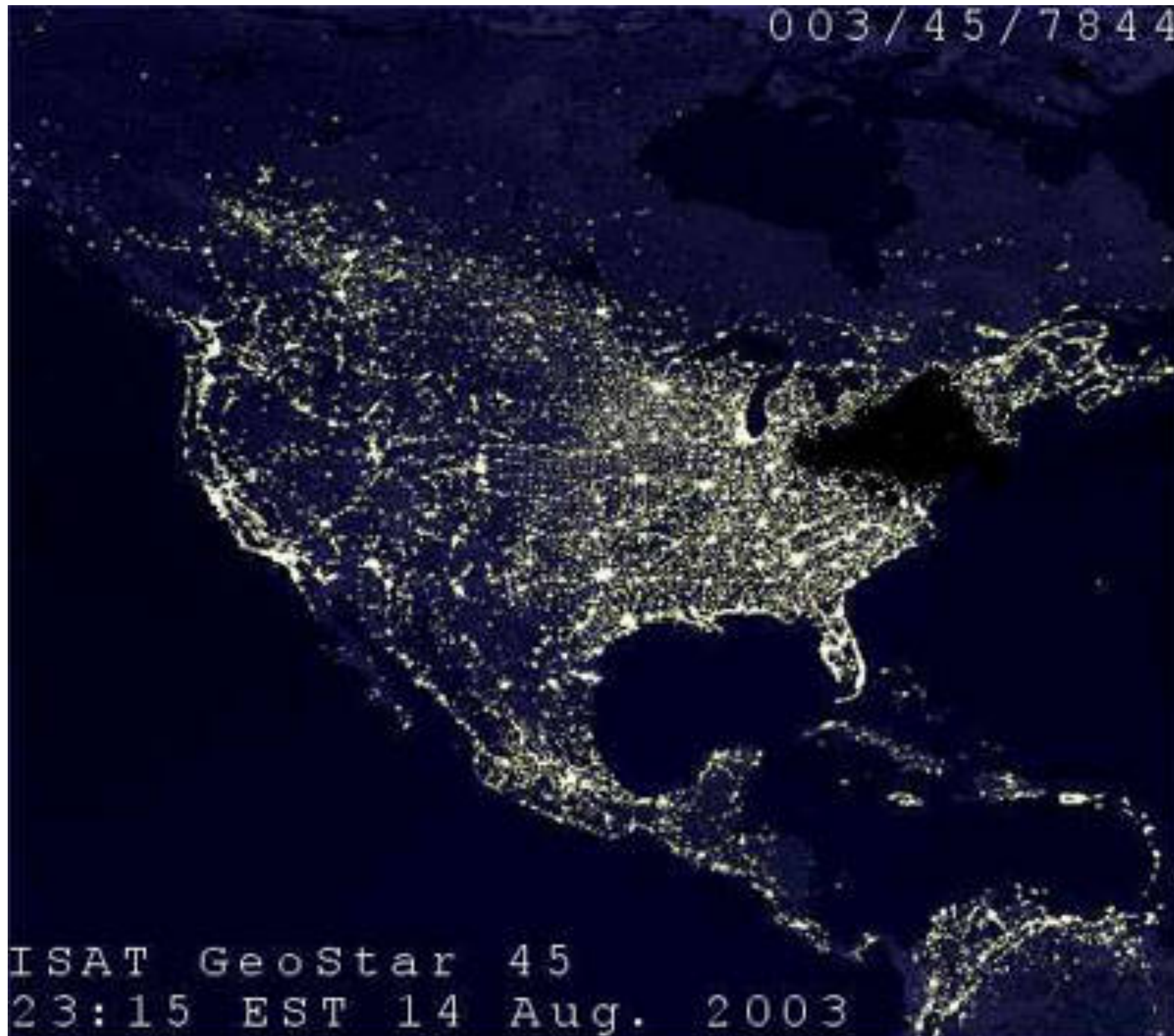
Global bugs

- 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.

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

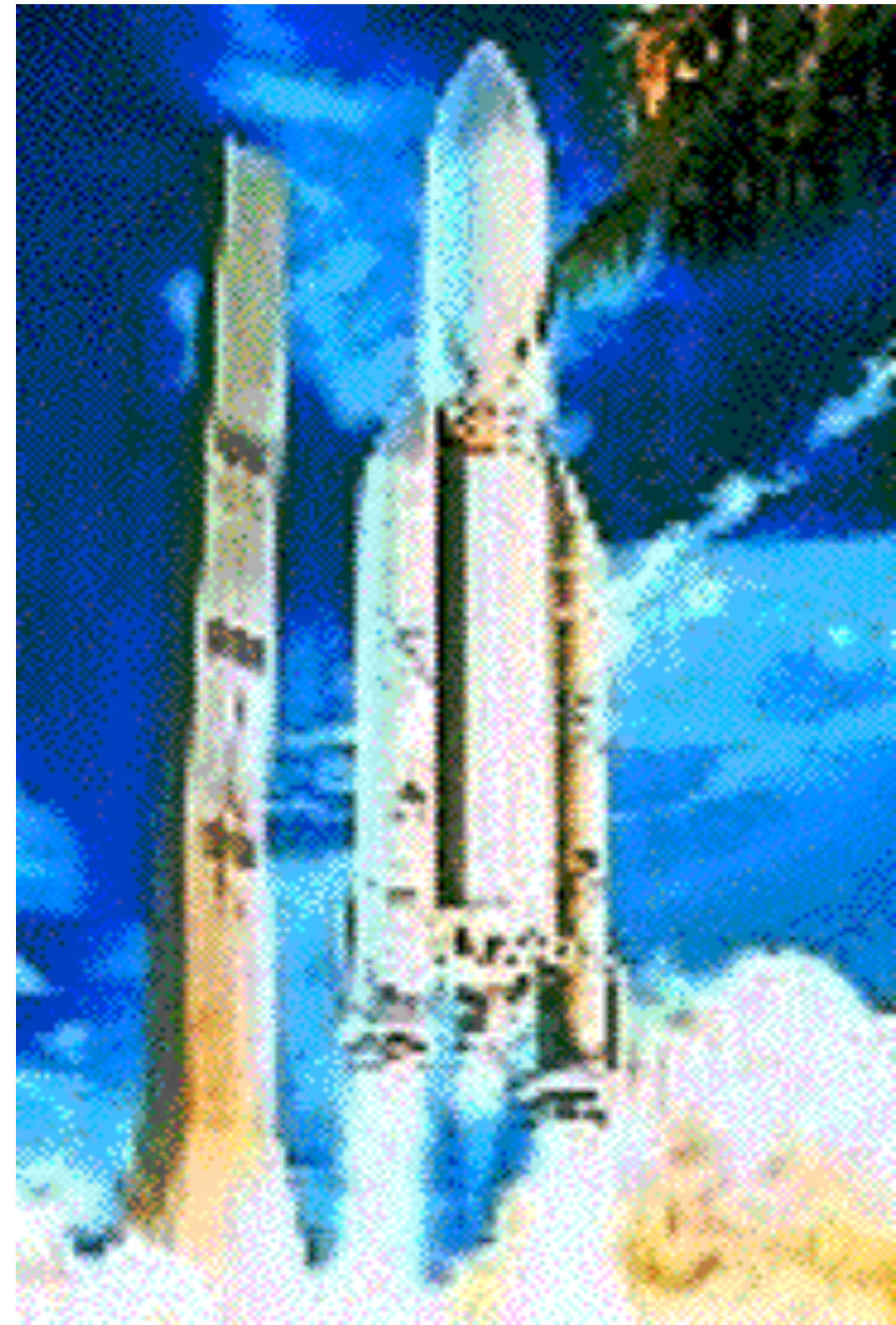
```
public class SimpleApplet extends java.applet.Applet{
    java.awt.Image art;

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

    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 $> 20^\circ$,
 - 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 : $\text{abs}(\text{BH}) < 32768.0$
 - OK for Ariane 4 but not Ariane 5
- *Need to specify exact contracts*

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

NIST top 25

Brief Listing of the Top 25

This is a brief listing of the Top 25 items, using the general ranking.

NOTE: 16 other weaknesses were considered for inclusion in the Top 25, but their general scores were not high enough. They are listed in a separate ["On the Cusp"](#) page.

