What's going on inside your brain when you (don't) find a bug?

(and what we can do to improve software reliability?)

Henrique Madeira, João Durães,
João Castelhano, Catarina Duarte, and Miguel Castelo Branco

University of Coimbra, Portugal

68th Meeting of the IFIP 10.4 Working Group on Dependable Computing and Fault Tolerance, Búzios, Brazil, June 25-28, 2015

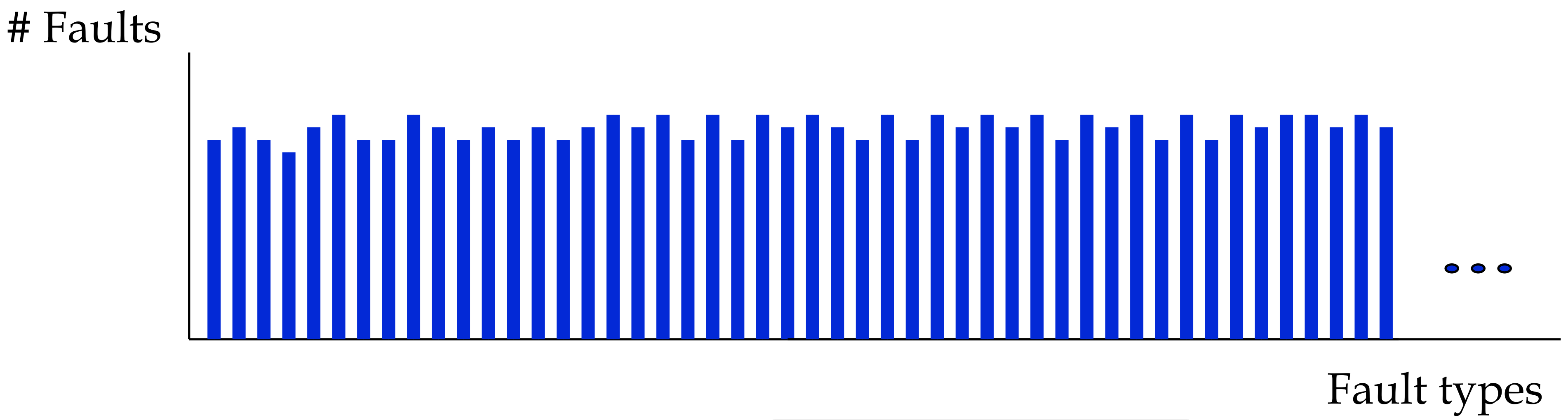
University of Coimbra

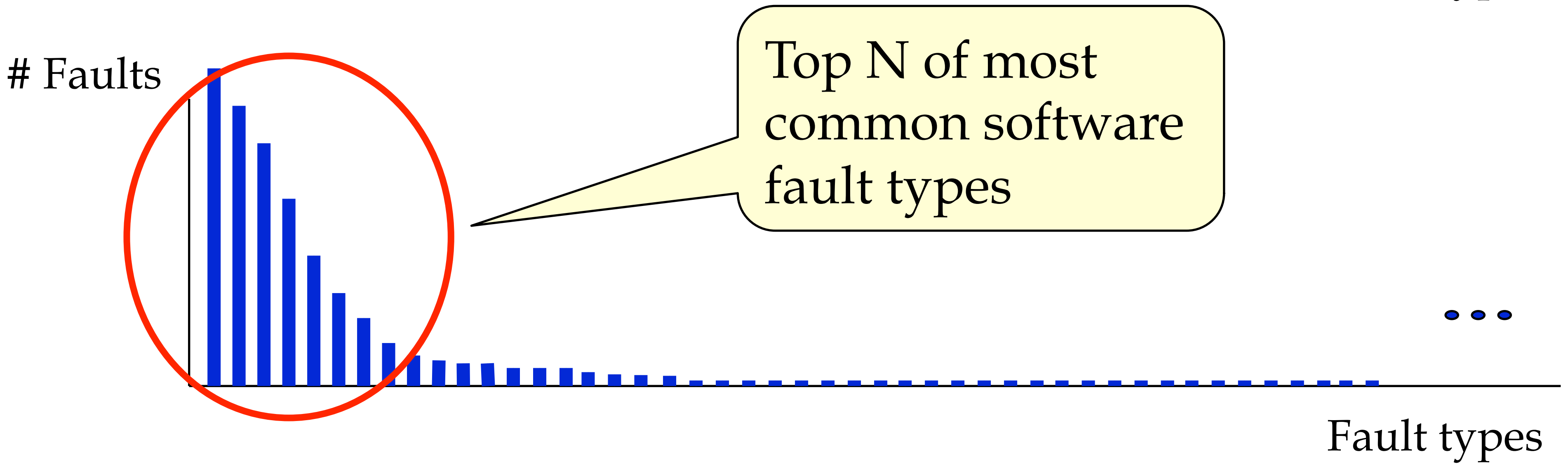
Some years ago…
Fault models for software faults

