A neuroscience and AI approach to software bugs: expectations and some tangible results

Henrique Madeira

Department of Informatics Engineering

Faculty of Science and Technology

University of Coimbra - Portugal

76th Meeting of the IFIP 10.4 Working Group on Dependable Computing and Fault Tolerance
Hood River - 27 June 2019 — 1 July 2019

University of Coimbra

Software faults (bugs)

Specification

Design

< - - - >

Code

```
context "on the lessons index page" do

  let(:course1) { Course.first }
  let(:lesson1) { course1.lessons.first }

  before do
    visit course_path(course1.title_url)
  end
```

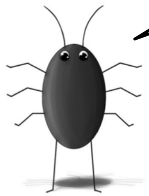
```
hat course" do
```

```
, :text => lesson.title)
```

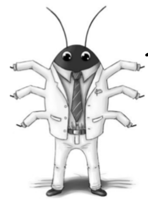
```
  end

  it "should not include lessons for any other course" do
    not_included_lesson = Course.where("id != #{course1.id}").first
      .lessons.first
    # puts not_included_lesson.inspect
    subject.should_not have_selector("h3", :text =>
```

Bugs are a very, very, very difficult problem…



I'm a **bohrbug!**



I'm a **mandelbug!**

I'm an **age related bug!**

Software faults: a persistent problem

- Software reliability is mainly based on fault avoidance using **good software engineering methodologies**
- In real systems (i.e., not toys) → **fault avoidance not successful** → **Fault-tolerance is needed**, unless the impact of failures is acceptable.
- Rule of thumb for fault density in software (Rome labs, USA)
  - **10-50 faults per 1,000 lines of code** → for good software
  - **1-5 faults per 1,000 lines of code** → for critical applications using highly mature software development methods and having intensive testing

Software faults: a persistent problem

- Software reliability is mainly based on fault avoidance using **good software engineering methodologies**

- In real systems (i.e., ... **ssful → Fault-tolerance is needed,** ... ble.

- Rule of thumb for fa ...)
  - **10-50 faults per 1,0** ...
  - **1-5 faults per 1,00** ... ng highly mature software developme ...

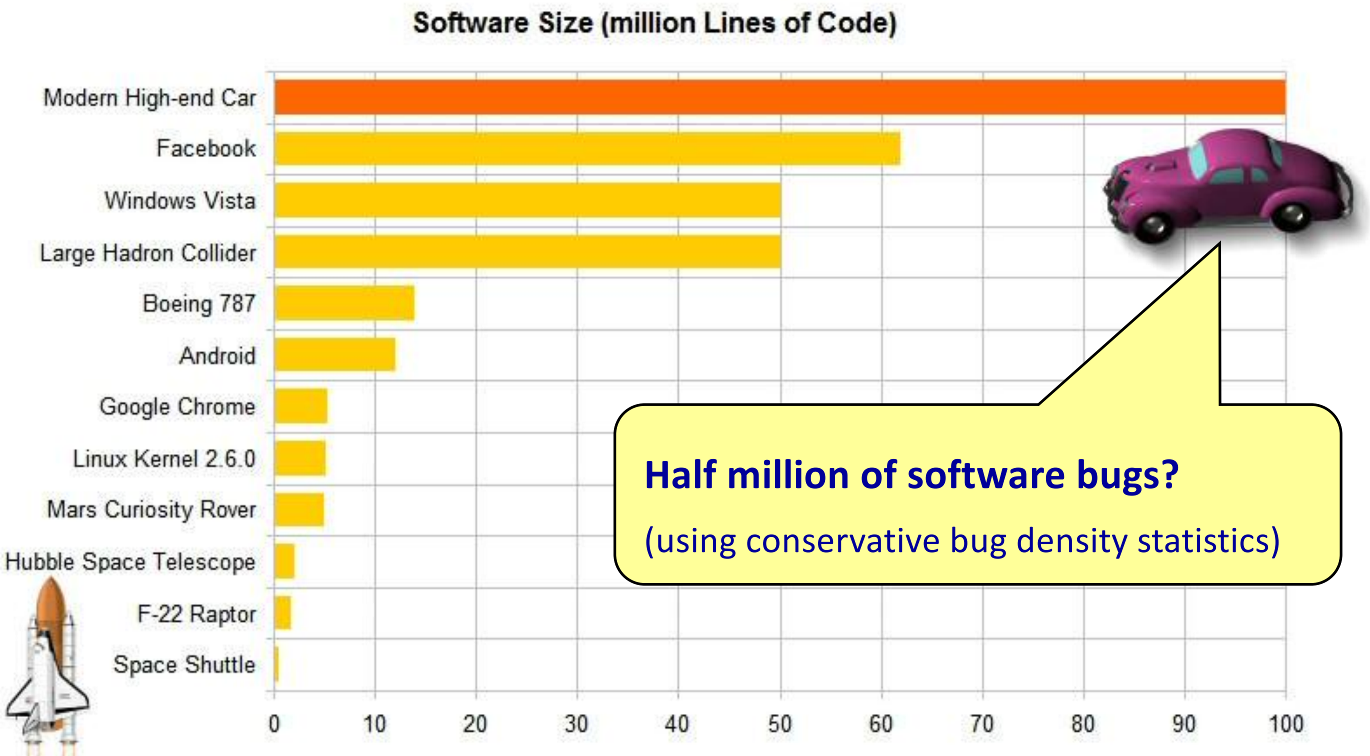
> - SW development methodologies
> - Static analysis tools
> - Software inspections
> - Model checking
> - Testing, testing, testing
> - Verification and validation
> - …

Software faults: a persistent problem

- Software reliab[...] using **good software engin[...]**
- In real systems [...]ul → **Fault-tolerance is nee[...]**
- Rule of thumb f[...]
  - **10-50 faults p[...]**
  - **1-5 faults per[...]** highly mature software devel[...]

Henrique Madeira, DEI-FCTUC, 2019

Size matters: examples

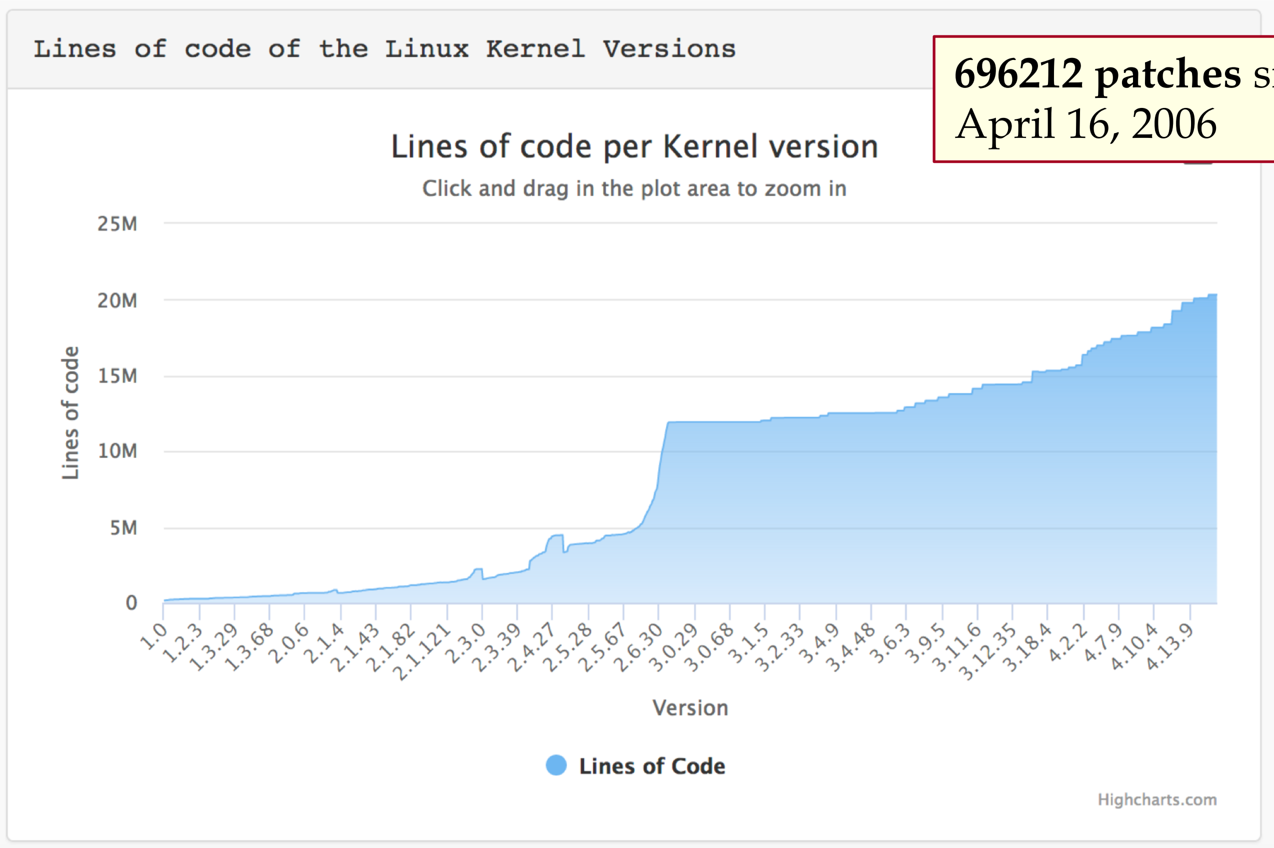


Half million of software bugs?