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere
[17]	65.5	CWE-732	Incorrect Permission Assignment for Critical Resource
[18]	64.6	CWE-676	Use of Potentially Dangerous Function
[19]	64.1	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
[20]	62.4	CWE-131	Incorrect Calculation of Buffer Size
[21]	61.5	CWE-307	Improper Restriction of Excessive Authentication Attempts
[22]	61.1	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
[23]	61.0	CWE-134	Uncontrolled Format String
[24]	60.3	CWE-190	Integer Overflow or Wraparound
[25]	59.9	CWE-759	Use of a One-Way Hash without a Salt

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

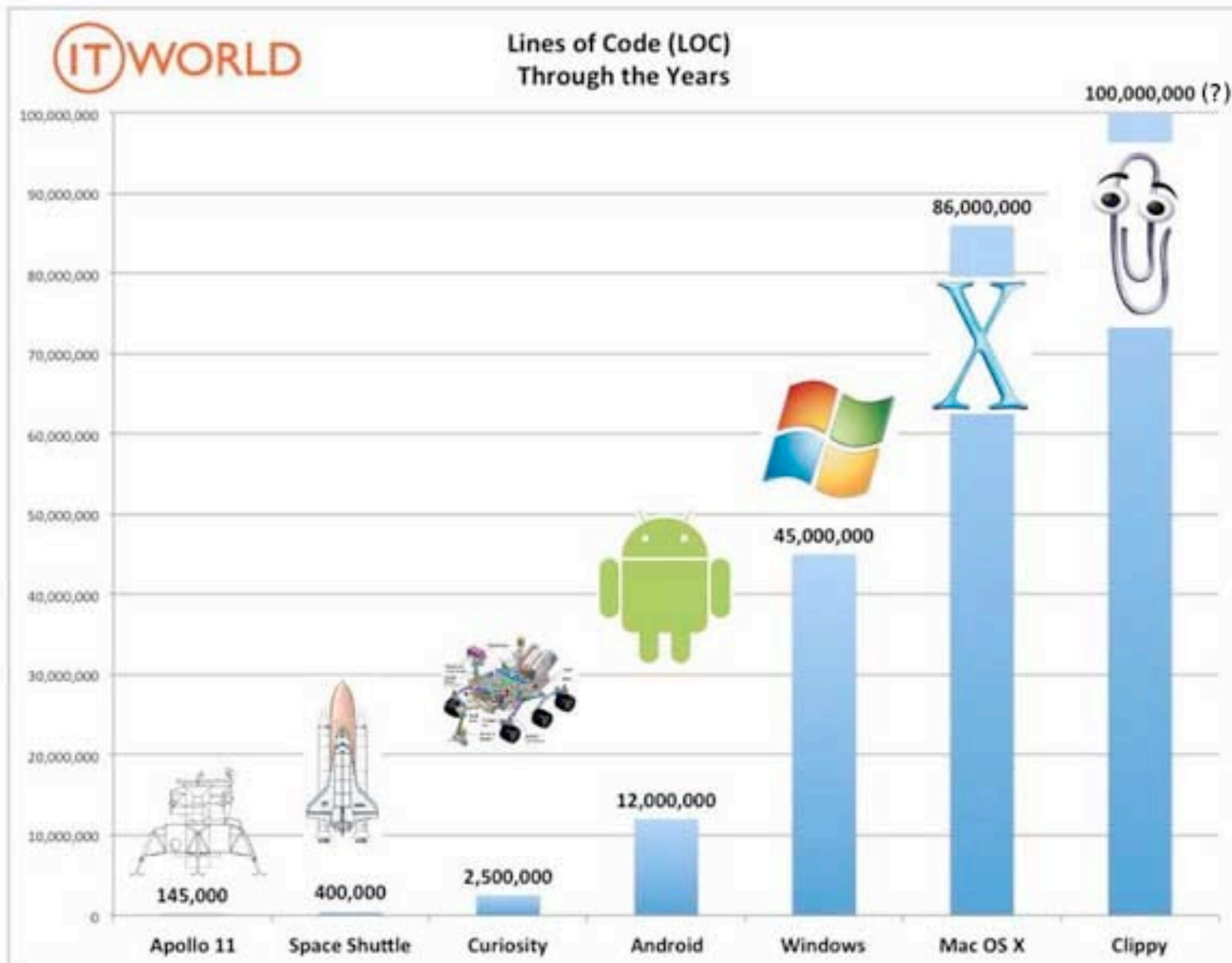
```
Acquérir une valeur positive n
Tant que n > 1 faire
    si n est pair
        alors n := n / 2
    sinon n := 3n+1
Sonner alarme;
```

- 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 <http://www.itworld.com/big-datahadoop/288893/lines-code-apollo-curiosity>

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

* 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)

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.

Programming-in-the-large

[Follow @Ohloh](#)[Sign In](#)[Join Now](#)[Projects](#)[People](#)[Tools](#)[Meta](#)Projects 

GNU Compiler Collection

[Settings](#) | [Report Duplicate](#)

3,756

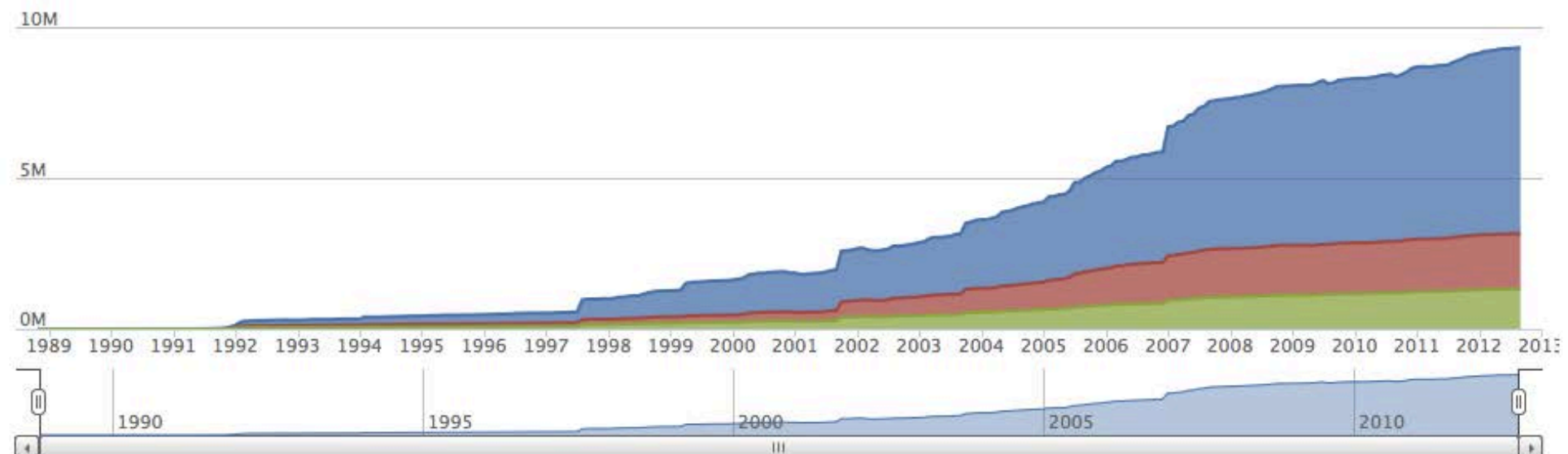
[I Use This!](#)

⌚ Analyzed about 12 hours ago based on code collected about 12 hours ago.