- **Software faults**
  - **Residual bugs** that escape testing and go to the deployed component/product
  - Simplification: focus on **faults in the code**; i.e., assume the requirements and functional specification are correct
- Research question: **are software faults different one from each other or most of them fall in a small set of fault types?**

(ODC classification and data field available was not fine grain enough to answer the question)

Some years ago…
Fault models for software faults





The "Top-N" software faults

Fault types	**Perc. Observed in field study**	**ODC classes**
Missing "If (*cond*) { statement(s) }"	9.96 %	Algorithm
Missing function call	8.64 %	Algorithm
Missing "AND EXPR" in expression used as branch condition	7.89 %	Checking
Missing "if (*cond*)" surrounding statement(s)	4.32 %	Checking
Missing small and localized part of the algorithm	3.19 %	Algorithm
Missing variable assignment using an expression	3.00 %	Assignment
Wrong logical expression used as branch condition	3.00 %	Checking
Wrong value assigned to a value	2.44 %	Assignment
Missing variable initialization	2.25 %	Assignment
Missing variable assignment using a value	2.25 %	Assignment
Wrong arithmetic expression used in parameter of function call	2.25 %	Interface
Wrong variable used in parameter of function call	1.50 %	Interface
Total faults coverage	**50.69 %**	

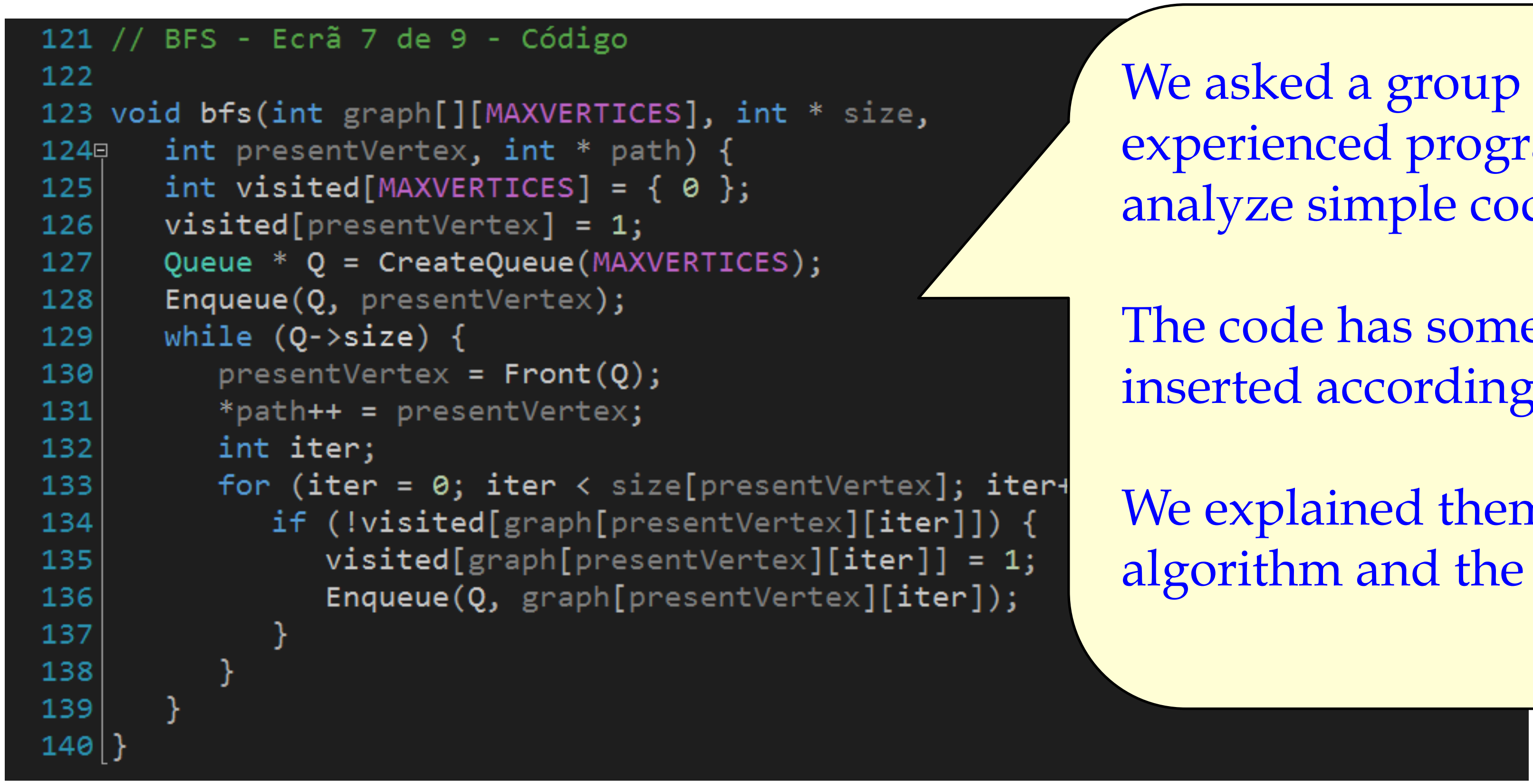
Field data studies on SW faults and SW fault models representativeness