(using conservative bug density statistics)

From Rich Rogers, https://twitter.com/richrogersiot/status/958112741218111489

Linux kernel size: another example

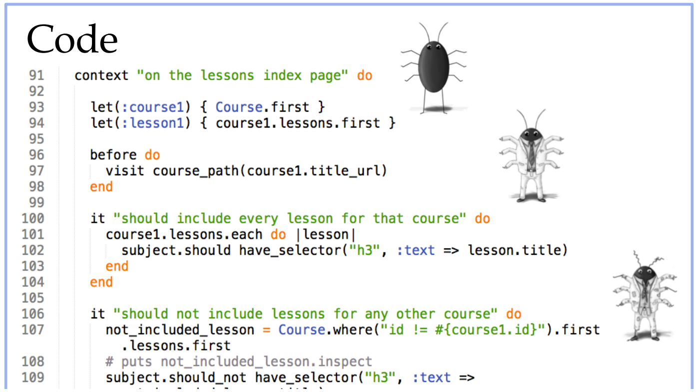


696212 patches since April 16, 2006

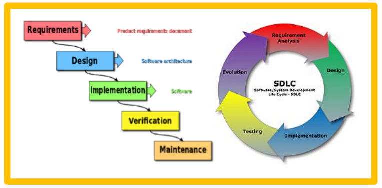
Henrique Madeira, DEI-FCTUC, 2019

Three communities: three attitudes towards bugs

Reality…

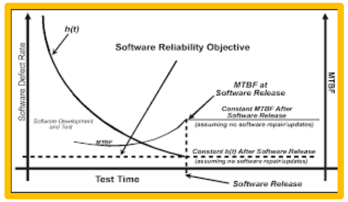


Software Engineering



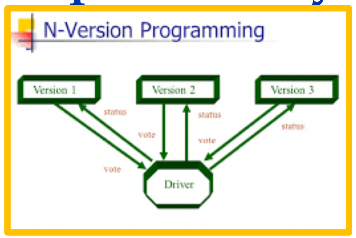
The process is the solution

Software Reliability



Models and tools are the solution

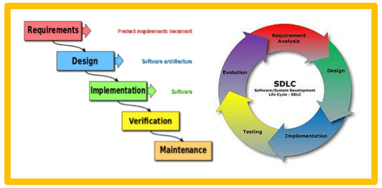
Dependability



Architecture is the solution

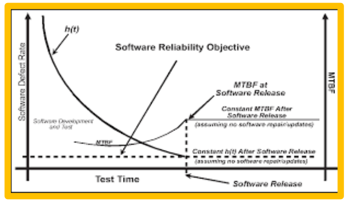
Three communities: three attitudes towards bugs

Software Engineering



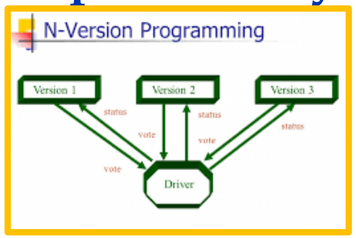
…tion

Software Reliability



…the solution

Dependability



Architecture is the solution

What is missing?...

→to study the root causes of bugs as result of **human errors** in highly abstract and complex tasks, such as code development and code inspection

```
    it "should not include lessons for any other course" do
      not_included_lesson = Course.where("id != #{course1.id}").first
        .lessons.first
      # puts not_included_lesson.inspect
      subject.should_not have_selector("h3", :text =>
```

A neuroscience and AI approach to software bugs: expectations and some tangible results

Outline

- Introduction
- Neuroscience perspective on software code
  - Code comprehension
  - What's going on inside your brain when you (don't) find a bug?

Results from experiment

- Expectations and some tangible results
  - Biofeedback Augmented Software Engineering
  - Intelligent code biofeedback annotation using HRV and pupillography

Results from experiment

- Conclusion

Neuroscience perspective on software code

Code comprehension

Neuroscience perspective on software code

Medical Imaging for Software Engineering

- fMRI – Functional Magnetic Resonance Imaging
- EEG – Electroencephalography
- fNIRS – Functional Near-Infrared Spectroscopy

Neuroscience perspective on software code

Keynote at ICSE/ICPC, May 2019

ICSE 2019 (series) / ICPC 2019 (series) / Presentations /

What goes on in your brain when you read and understand code?

Track ICPC 2019 ICPC Presentations

When **Sat 25 May 2019 09:15 - 10:00 at Laurier** - Keynote Chair(s): Federica Sarro

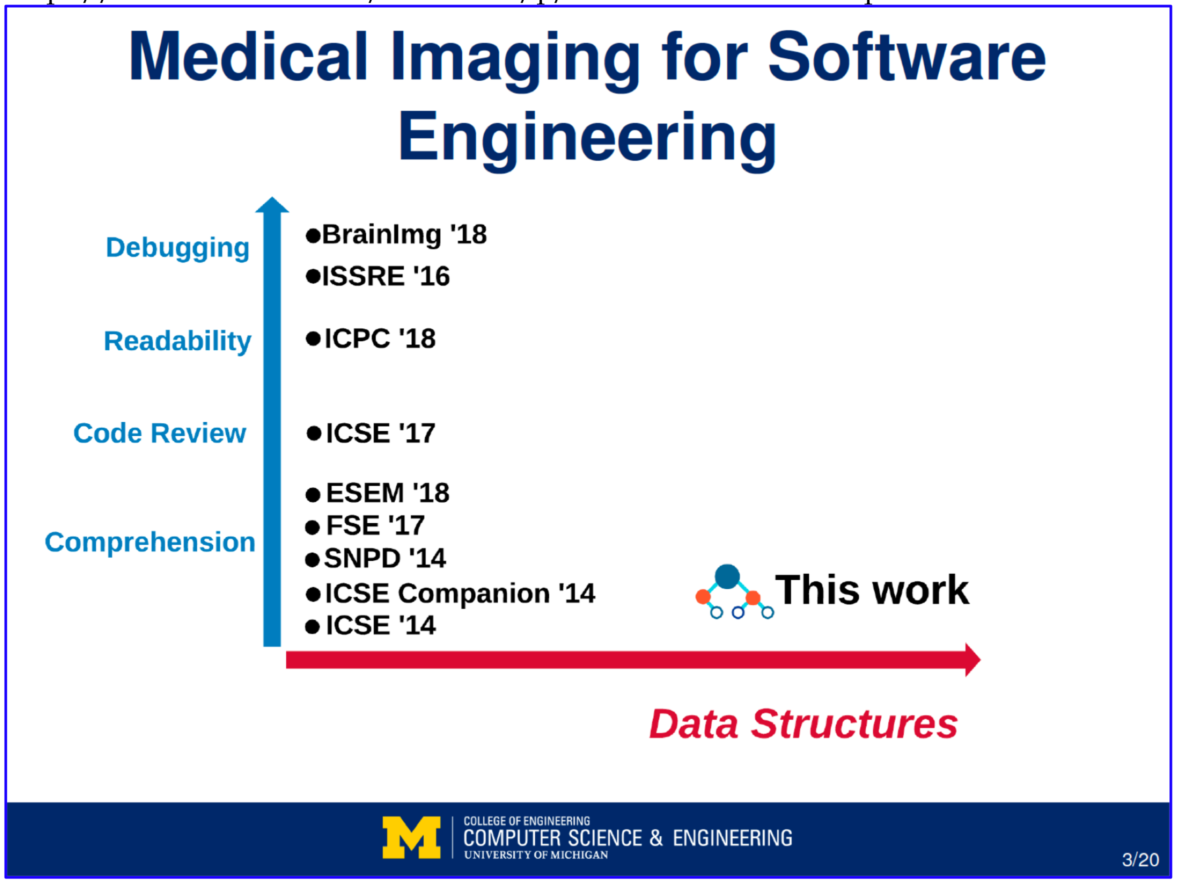
Abstract Within the last few years, high-resolution medical imaging technologies have grown in popularity for research in software engineering in general and program comprehension in particular. New approaches such function magnetic resonance imaging (fMRI) and functional near-infrared spectroscopy (fNIRS) complement more established approaches such as eye tracking and electroencephalograms (EEG), helping us to augment unreliable or subjective self-reporting with more objective measures of the neurobiological correlates of software engineering. This keynote summarizes recent exciting results using such techniques, from multiple authors, contrasting them to more traditional studies. We highlight the "game changing" areas of program comprehension that can be more rigorously targeted with these approaches (including expertise, efficiency, and problem difficulty, among others). We also lay out a number of the challenges associated with such studies (including experimental design, statistical analysis, regulatory compliance, reproducibility, and cost, among others). We conclude with a call to arms, surveying compelling ideas and experiments from psychology that have not yet been applied to program comprehension research.

Westley Weimer
University of Michigan

Henrique Madeira, DEI-FCTUC, 2019

Neuroscience perspective on software code

https://web.eecs.umich.edu/~weimerw/p/weimer-icse2019-slides.pdf



Less than 12 papers so far… but the trend is growing fast.

All studies are exploratory; far from being definitive.