Languages

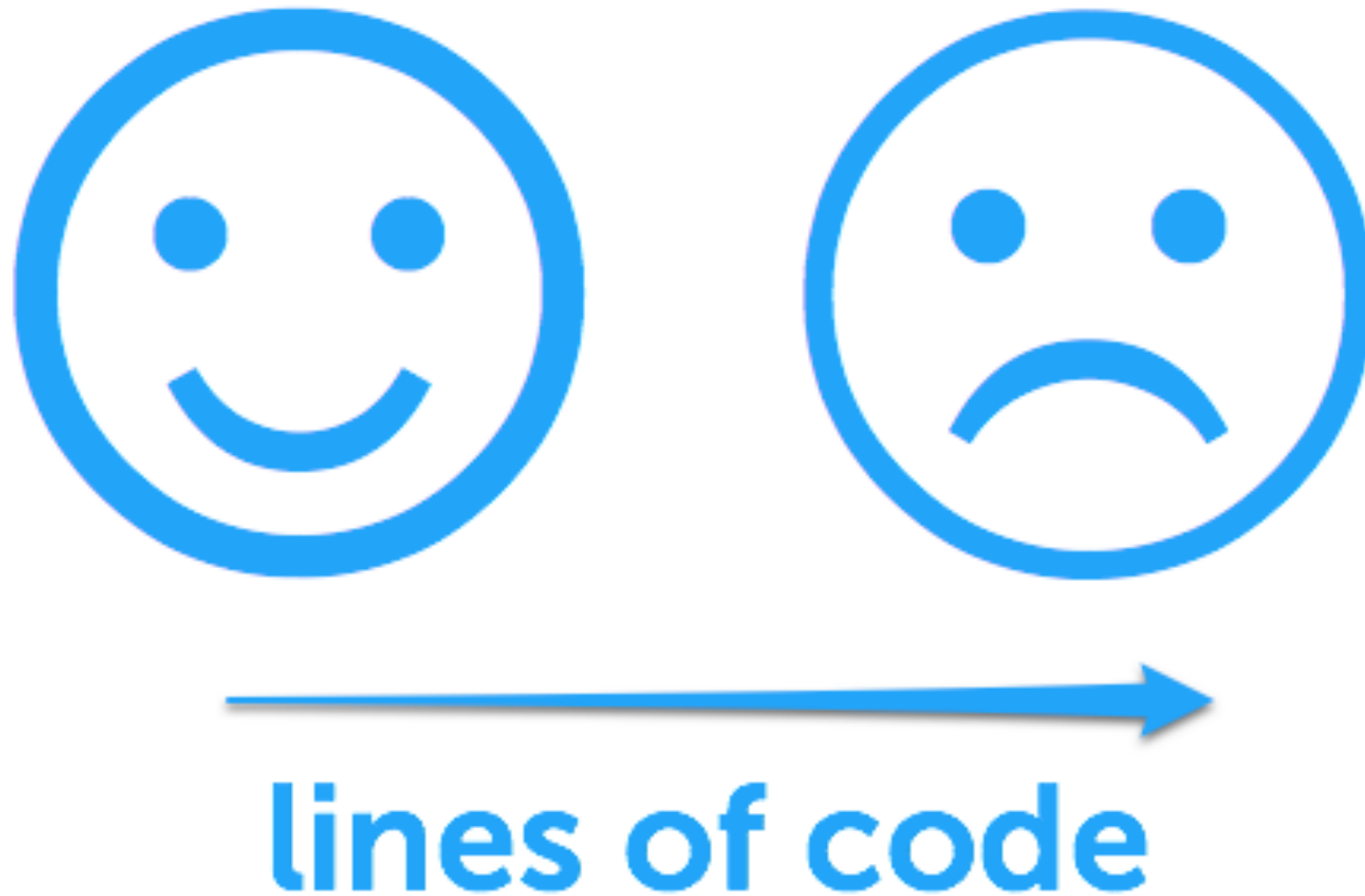
Total Lines :	9,337,085	Code Lines :	6,174,724	Percent Code Lines :	66.1%
Number of Languages :	33	Total Comment Lines :	1,832,058	Percent Comment Lines :	19.6%
		Total Blank Lines :	1,330,303	Percent Blank Lines :	14.2%

Code, Comments and Blank Lines









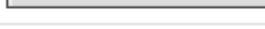








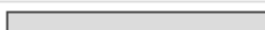





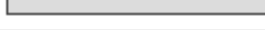


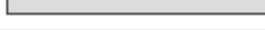



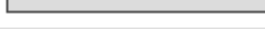


Zoom 

See <http://www.ohloh.net/p/gcc>. Retrieved 2012-09-16.

Programming-in-the-large



But also...

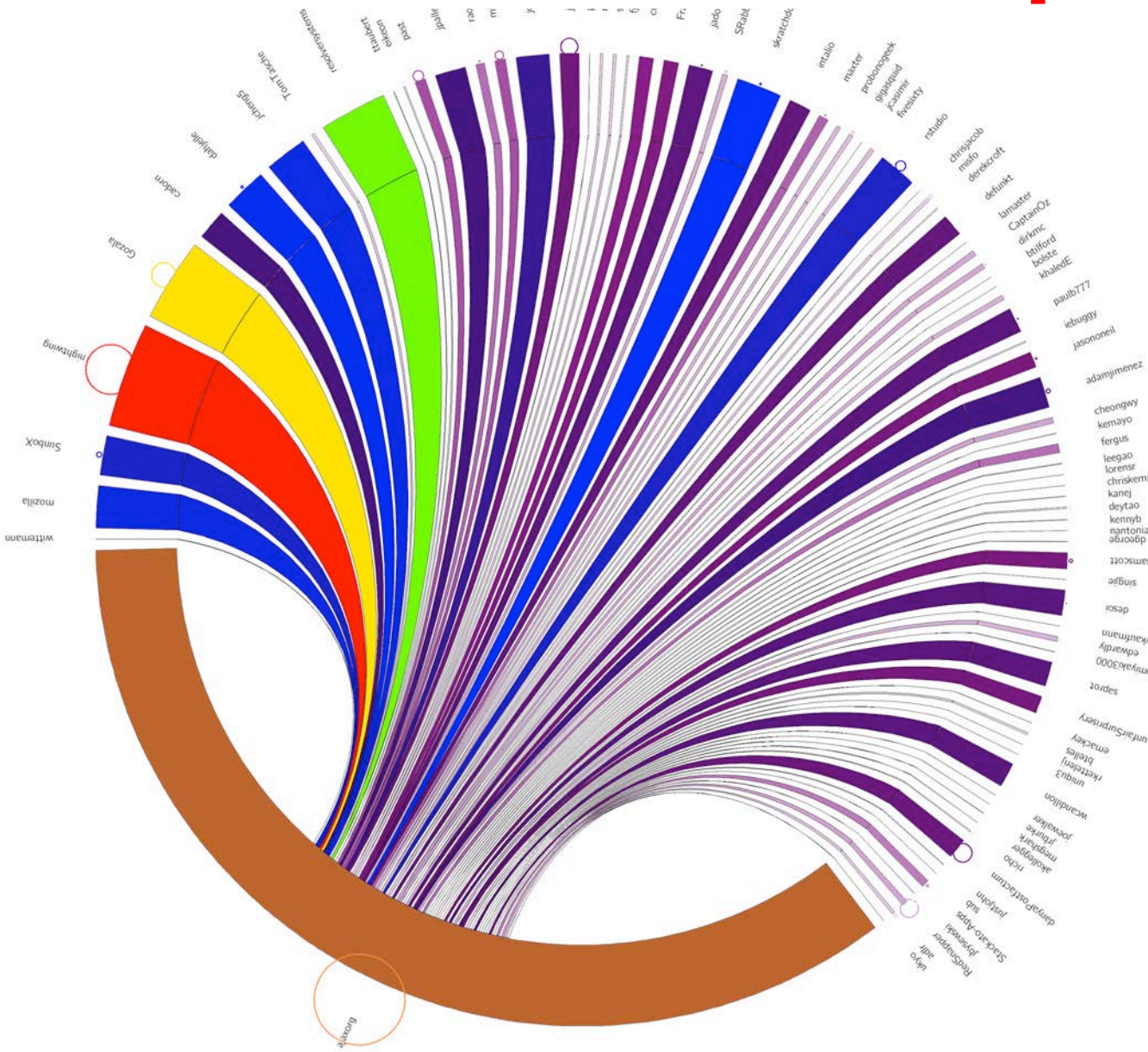
Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
C	2,300,710	476,978	17.2%	452,773	3,230,461	 34.6%
C++	1,206,025	250,128	17.2%	252,971	1,709,124	 18.3%
Java	743,003	699,939	48.5%	179,887	1,622,829	 17.4%
Ada	729,322	335,302	31.5%	252,886	1,317,510	 14.1%
Autoconf	450,574	756	0.2%	71,979	523,309	 5.6%
HTML	214,572	6,279	2.8%	43,661	264,512	 2.8%
Fortran (Fixed-format)	113,138	2,326	2.0%	15,909	131,373	 1.4%