For more details:

- **"Definition of Software Fault Emulation Operators: a Field Data Study"**, J. Durães and H. Madeira, IEEE/IFIP International Conference on Dependable Systems and Networks, Dependable Computing and Communications, DSN-DCC 2003, San Francisco, CA, USA, June 22-25, 2003.
- **"Emulation of Software Faults: A Field Data Study and a Practical Approach"**, J. Durães and H. Madeira, IEEE Transactions on Software Engineering, Vol. 32, No. 11, November 2006.
- **"On Fault Representativeness of Software Fault Injection"**, R. Natella, D. Cotroneo, J. Duraes, H. Madeira, IEEE Transactions on Software Engineering, December 2011

What really is a software faults?

Simple experiment of SW fault injection (similar to defect seeding)



```
// BFS - Ecrã 7 de 9 - Código

void bfs(int graph[][MAXVERTICES], int * size,
    int presentVertex, int * path) {
    int visited[MAXVERTICES] = { 0 };
    visited[presentVertex] = 1;
    Queue * Q = CreateQueue(MAXVERTICES);
    Enqueue(Q, presentVertex);
    while (Q->size) {
        presentVertex = Front(Q);
        *path++ = presentVertex;
        int iter;
        for (iter = 0; iter < size[presentVertex]; iter+
            if (!visited[graph[presentVertex][iter]]) {
                visited[graph[presentVertex][iter]] = 1;
                Enqueue(Q, graph[presentVertex][iter]);
            }
        }
    }
}
```

We asked a group of 12 experienced programmers to analyze simple code snippets.

The code has some bugs, inserted according to the Top N.

We explained them first the the algorithm and the pseudo code.

Observations

- **Each participant found only a fraction of the bugs.**
- **All of them indicated some wrong bugs (false positives)**
- **The results of the group cover all the bugs**

Some questions

- If SW bugs are such simple things, why do programmers so often fail see them?
  (even when we remove all the context that makes things complex)
- Why do some people see a given bug while others don't?
- Why is the percentage of false positives so high?
- What can we do to improve the chances of spotting more bugs during program coding (and during testing)?

What's going on inside your brain when you (don't) find a bug?

Functional Magnetic Resonance Imaging (fMRI)

Added features

- Screen
- Eye tracking
- Joystick

Functional Magnetic Resonance Imaging (fMRI)

- fMRI uses the magnetic properties of blood to analyze brain activity in specific areas.
- Based on small changes in blood. Referred to as BOLD (Blood Oxygen Level-Dependent) imaging.
- Creates highly detailed 3D images of the brain in successive instants (sampling 2 seconds)
- Active areas of the brain in a given moment are detected by filtering out the active voxels, when compared to a base level activity.