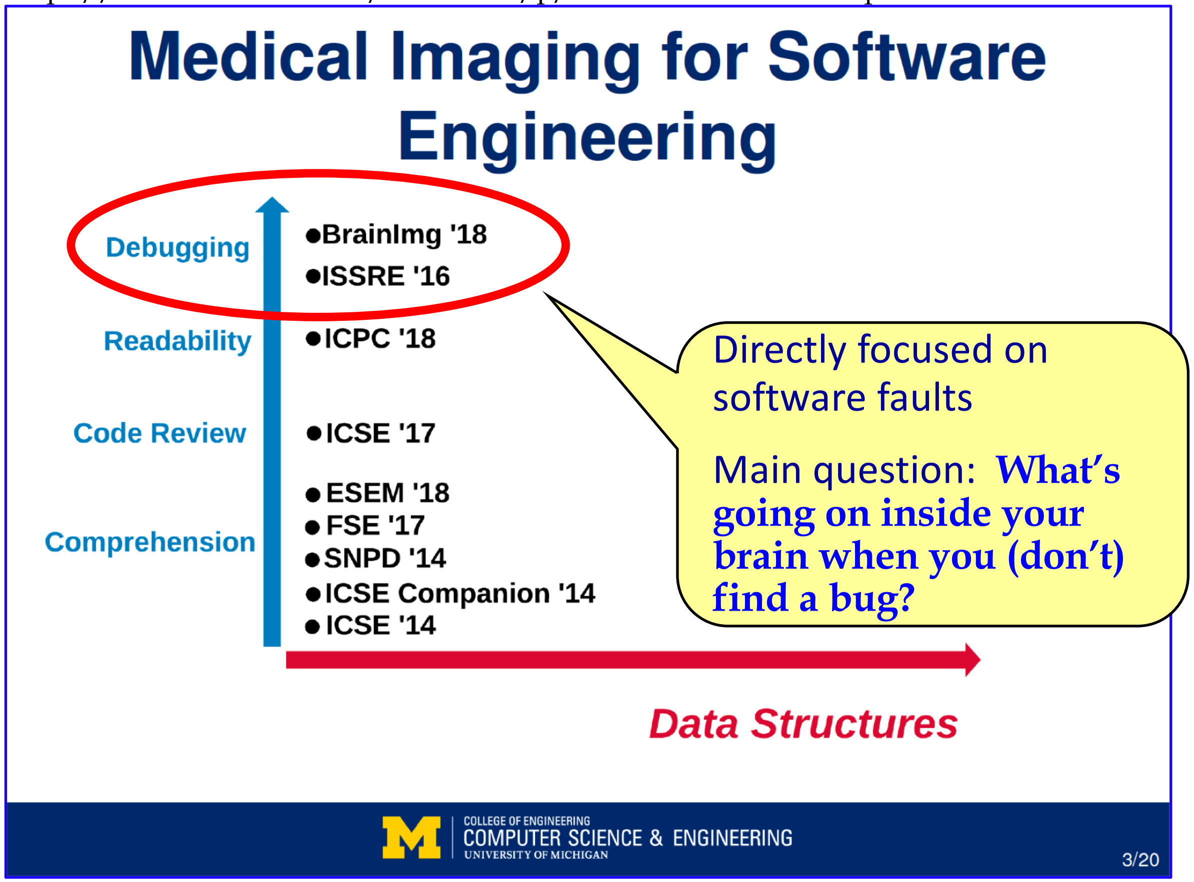
- "**Distilling Neural Representations of Data Structure Manipulation using fMRI and fNIRS**", Yu Huang, Xinyu Liu, Ryan Krueger, Tyler Santander, Xiaosu Hu, Kevin Leach and **Westley Weimer**, International Conference on Software Engineering (ICSE) 2019.
- "**A Look into Programmers' Heads**", Norman Peitek, **Janet Siegmund,** Sven Apel, Christian Kästner, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, André Brechmann , IEEE Transactions on Software Engineering, August, 2018.

Some general conclusions from fMRI/fNIRS studies

- Code comprehension is linked to the activation of five brain regions, which are related to working memory, attention, and language processing.
- Language processing seems to be essential for code comprehension (Dijkstra was right…) but..
- Brain regions related to mathematic processing were also active (in another study, suggesting that the code task is determinant for the language/math balance)
- fMRI (and possibly fNIRS) can be used to measure programming experience and knowledge
- Neural relationship between mental spatial ability and abstract data structure manipulation (but participants reported no subjective experience of similarity).

Neuroscience perspective on software code

https://web.eecs.umich.edu/~weimerw/p/weimer-icse2019-slides.pdf



Directly focused on software faults

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

Less than 12 papers so far… but the trend is growing fast.

All studies are exploratory; far from being definitive.

- "**Distilling Neural Representations of Data Structure Manipulation using fMRI and fNIRS**", Yu Huang, Xinyu Liu, Ryan Krueger, Tyler Santander, Xiaosu Hu, Kevin Leach and **Westley Weimer**, International Conference on Software Engineering (ICSE) 2019.
- "**A Look into Programmers' Heads**", Norman Peitek, **Janet Siegmund,** Sven Apel, Christian Kästner, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, André Brechmann , IEEE Transactions on Software Engineering, August, 2018.

Neuroscience perspective on software code

Ricardo Couceiro
CISUC, University of Coimbra
Coimbra, Portugal

João Castelhano
ICNAS, University of Coimbra
Coimbra, Portugal

Miguel Castelo Branco
ICNAS/CIBIT, University of Coimbra
Coimbra, Portugal

Gonçalo Duarte
CISUC, University of Coimbra
Coimbra, Portugal

Catarina Duarte
ICNAS, University of Coimbra
Coimbra, Portugal

Paulo de Carvalho
CISUC, University of Coimbra
Coimbra, Portugal

João Durães
CISUC, Polytechnic Institute of Coimbra
Coimbra, Portugal

César Teixeira
CISUC, University of Coimbra
Coimbra, Portugal

Henrique Madeira
CISUC, University of Coimbra
Coimbra, Portugal

SW reliability people

Artificial intelligence people

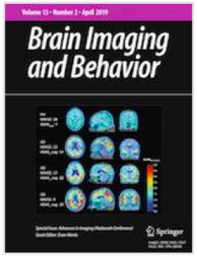
Biomedical Engineers

Neuroscientists

Henrique Madeira, DEI-FCTUC, 2019

Brain network underlying human errors in SW development activities

Springer Link



Brain Imaging and Behavior
pp 1–15 | Cite as

The role of the insula in intuitive expert bug detection in computer code: an fMRI study

Authors | Authors and affiliations

Joao Castelhano, Isabel C. Duarte, Carlos Ferreira, Joao Duraes, Henrique Madeira, Miguel Castelo-Branco

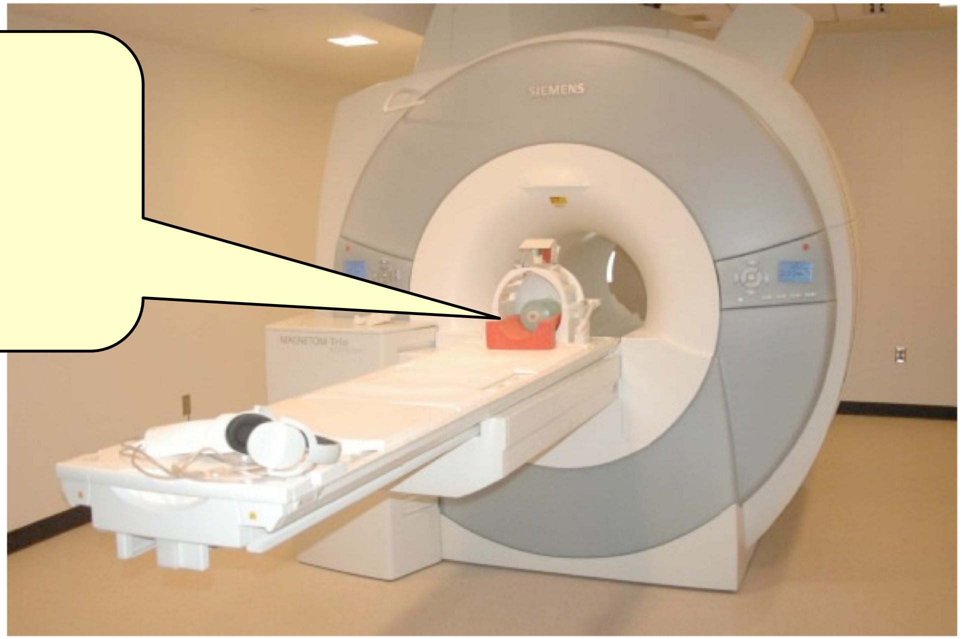
Open Access | Original Research
First Online: 09 May 2018

Henrique Madeira, DEI-FCTUC, 2019

Experimenting using fMRI?
What should we look out for?

Added features

- Screen
- Eye tracking
- Joystick



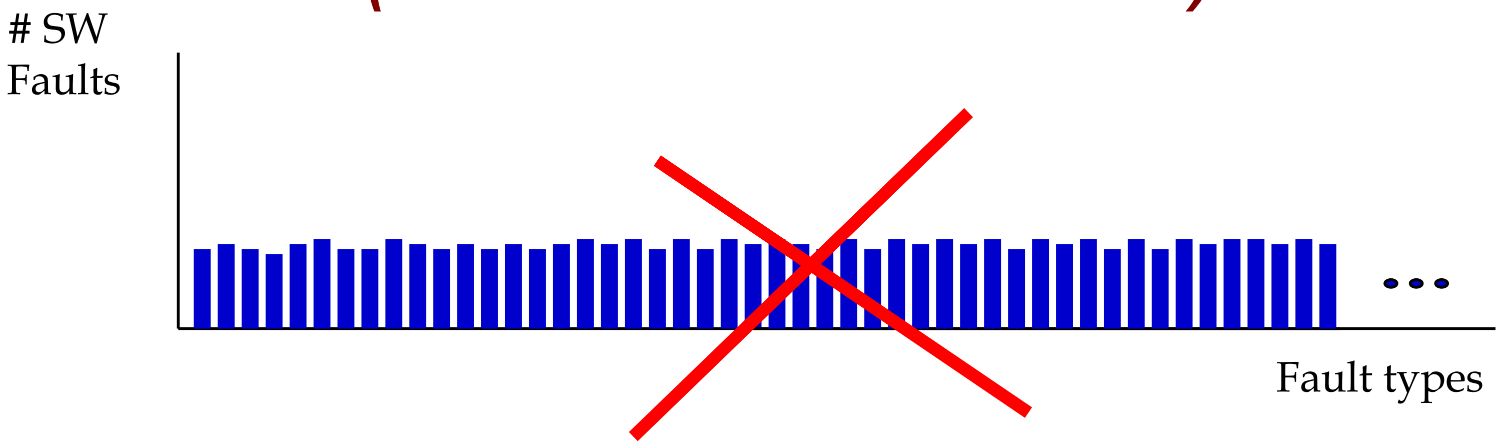
3T Magnetom Trio Tim MRI scanner, 12-channel head coil (Siemens)