Make	112,507	3,917	3.4%	14,123	130,547	 1.4%
Go	66,921	11,083	14.2%	4,904	82,908	 0.9%
Assembly	51,774	13,375	20.5%	10,080	75,229	 0.8%
XML	49,875	675	1.3%	6,062	56,612	 0.6%
Objective-C	28,137	5,215	15.6%	8,279	41,631	 0.4%
shell script	19,657	5,823	22.9%	4,417	29,897	 0.3%
Fortran (Free-format)	17,068	3,305	16.2%	1,686	22,059	 0.2%
Perl	16,549	3,869	18.9%	2,463	22,881	 0.2%
TeX/LaTeX	12,823	6,358	33.1%	1,639	20,820	 0.2%
Scheme	11,023	1,010	8.4%	1,205	13,238	 0.1%
Automake	10,775	1,210	10.1%	1,626	13,611	 0.1%
Modula-2	4,326	983	18.5%	826	6,135	 0.1%
Objective Caml	2,930	578	16.5%	389	3,897	 0.0%
XSL Transformation	2,896	450	13.4%	576	3,922	 0.0%
AWK	2,318	569	19.7%	376	3,263	 0.0%
CSS	2,049	171	7.7%	453	2,673	 0.0%
Python	1,735	410	19.1%	404	2,549	 0.0%
Pascal	1,044	141	11.9%	218	1,403	 0.0%
C#	879	506	36.5%	230	1,615	 0.0%
DCL	698	154	18.1%	15	867	 0.0%
JavaScript	655	404	38.1%	144	1,203	 0.0%
Tcl	392	113	22.4%	72	577	 0.0%
Haskell	154	0	0.0%	17	171	 0.0%
CMake	134	31	18.8%	25	190	 0.0%
Matlab	57	0	0.0%	8	65	 0.0%
DOS batch script	4	0	0.0%	0	4	 0.0%