Experiment protocol

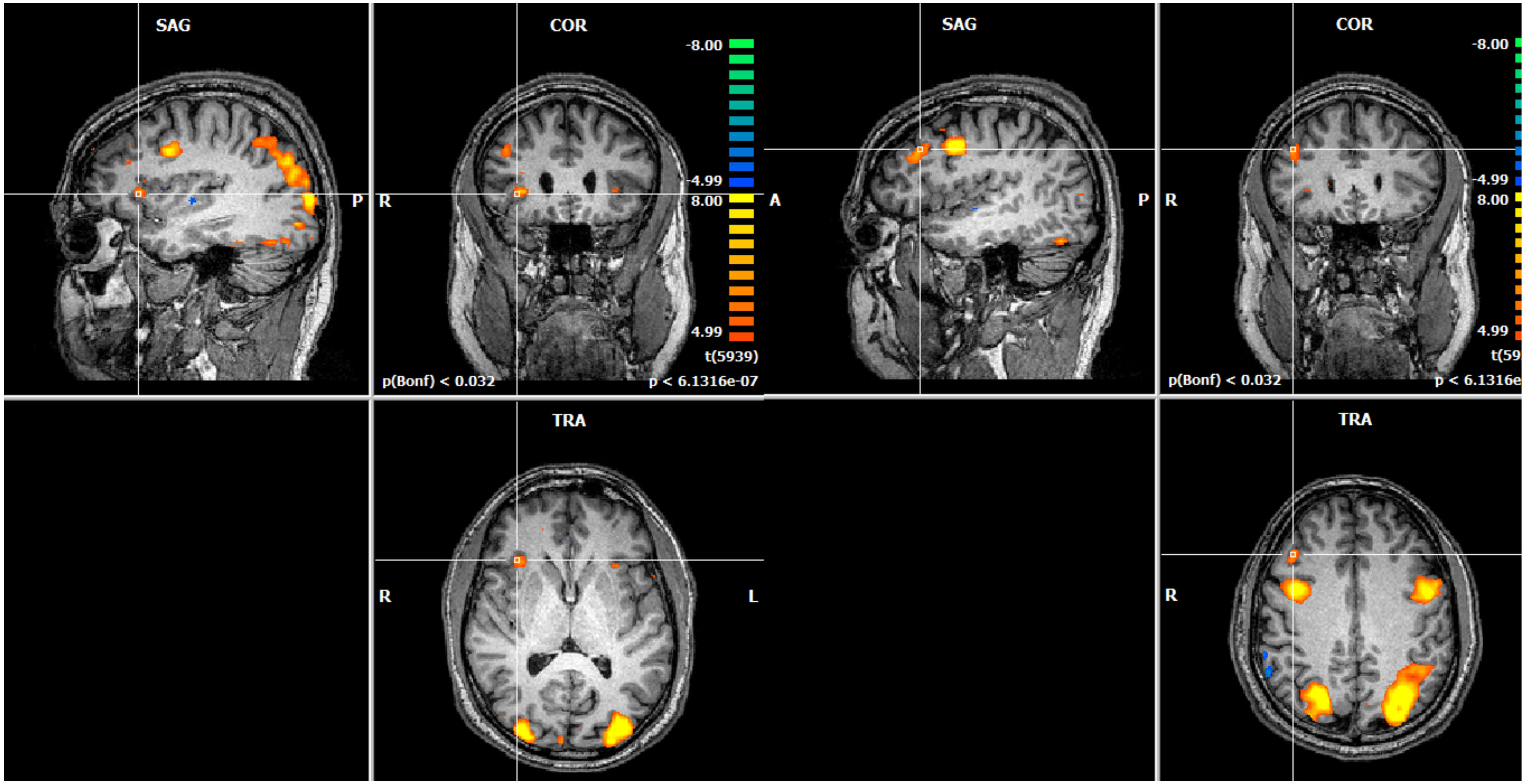
- Group of volunteers, divided in two groups: very experienced programmers and experienced programmers
- Three simple programs in C: quick sort, shell sort and matrix multiplication
- All programs contain a small number of realistic bugs, inserted beforehand
- The algorithm and the pseudo code is explained to each volunteer, before the experiment.
- Each volunteer analyzes the code inside the fRMI:
  - Records the bugs he/she founds
  - Corrections are allowed (i.e., clear a bug indication)
  - The eye tracking is synchronized with the fRMI (same time scale)
  - After the session inside the fRMI, the volunteer indicates the level of confidence he/she has on the each bug identified

Current status

- 8 full experiments so far (we need some more)
- The analysis of the data is very heavy and takes a long time. Includes:
    - Functional areas of the brain activated while the volunteer analyses areas of interest (i.e., where the bugs are, where the false positives were indicated, etc.)
    - Particular attention to brain areas related to abstract knowledge (cognitive), decision taking, association, short term memory, among others.
    - Patters from the eye tracking and correlation with the fRMI

Sample of fRMI image

Results from 8 volunteers; basic areas activated in ALL volunteers



Where are we looking at?

- Activation of brain decision areas in code lines where a bug was injected.
- Brain activity when the volunteer found a real bug and when indicated a false positive.
- Impact of the code complexity where bug is inserted.
- Impact of recursive code structures.

Preliminary results (at code inspection level) indicate a big difference between highly experienced programmers and experienced programmer → correspondence to brain activity

Some unique features of the experiment

- To the best of our knowledge, it’s the first time decisions based on highly abstract concepts are analyzed using fRMI.
- Software code seems to be a good choice because we have clear complexity metrics.
- Results are being exploited in two directions: by neuroscientist and by software engineering researchers.
- Risky experiment… in the sense that we have no guaranties to find something relevant.