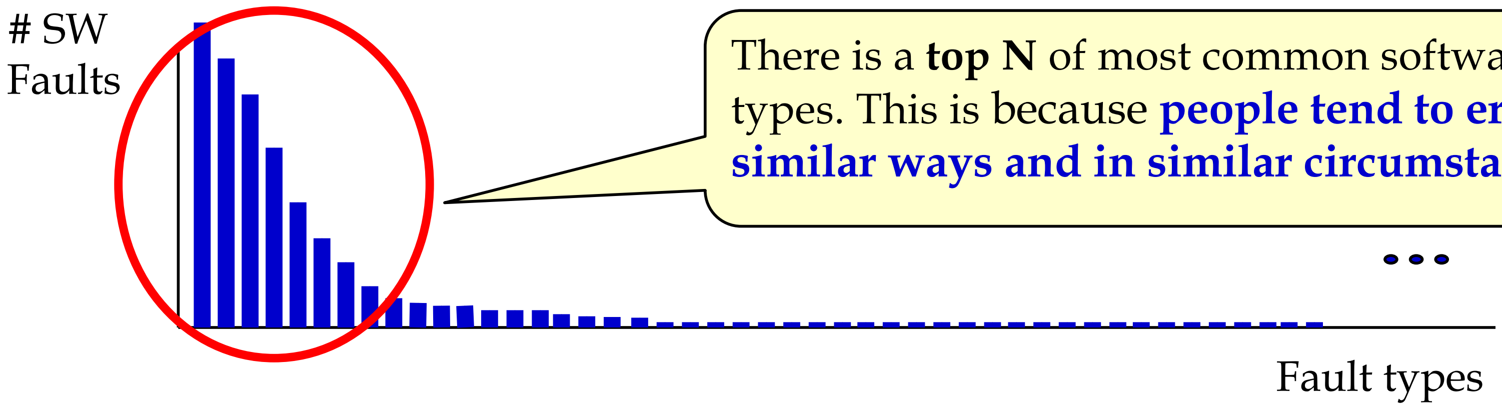
Anatomical images acquired using MPRAGE sequence with resolution of 1 mm^3

Functional analysis done using BrainVoyager QX 2.8 (BrainInovation)

Henrique Madeira, DEI-FCTUC, 2019

Fault models for software faults
(results from field studies)

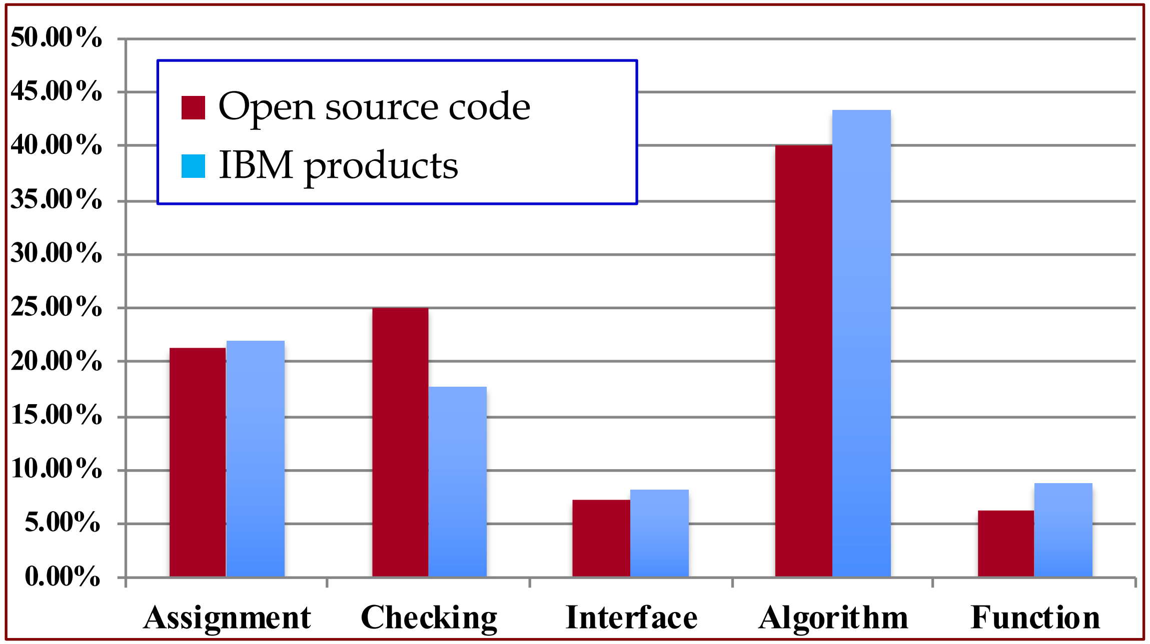




There is a **top N** of most common software fault types. This is because **people tend to err in similar ways and in similar circumstances.**

Henrique Madeira, DEI-FCTUC, 2019

People fail in similar ways and similar circumstance



Field studies:
ODC classification of software faults found in deployed software

Henrique Madeira, DEI-FCTUC, 2019

Different environments, different cultures, different development processes, different systems and applications, different programming languages, etc., etc…

→ but apparently similar error patterns; **people is the only common element**

Experimenting using fMRI? What should we look out for?

There is in fact a small number of most frequent types of bugs and error prone scenarios → **This is our focus**

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 32, NO. 11, NOVEMBER 2006 849

Emulation of Software Faults: A Field Data Study and a Practical Approach

João A. Durães, *Member*, *IEEE*, and Henrique S. Madeira, *Member*, *IEEE*

Field study on software faults

Proceedings of the 18th ISSAT International Conference on Reliability and Quality in Design
July 26-28, 2012 - Boston, Massachusetts, U.S.A.

A Taxonomy System to Identify Human Error Causes for Software Defects

A. Fuqun Huang, B. Bin Liu, and C. Bing Huang

School of Reliability&System Engineering
Beihang University
Beijing, China
Email: huangfuqun@gmail.com

Cognitive psychology perspective on software faults

Functional Magnetic Resonance Imaging (fMRI)

- **fMRI** uses the magnetic properties of blood to analyze brain activity in specific areas.

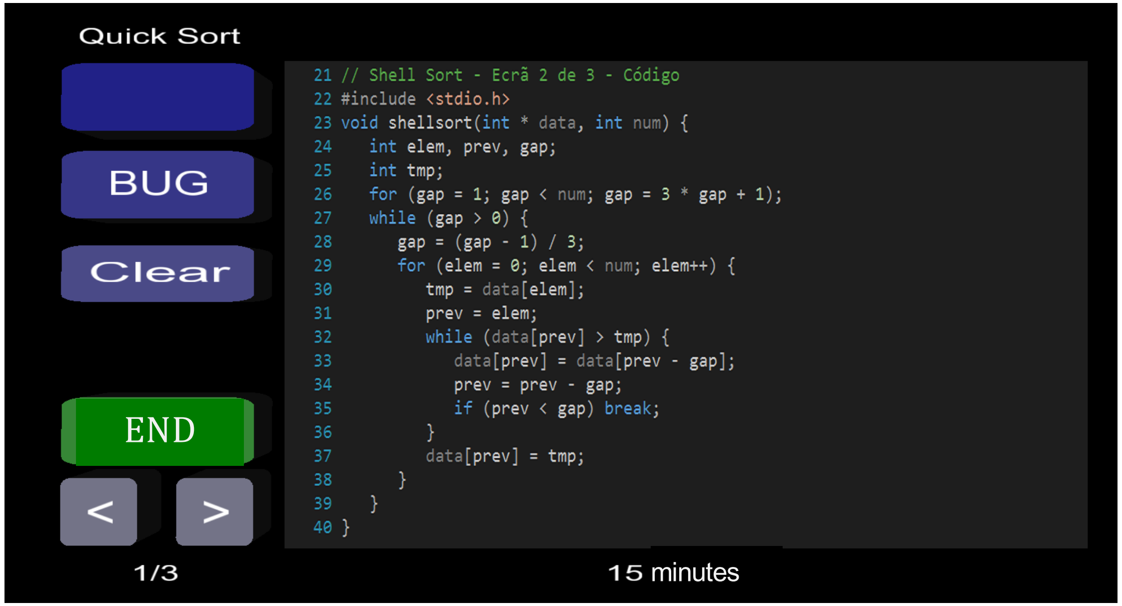
- BOLD (Blood Oxygen Level-Dependent) imaging.
- Creates highly detailed 3D images of the brain in successive instants (sampling 2 seconds)
- Active areas of the brain are detected by filtering out the active voxels, when compared to a base level activity (i.e. fMRI is a differential technique).

  To find the brain areas that are active in searching for bugs we need to filter out the active brain areas when the participant is just reading and understanding the code (and vision areas, movement, etc.).