Totals	6,174,724	1,832,058		1,330,303	9,337,085	

- Interoperability

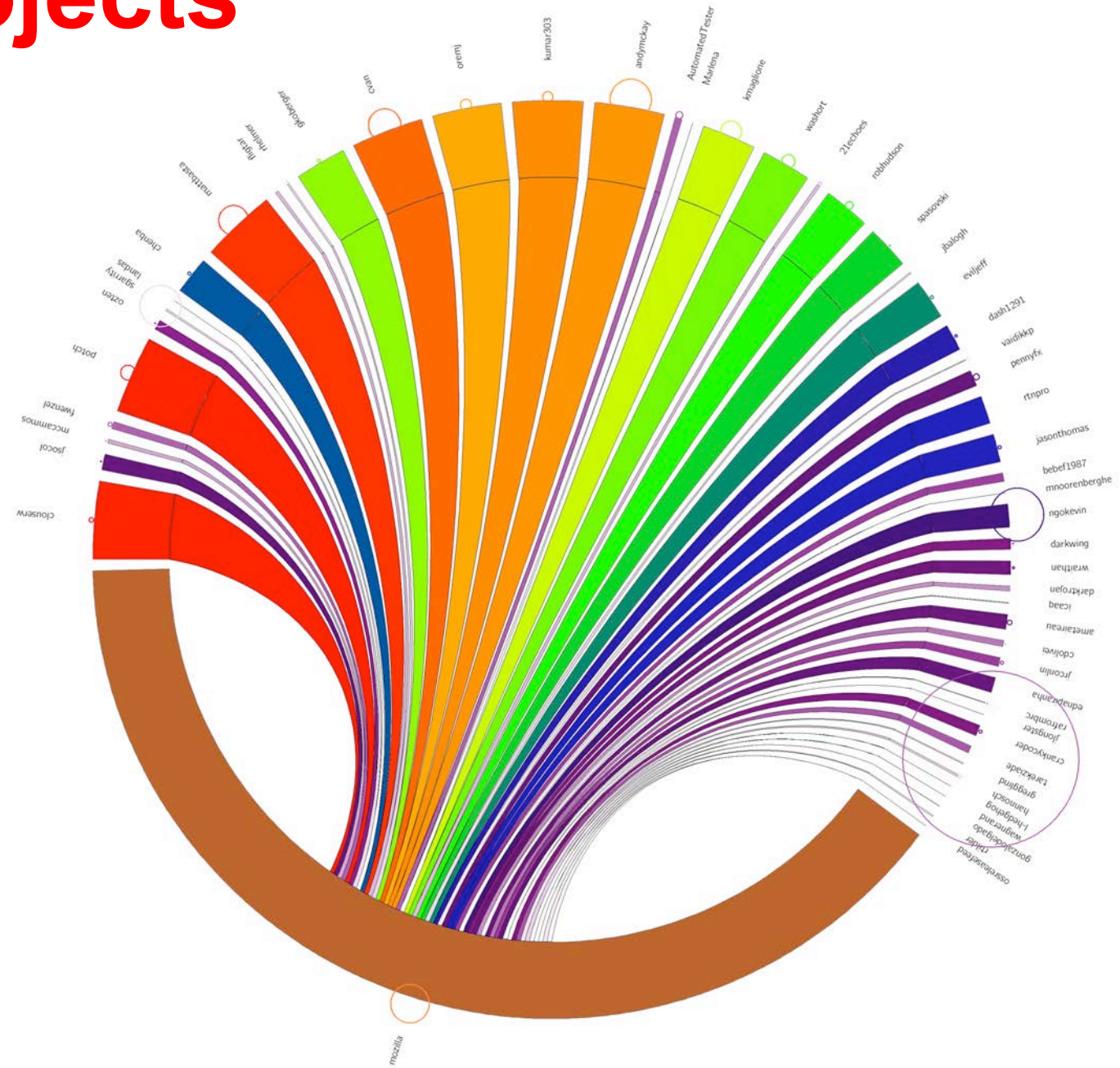
See <http://www.ohloh.net/p/gcc>. Retrieved 2012-09-16.

Programming-in-the-large

- Collaborative projects

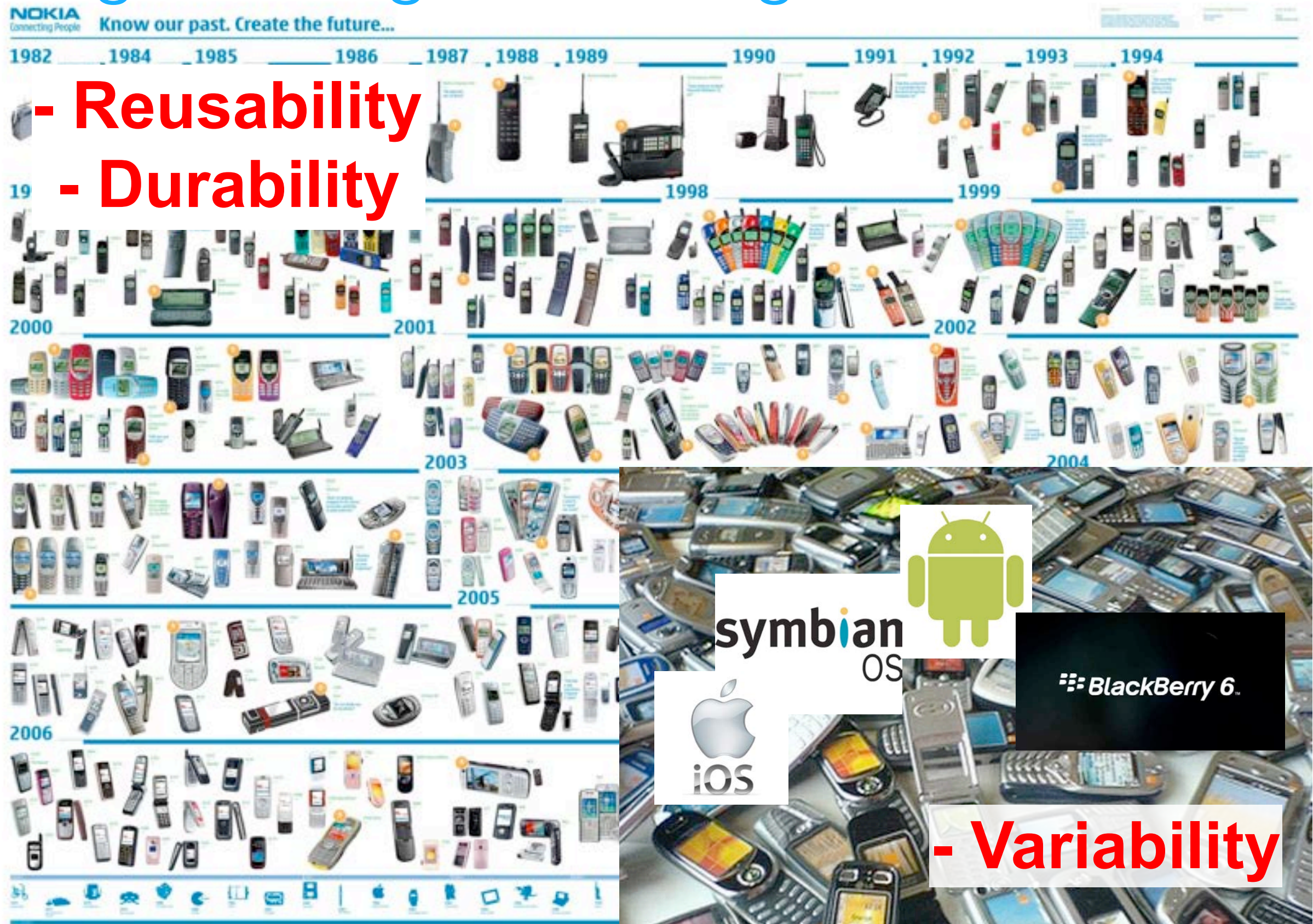


137 contributors, 5000 commits, 1300 forks
<https://github.com/ajaxorg/ace>



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

Programming-in-the-large



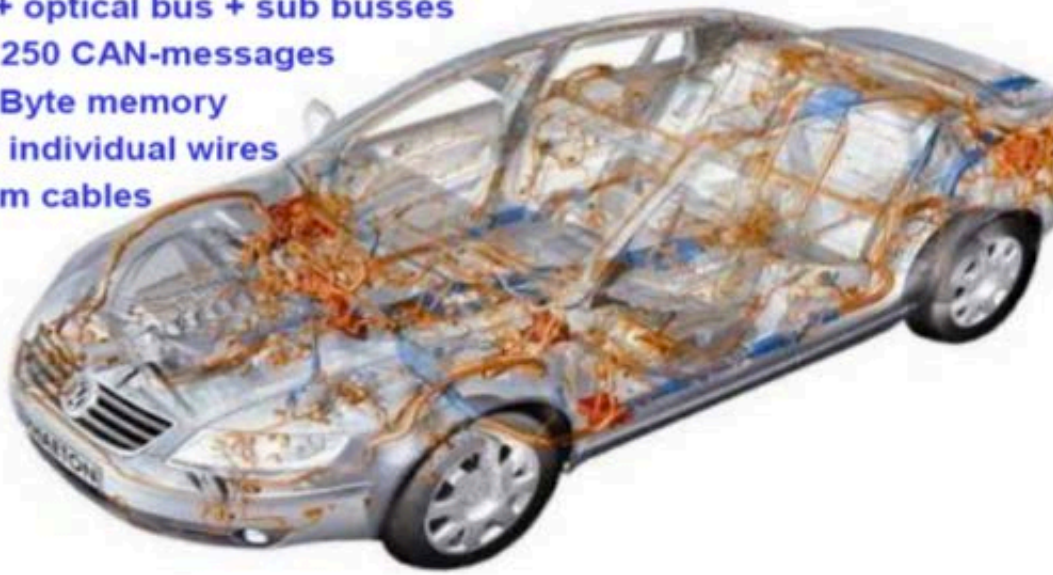
Programming-in-the-large

- 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

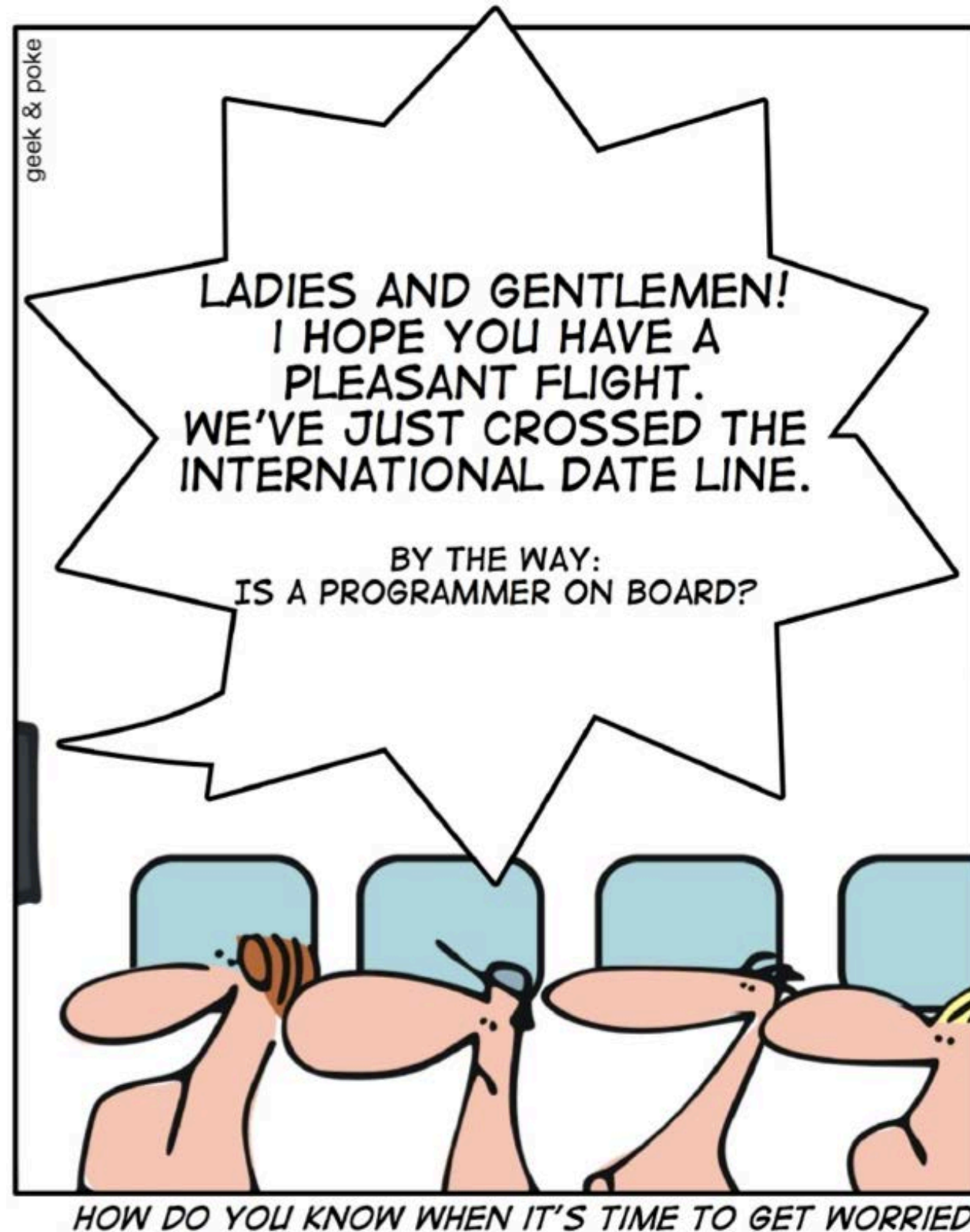


Programming-in-the-large

- ***"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"***

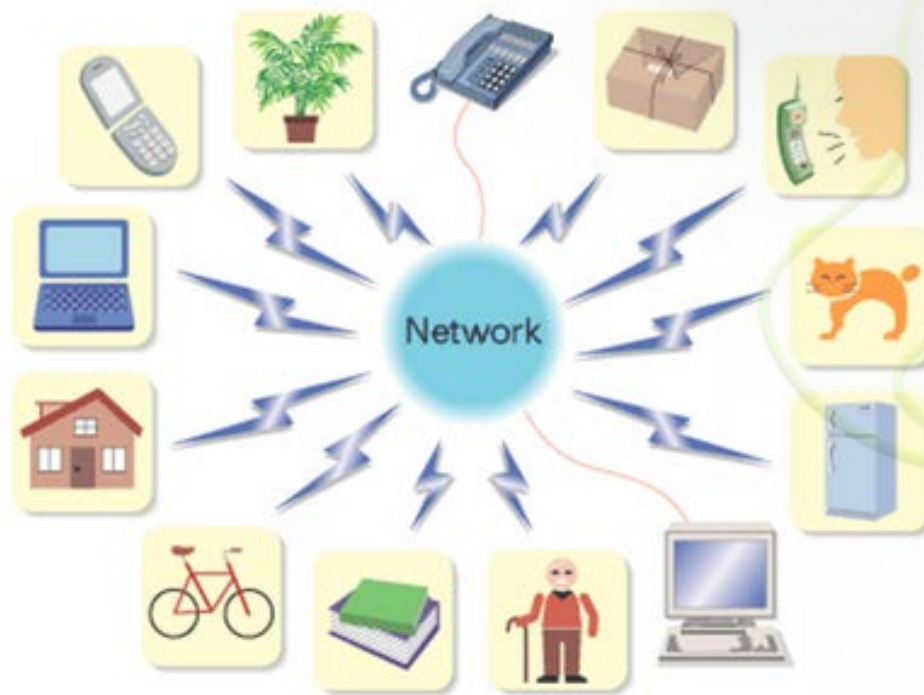
"This Car Runs on Code", By Robert N. Charrette, IEEE Spectrum, Feb. 2009, see <http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code>

Programming-in-the-large



Programming-in-the-large

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



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 bancaire et Cobol)

Long term availability...