Experiment protocol overview

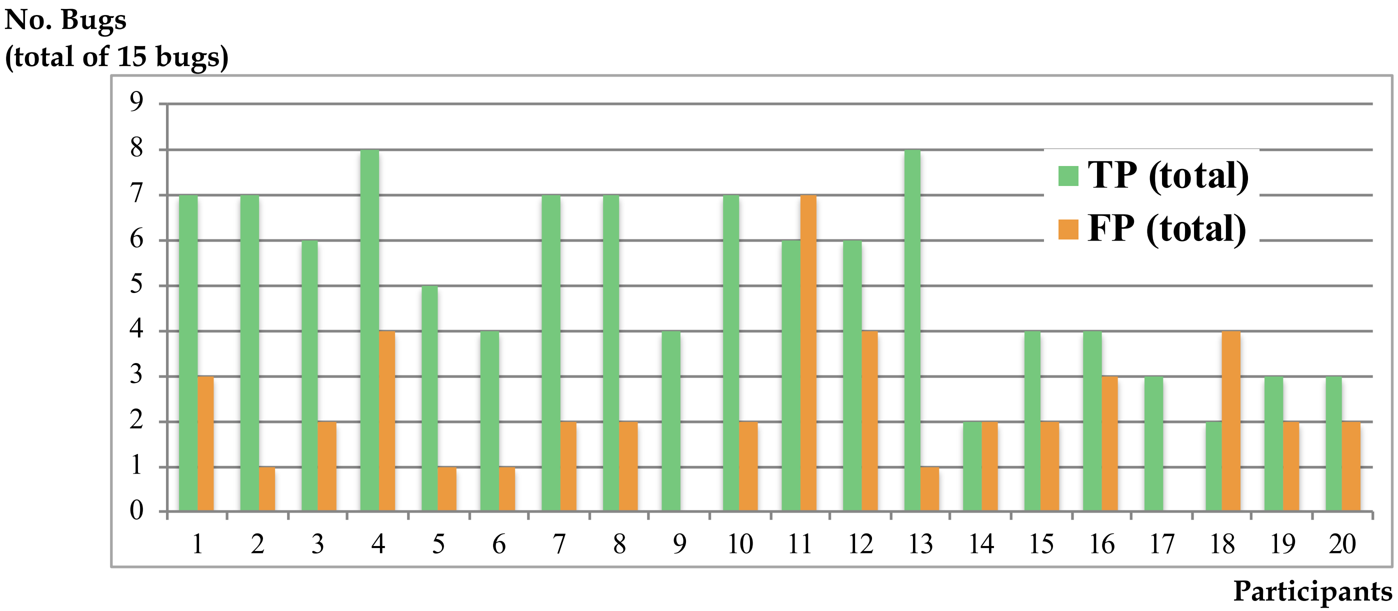
- Group of volunteers (experienced and very experienced programmers) are asked to do a code inspection inside a **fMRI system** (20 volunteers)
- Three simple programs in C: quick sort, shell sort and matrix multiplication. Consistent in size with the amount of code addressed in typical Fagan's inspections.
- The programs contain a small number of realistic bugs (using the Top N most frequent bugs types), inserted beforehand (a total of 15 bugs) (some other programs are used to create the baseline for contrast).
- The algorithm and pseudo code is explained to the volunteers before the experiment (as in Fagan's inspections; but the inspection itself is individual).
- Each volunteer analyzes the code inside the fMRI:
  - Records the bugs he/she founds
  - Corrections are allowed (i.e., clear a bug indication)
  - The eye tracking is synchronized with the fMRI (same time scale)
  - After the session inside the fMRI, the volunteer indicates the level of confidence he/she has on the each bug identified

Example of the screen available for the volunteers



- The cursor is controlled by a joystick (with an “enter button”)
- Brain activity related to movements, vision, hearing, etc. is filtered out by software.

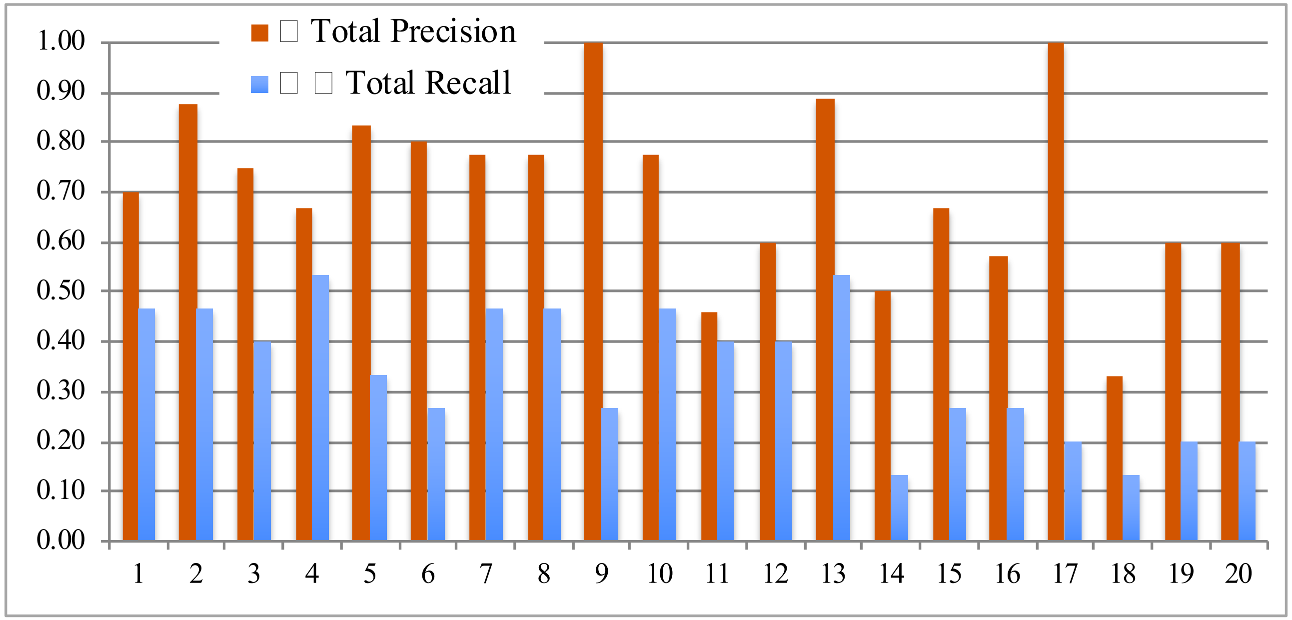
Code inspection results: True positives and false positives



True Positive (TP) – Bugs correctly identified (i.e., correspond to bugs inserted in the programs)

False Positive (FP) – Bugs incorrectly identified (i.e., do not correspond to bugs inserted)

Code inspection results: precision and recall



Precision = TP / (TP + FP) → **Average Precision = 0.6959** **Stdev = 0.174**

Recall = TP / Total real bugs → **Average Recall = 0.3433**

Stdev = 0.132

Henrique Madeira, DEI-FCTUC, 2019

Where are we looking at?

Neuroscience perspective:
(Brain activity in highly abstract tasks was not much investigated)

- Are there specific brain areas responsible for bug detection?
- Is there a specific area (or network) responsible for the "eureka moment" of finding a bug?
- Is the suspicion of bug different from bug confirmation?
- Is the sense of an uncertain feeling in the presence of a bug related to specific brain areas?
- What happens in the brain when an expert looks at the lines of code where a bug is and does not suspect nor detect the bug?

on and

did not

Where are we looking at?

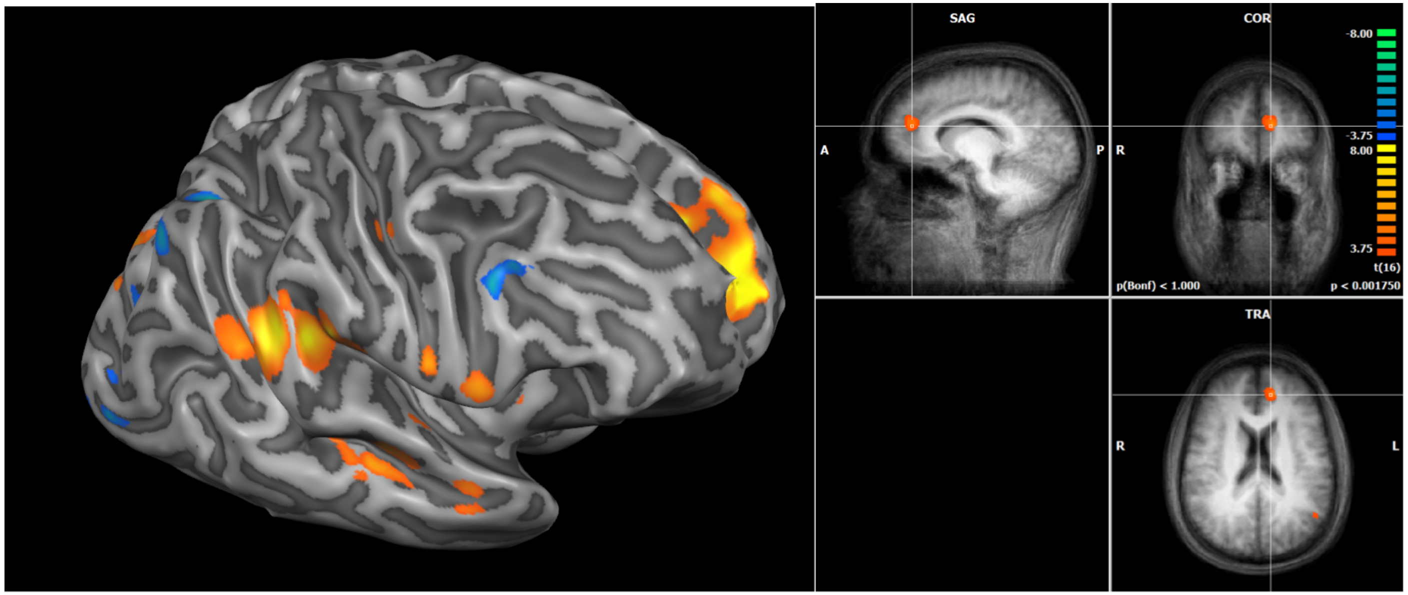
Software reliability perspective:

- Why do some people see a given bug while others don't?
- Why is the precision in code inspections relatively low?
- What can we do to improve the chances of spotting more bugs during program coding (and during testing)?
- Can we measure (estimate) participants skills using fMRI results?
- Can we measure cognitive load (amount of "mental effort") when reading and understanding a program snippet?
- Can we correlate "mental effort" with software complexity metrics?