AIRBUS A300 Life Cycle

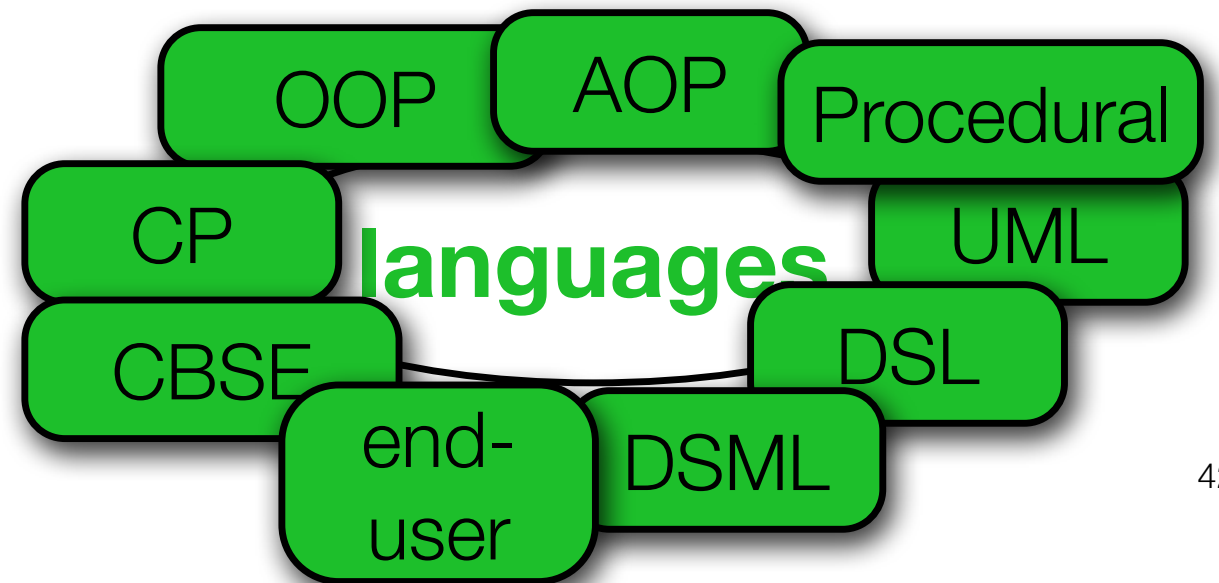
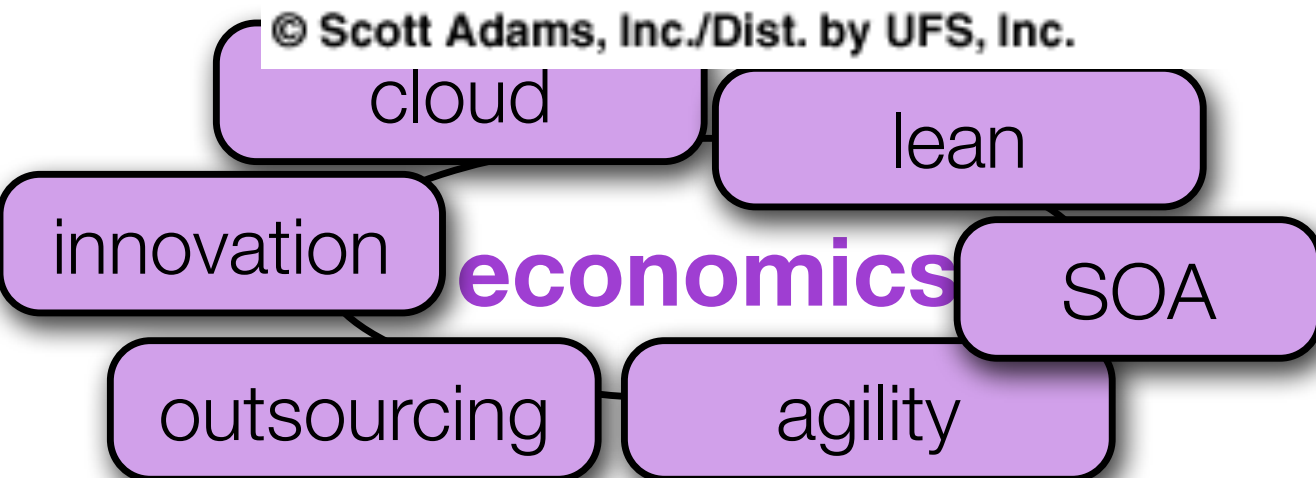
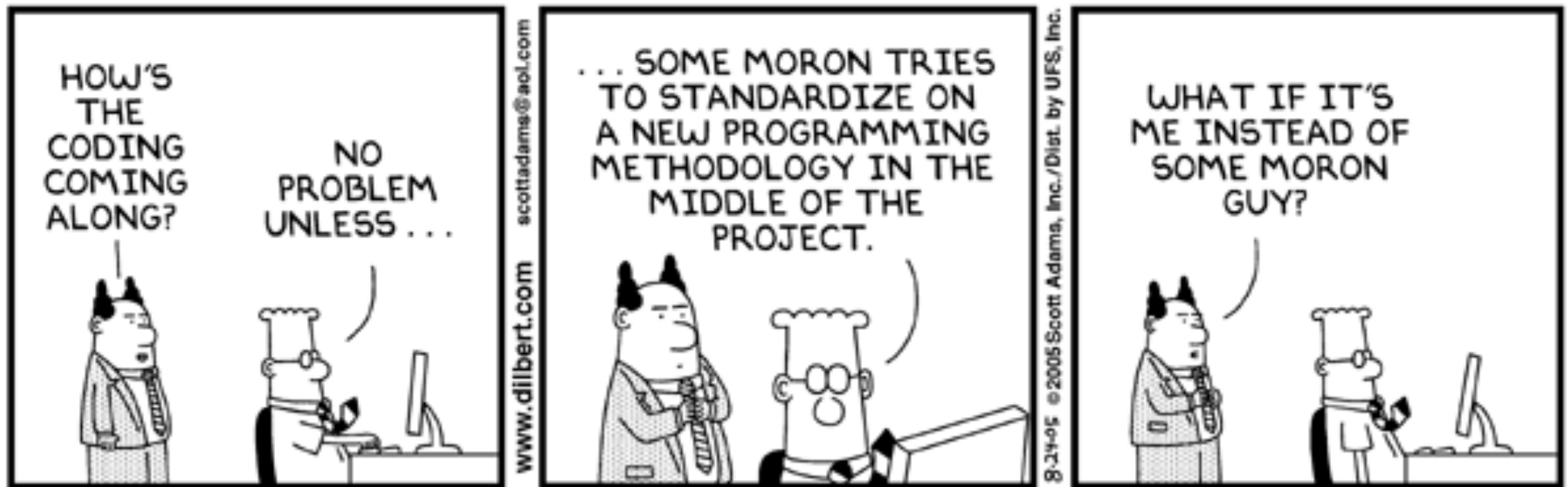
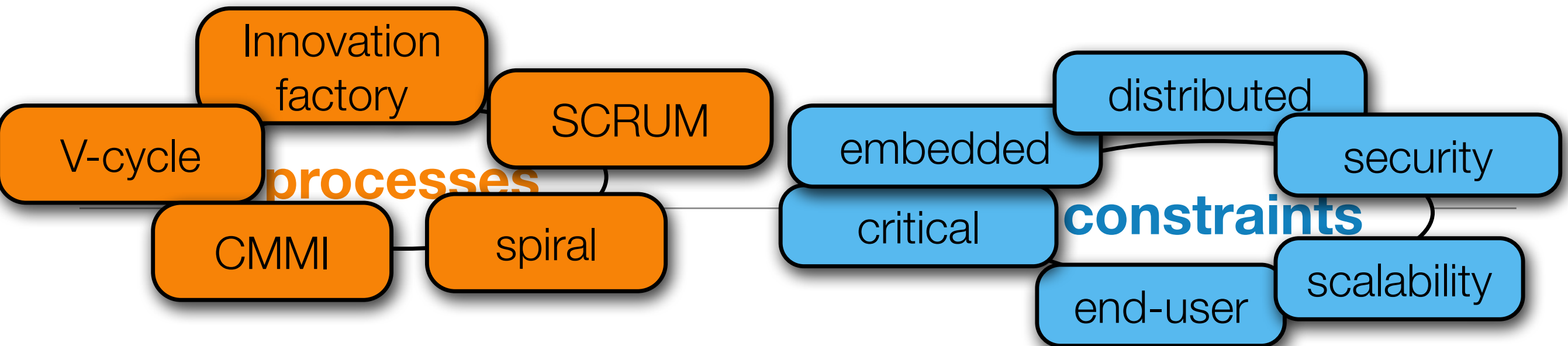
Program began in 1972, production stopped in 2007

2007-1972 = 35 years...