n and

id not

Sample of fRMI image: bug confirmation



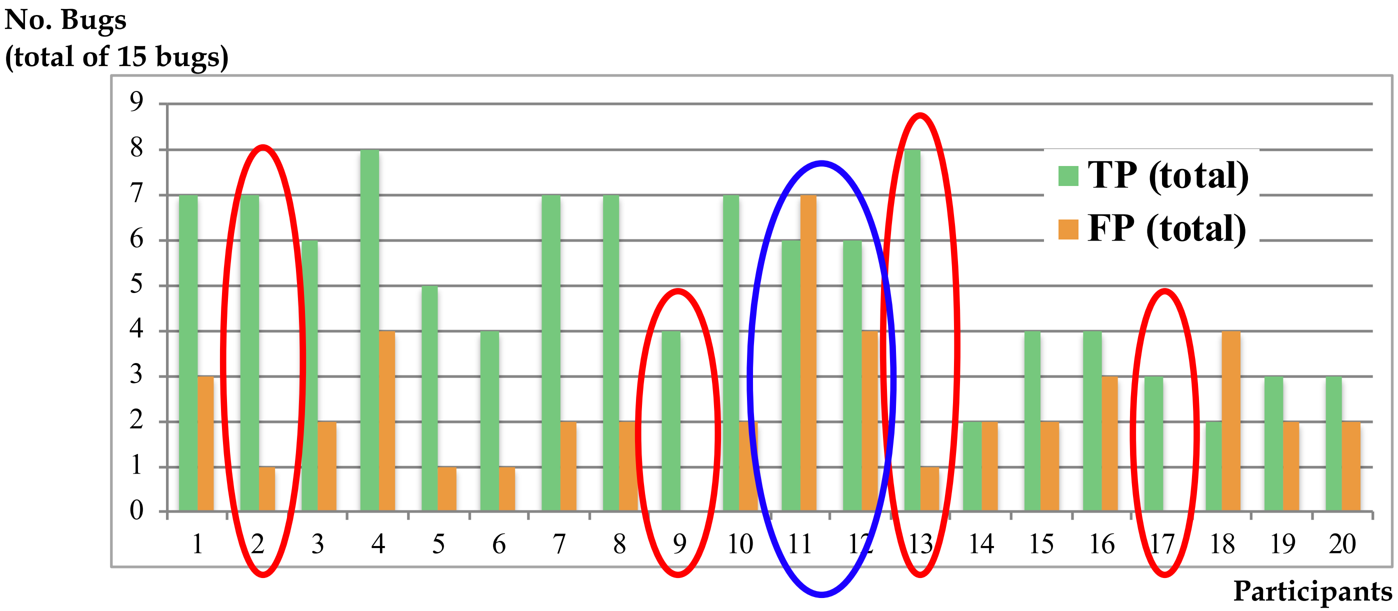
The BOLD activated areas at the moment of bug confirmation.

fMRI results summary

Contrast	Label	Brodmann area	Pea kX	Pea kY	Pea kZ	p	No Voxels
Bug identification vs Baseline	L Medial Frontal Gyrus	BA9	-12	44	22	0,000053	6316
	L Cuneus	BA18	-3	-94	4	0,000167	939
	L Insula	BA13	-39	14	10	0,001095	446
	L Superior Temporal Gyrus	BA39	-54	-55	28	0,000955	1841
Suspicious vs Bug	R Inferior Temporal Gyrus	BA19	45	-54	-2	0,000393	1769
	R Insula	BA13	45	8	-2	0,000188	1393
	R Inferior Occipital Gyrus	BA19	36	-79	-5	0,000249	1664
Code with bugs vs Neutral (code reading; no bugs)	R Middle Frontal Gyrus	BA8	51	8	40	0,000008	1208
	R Precuneus	BA19	30	-61	37	0,000001	721
	R Lingual Gyrus	BA17	15	-94	-14	0,000001	510
	L Precuneus	BA19	-27	-70	40	0,000008	1791
	L Inferior Occipital Gyrus	BA18	-33	-85	-14	0,000008	570

Insula is a region critically involved in the processing of error uncertainty during bug monitoring and programming decision.

Code inspection results: True positives and false positives



True Positive (TP) – Bugs correctly identified (i.e., correspond to bugs inserted in the programs)

False Positive (FP) – Bugs incorrectly identified (i.e., do not correspond to bugs inserted)

Some things we can see directly through fMRI

- The distinct role for the insula in bug monitoring and detection and a novel connectivity pattern related to the quality of error detection (first step for dicovering the brain activation patterns for the eureka moment of bug finding).

- "Mental effort" while reading/understanding the code, and consequently the correlation between mental effort and software complexity metrics.

- Activation of specific brain regions (e.g., language, mathematical, decision taking, in addition to the already known areas associated to code comprehension) and activation patterns such as attention patterns. This can be combined with eye tracking to provide fine grain analysis.

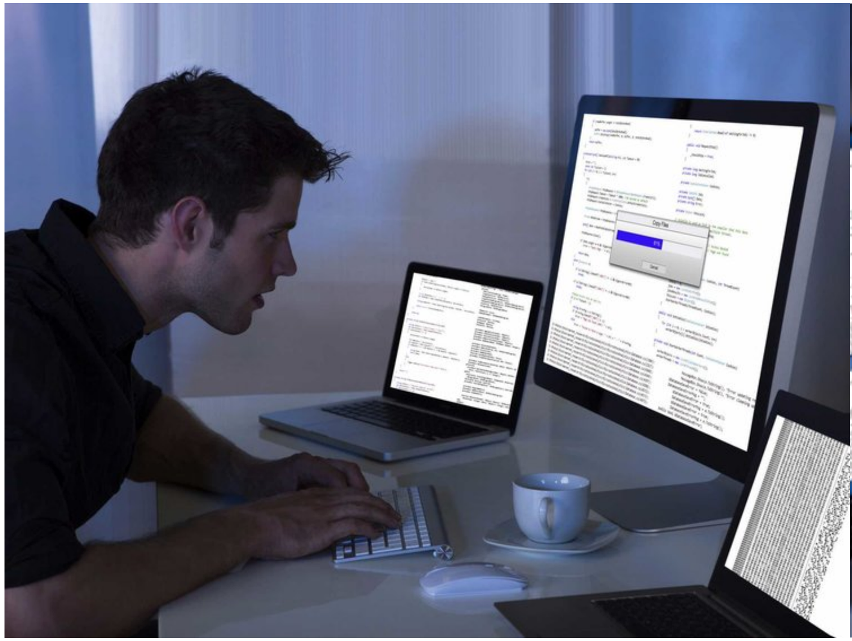
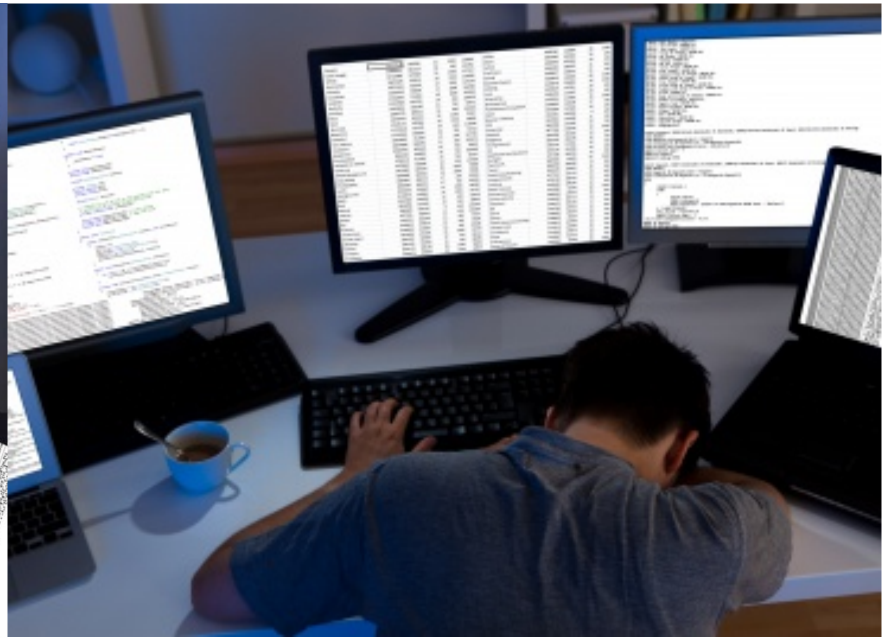
- Estimation/measurement of proficiency in the programming language

Expectations and some tangible results

- Biofeedback Augmented Software Engineering
- Intelligent code biofeedback annotation using HRV and pupillography

Software faults are human faults

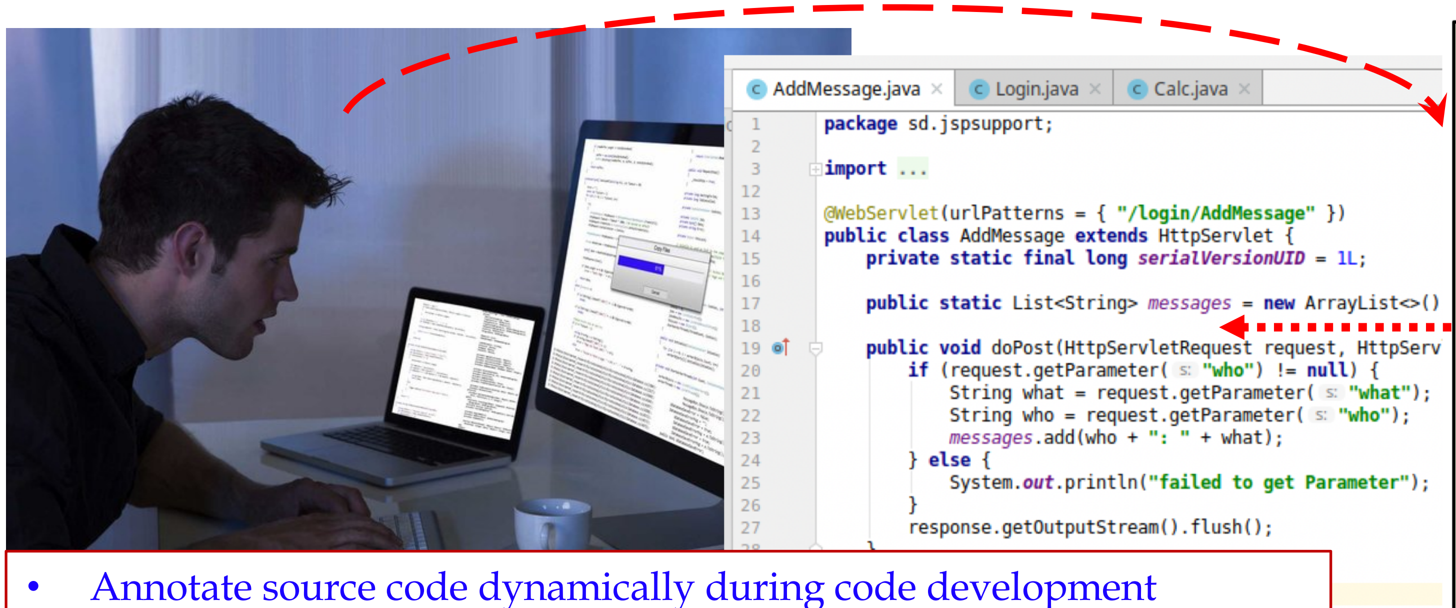
Biofeedback Augmented Software Engineering

Henrique Madeira, DEI-FCTUC, 2019

Biofeedback Augmented Software Engineering

Key step: biofeedback code annotation



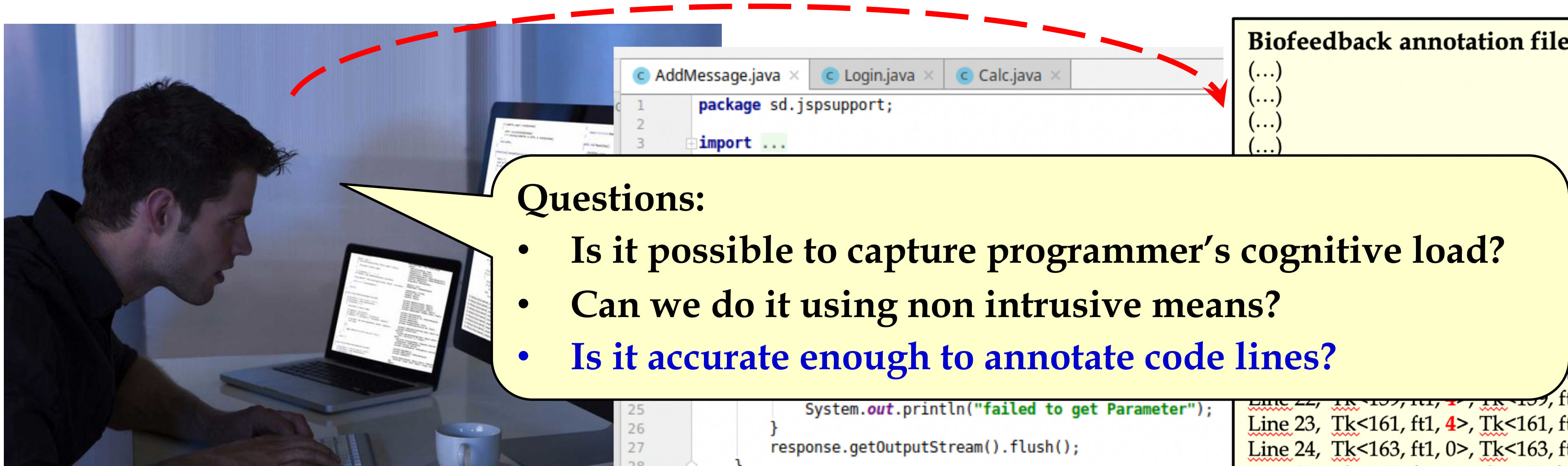
Biofeedback annotation file

(…)
(…)
(…)
(…)
(…)
Line 14, Tk<123, ft1, 0>, Tk<123, ft2, 1>,
Line 15 Tk<134, ft1, 0>, Tk<134, ft2, 0>,
Line 16, Tk<138, ft1, 0>, Tk<138, ft2, 0>,
Line 17, Tk<143, ft1, 0>, Tk<143, ft2, 0>,
Line 18, Tk<145, ft1, 0>, Tk<145, ft2, 0>,
Line 19, Tk<147, ft1, 0>, Tk<147, ft2, 1>,
Line 20, Tk<153, ft1, 0>, Tk<153, ft2, 1>,
Line 21, Tk<156, ft1, 3>, Tk<156, ft2, 3>,
Line 22, Tk<159, ft1, 4>, Tk<159, ft2, 3>,
Line 23, Tk<161, ft1, 4>, Tk<161, ft2, 4>,
Line 24, Tk<163, ft1, 0>, Tk<163, ft2, 1>,
Line 25, Tk<165, ft1,1>, Tk<165, ft2, 0>,
Line 26, Tk<168, ft1, 0>, Tk<168, ft2, 0>,
Line 27, Tk<170, ft1, 0>, Tk<170
Line 28, Tk<173, ft1, 0>, T
Line 29, Tk<174, ft1, 0

- Annotate source code dynamically during code development
- Annotations reflect programmers' cognitive load, stress level, etc.
- Annotations linked to lexical tokens; all lines of code are annotated.

Biofeedback Augmented Software Engineering

Key step: biofeedback code annotation



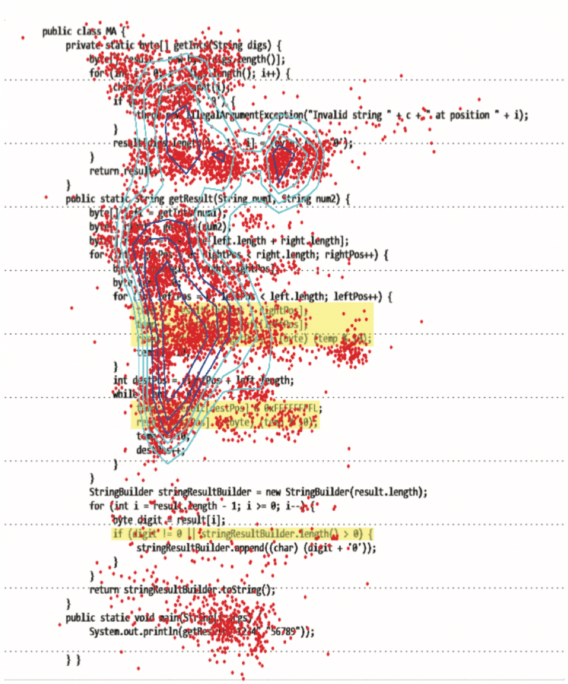
Biofeedback annotation file

(…)
(…)
(…)
(…)
…2, 1>,
…2, 0>,
…2, 0>,
…2, 0>,
…2, 0>,
…2, 1>,
…2, 1>,
…2, 3>,
…, ft2, 3>,
Line 23, Tk<161, ft1, 4>, Tk<161, ft2, 4>,
Line 24, Tk<163, ft1, 0>, Tk<163, ft2, 1>,
Line 25, Tk<165, ft1,1>, Tk<165, ft2, 0>,
Line 26, Tk<168, ft1, 0>, Tk<168, ft2, 0>,
Line 27, Tk<170, ft1, 0>, Tk<170,
Line 28, Tk<173, ft1, 0>, T
Line 29, Tk<174, ft1, 0

- Annotate source code dynamically during code development
- Annotations reflect programmers' cognitive load, stress level, etc.
- Annotations linked to lexical tokens; all lines of code are annotated.

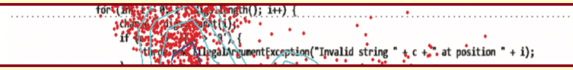
How can we gather programmer's cognitive state?

Examples o wearable and low intrusive devices that can capture **autonomic nervous systems manifestations that could be related to cognitive load**



How can we gather programmer’s cognitive state?

Problem:
These sources have noise and are sensitive to stress conditions totally unrelated to the software development activities

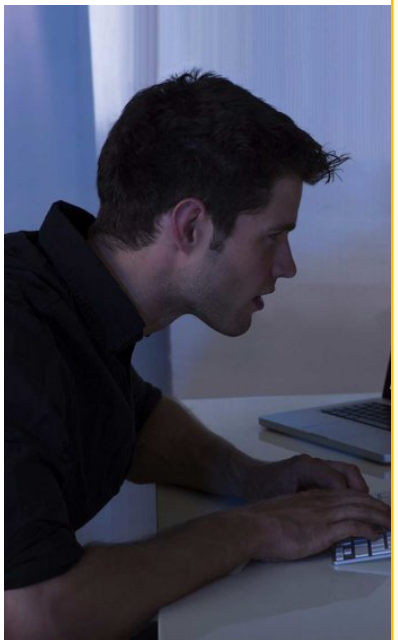


In this experiment…

- We assess the possibility of using **pupillography** and **HRV** as indicators of programmers’ mental effort and cognitive load.
- Pupillography is reasonably immune to noise and extraneous conditions.
- Pupillography is non intrusive.
- HRV is low intrusive.

Biofeedback Augmented Software Engineering

What can we do if we have accurate code annotations reflecting programmer's cognitive state?
(annotation represent cognitive load such as mental effort, stress, attention levels, fatigue, etc.)