Support will last until 2050

2050-1972 = 78 years !!

**On board software development
for very long lifecycle products**



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

Constructive approach

- Guarantee the absence of bugs
- Top-down approach
- Model-driven development + formal analysis
- Formal proof
 - Automatic or manual
 - Offers exhaustive guarantees 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

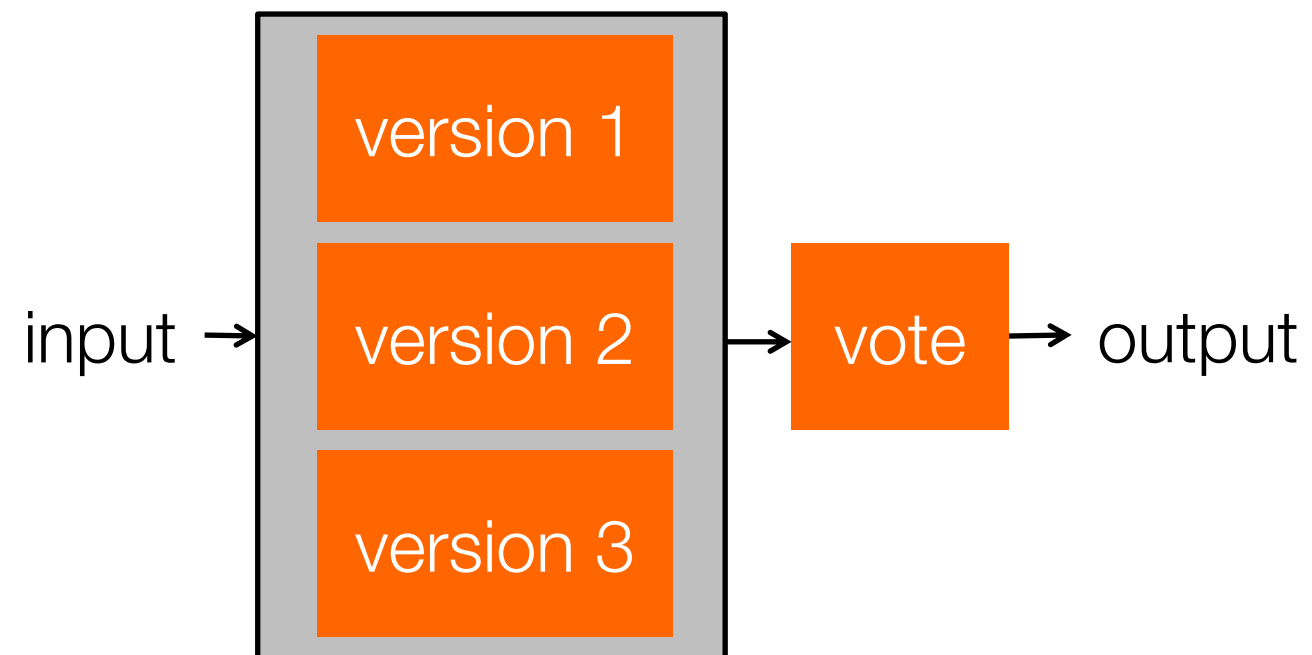
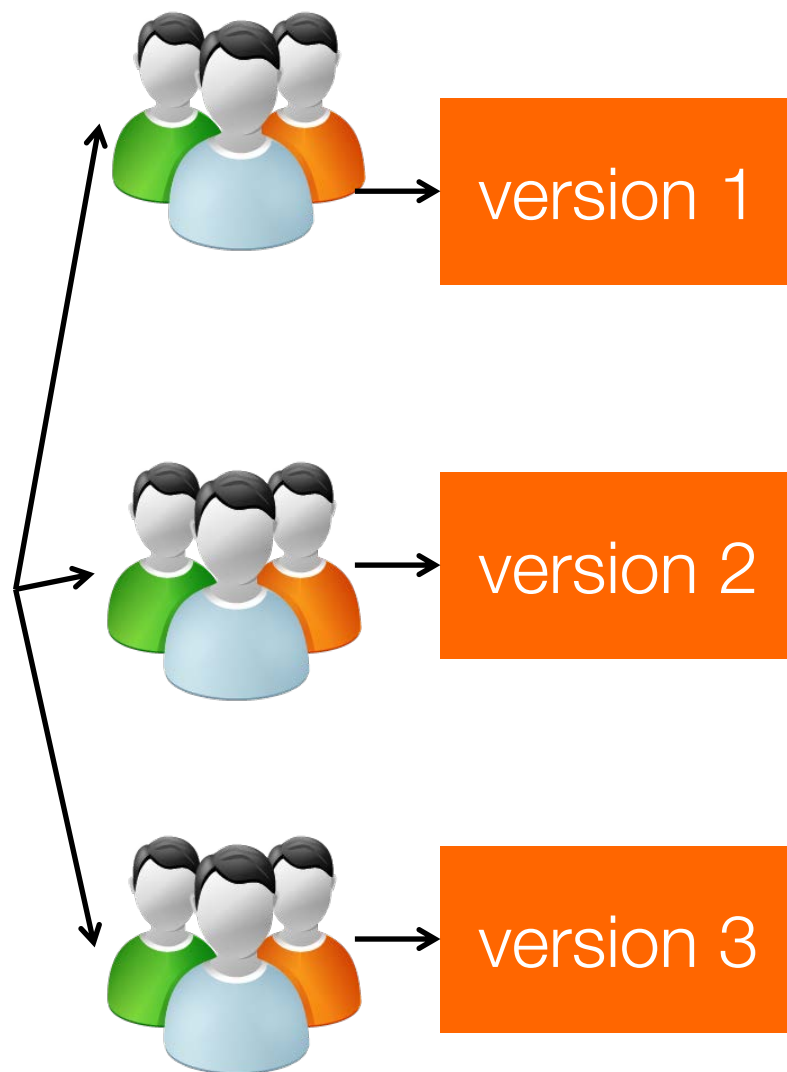
- Look for the presence of bugs
- Heuristic-based
- Analyze all sorts of software artefacts (code, models, requirements, etc.)
- Software testing

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
 - Design diversity at the systems level
 - Exception handling at the source code level
 - Randomization at the machine code level

Fault-tolerance

- N-version programming



Project Name: Requirements Document (version 1.0)

To use this template:

1. Replace any red italicized text with your own text. You may remove or add sections as needed for your particular project.
2. Enter the project name in the title and footer (and change the document version number, if needed).
3. If your document is very long, break each numbered chapter into its own document within the project. This will make it easier to replace updates.
4. Delete these instructions and any other italicized instructions.

Project:
Domain:
Prepared by:

Document status: ___ Draft ___ Proposed ___ Validated ___ Approved

1. Introduction

This document contains the system requirements for **project name**. These requirements have been derived from several sources, including **brief listing of most important sources**.

1.1 Purpose of This Document

This document is intended to guide development of **project name**. It will go through several stages during the course of the project:

1. **Draft**: The first version, or draft version, is completed after requirements have been discussed, recorded, classified, and prioritized.
2. **Proposed**: The draft document is then proposed as a potential requirements specification for the project. The proposed document should be reviewed by several parties, who may comment on any requirements and any priorities, either to agree, to disagree, or to clarify missing requirements. Reviews include end users, developers, project managers, and any other stakeholders. The document may be amended and improved several times before moving to the next stage.
3. **Validated**: Once the various stakeholders have agreed to the requirements in the document, it is considered validated.
4. **Agreement**: The validated document is accepted by representatives of each party of stakeholders as an appropriate statement of requirements for the project. The developers then use the requirements document as a guide to implementation and to check the progress of the project as it develops.

1.2 How to Use This Document

We expect that this document will be used by people with different skill sets. This section explains what parts of this document should be reviewed by various types of readers.

Types of Reader

In this section, list the different types of reader this document is aimed at. For example, **Project programmers, graphic designers, and users, project managers, etc.** For each type of reader, clearly state which sections are most pertinent to them, and which may be safely skipped.

Technical Background Required

Describe here the technical background needed to understand the document in general, and any particular expertise or understanding that is needed for specific sections.

Overview Sections

List here the sections that should be read by someone who only wishes to gain an overall understanding of the project, or which should be read first before technical requirements are reviewed.

Project Name: Requirements Document (version 1.0)

Google's innovation factory

- Google (2012 Update from Larry Page, CEO):
 - *Over 850,000 Android devices are activated daily through a network of 55 manufacturers and more than 300 carriers.*
 - *Google Chrome browser has over 200 million users.*
 - *Google launched Gmail in 2004 and now is used by more than 350 million people.*
 - *YouTube has over 800 million monthly users who upload an hour of video per second.*

See <http://investor.google.com/corporate/2012/ceo-letter.html>

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



Google

Innovation Factory:
Testing, Culture, &
Infrastructure

Patrick Copeland, Google
ICST 2010

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

Netflix's simian army

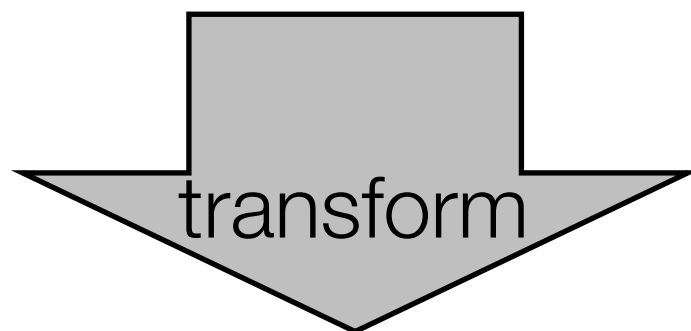
- Streaming TV network service
 - approx. 40 million subscribers
 - very high dependence on software and cloud (runs on Amazon EC2)
 - major player in open source
- Induce failure regularly
 - 'break' production code to check the system's ability to react
 - Chaos monkey: randomly terminates an instance in production
 - Chaos kong: take an entire region offline
 - Latency monkey: artificial delay in RESTful clients



Loop perforation: when good enough is better

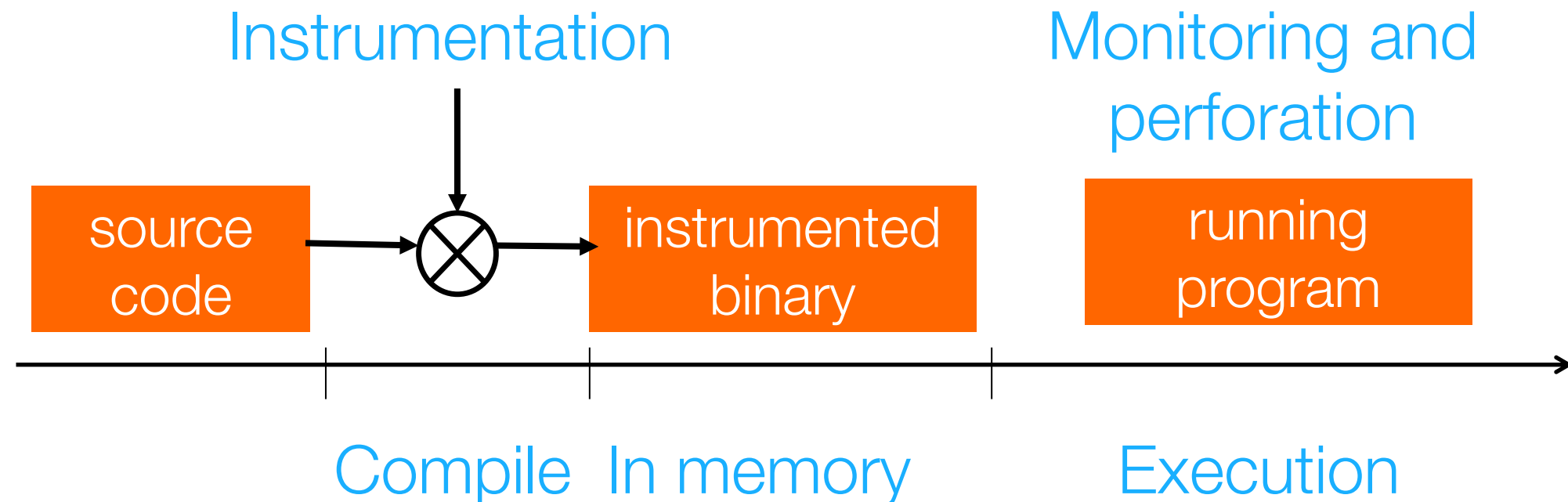
- “It used to be that people used computers for computations where there was a single, hard, logical right answer”
- Trade-off between accuracy and performance

for ($i=0$; $i < b$; $i++$) { ... }



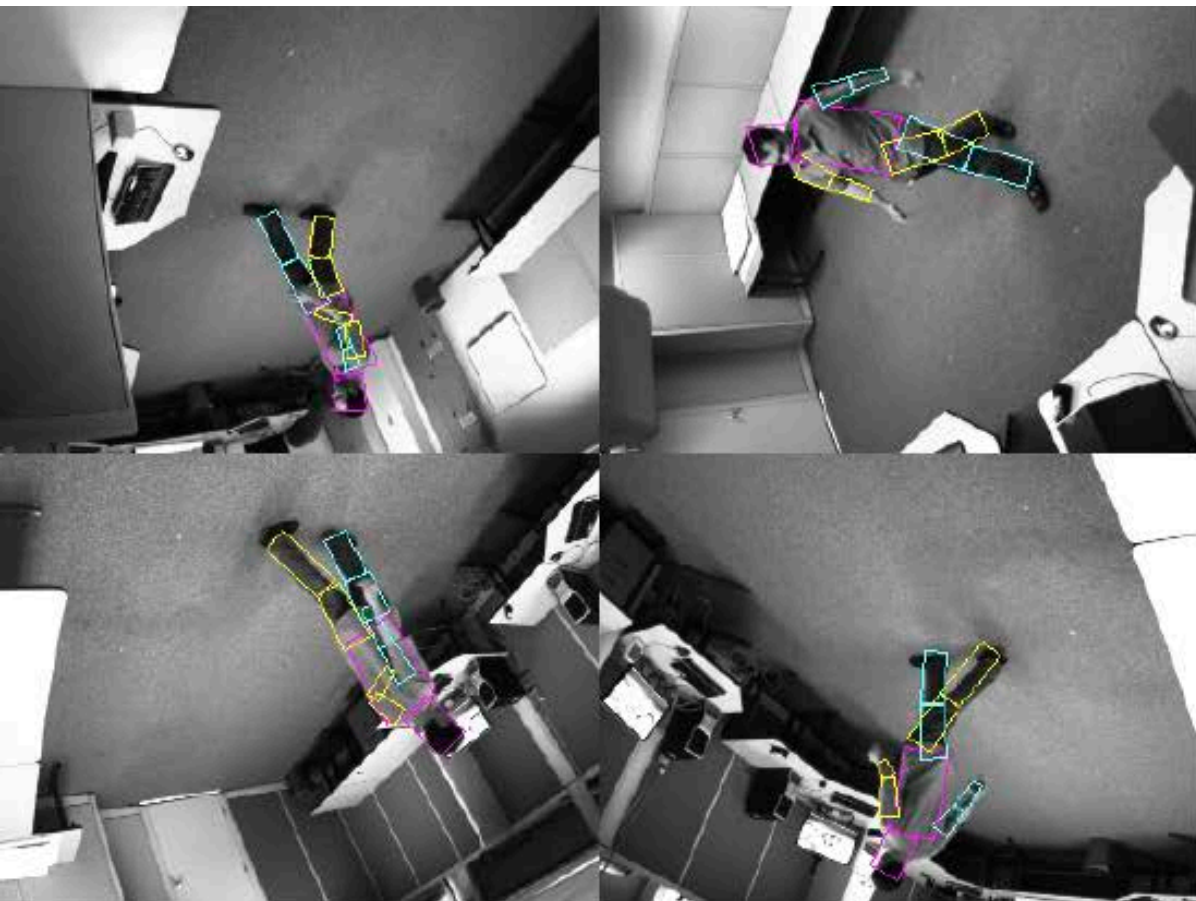
for ($i=0$; $i < b$; $i+=n$) { ... }

Loop perforation



- unsound transformation
- still useful

Loop perforation



In this class

- Software testing
 - most probably the technique you'll have to use
- for verification
 - validation is essential but requires the involvement of users => usually done by specific teams, who don't develop
- from the developer's perspective
 - you should test your software and you'll be assigned testing tasks in your future jobs

Acknowledgment

To make a long story short, all materials are shared since a long time with various friends and colleagues, and are the results of the collaborative effort of all.

Big thanks to, among others, Benoit Baudry (KTH, Sweden), Yves LeTraon (Univ. Luxembourg), and Jean-Marc Jézéquel (Univ. Rennes 1, France).