- **Biofeedback code highlighting** to provide **online warning of the programmer** by highlighting the lines of code that may have bugs and need a second look from the programmer.
- **Biofeedback-driven software testing** to optimize testing effort by taking into account the individual information gathered from each programmer that has participated in the code development.
- **Improved models of bug density estimation and SW risk analysis,** through the use of additional information on programmer's emotional and cognitive states, in conjunction to code complexity metrics and test coverage
- **Programmers' friendly integrated development environments** with automatic warning/enforcement of programmers' resting moments, when accumulated signs of fatigue and mental strain show that not only the code quality is doubtful but, above all, programmers' mental well-being must be protected.
- **Biofeedback optimized training needs** through the creation of individual programmer's profiles to help define training plans based on the biofeedback metadata.
- (there are more)

Proposed experiment

- **Goal**: assess the possibility of using **pupillography** and **HRV** as indicators of programmers' mental effort and cognitive overload.
- Focused on program comprehension (such as in a code inspection)
- Answer the following question: **is it possible to know if a programmer is reading complex or simple code through the analysis of the pupillography signal? The same for HRV.**
- A glimpse of very recent results showing that pupillography and HRV are accurate enough to allow the annotation of specific code lines

Experiment outline

Controlled experiment: programmers was asked to perform 4 tasks

Control task — Program (Java) comprehension tasks

Control task	C1	C2	C3
Reading natural language (60 sec)	**Counts the number of values in an array that fall within a given interval.**	**Multiplies two numbers using the classic weighed digits algorithm**	**Search 3 dimensional objects in a 3 dimension space**

- 30 volunteers (24 male, 6 female, age: 24.4 ± 6.18 yrs, 12 intermediate, 14 advan
- Instru A)
- The t code.
- After ons to confirm if he/she understood the program + a **NASA TLX survey**

Program	Lines of code	Nested Block Depth	No. params.	Cyclomatic complexity
C1	13	2	3	3
C2	42 (12+30)	3	3	3 + 6
C3	49	5	4	15

Henrique Madeira, DEI-FCTUC, 2019

Experiment outline

Controlled experiment: programmers was asked to perform 4 tasks

Control task | Program (Java) comprehension tasks

	C1	C2	C3

Some features of the programs:

- The code style was "normalized" (variables with meaningful names, no comments, etc.)
- The code has no complex math or difficult algorithms the participants may not know → the complexity is related to the language constructs.

C1	13	2	3	3
C2	42 (12+30)	3	3	3 + 6
C3	49	5	4	15

- Instru ... A)
- The t ... code.
- After ... ons to confirm if he/she understood the program + a **NASA TLX survey**

Henrique Madeira, DEI-FCTUC, 2019

Experiment outline

Controlled experiment: programmers was asked to perform 4 tasks

Control task

Reading natural language (60 sec)

Program comprehension tasks

C1
Counts the number of values in an array that fall within a given interval.

C2
Multiplies two numbers using the classic weighed digits algorithm

C3
Search 3 dimensional objects in a 3 dimension space

Goal: measure cognitive load while comprehending code.

Is it possible to identify the program a volunteer is looking at through the analysis of the pupillography and HRV signals?

Experiment protocol

Steps

1. **Baseline →** empty grey screen with a black cross in its center for 30 seconds.
2. **Reference activity →** text in natural language to be read by the participant (60 seconds max.).
3. **Baseline →** empty grey screen with a black cross in its center for 30 seconds.
4. **Code comprehension task → s**creen displays the code of the program to be analyzed for code comprehension. This step lasts up to 10 minutes maximum for each program.
5. **Empty grey screen** with a black cross for 30 seconds.
6. Repeat steps 4 and 5 program by program (C1, C2, C3)
7. **Survey 1**: NASA-TLX to assess the subjective mental effort perceived by each participant in the code comprehension.
8. **Survey 2**: check understanding of the program.

NASA-TLX results

Control task

Reading natural language (60 sec)

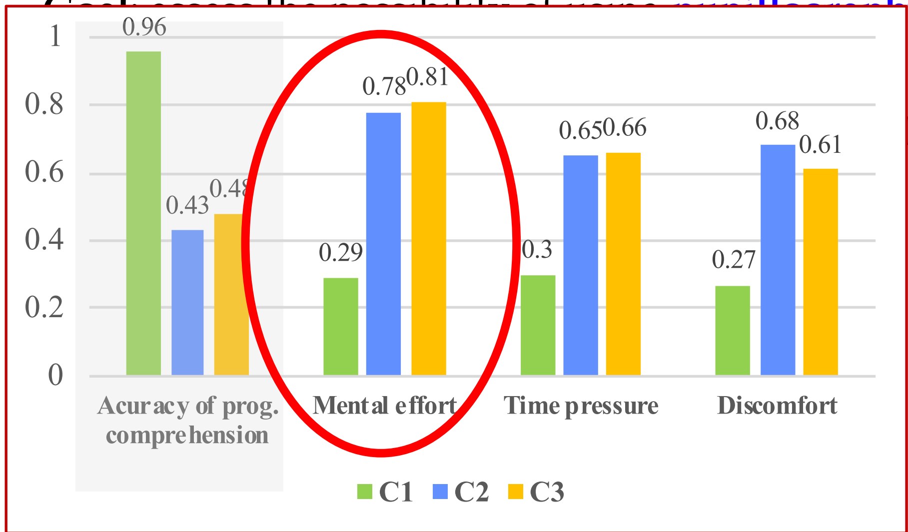
Program comprehension tasks

C1
Counts the number of values in an array that fall within a given interval.

C2
Multiplies two numbers using the classic weighed digits algorithm

C3
Search 3 dimensional objects in a 3 dimension space

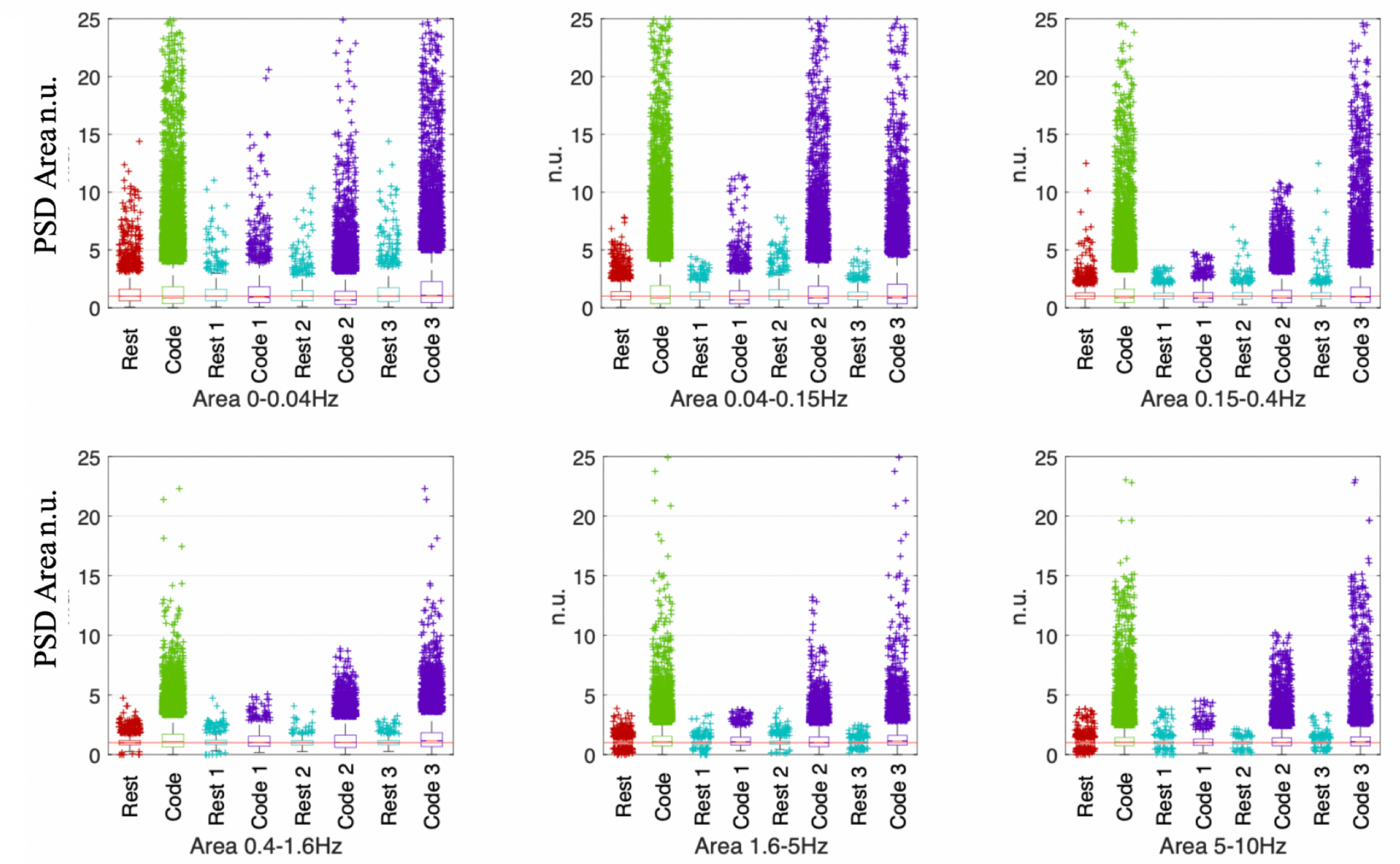
Subjective code complexity measured using NASA TLX



Program	Lines of code	Nested Block Depth	No. params.	Cyclomatic complexity
C1	13	2	3	3
C2	42 (12+30)	3	3	3 + 6
C3	49	5	4	15

- Participants consider the complexity of the 3 programs substantially different (especially for C1 when compared to C2 and C3)
- Code metrics do not map (always) to programmers' cognitive load. Metrics are not enough to guide testing effort.

Pupillography results



Henrique Madeira, DEI-FCTUC, 2019

Significance level results

	0 Hz to 0.04 Hz	0.04 Hz to 0.15 Hz	0.15 Hz to 0.4 Hz	0.4 Hz to 1.6 Hz	1.6 Hz to 5 Hz	5 Hz to 10 Hz
R vs C	P = 0.012					
R vs C1	P > 0.05					
R vs C2					P > 0.05	
R vs C3		P > 0.05				
C1 vs C2				P > 0.05		P = 0.017
C1 vs C3						
C2 vs C3			P = 0.011	P = 0.012		

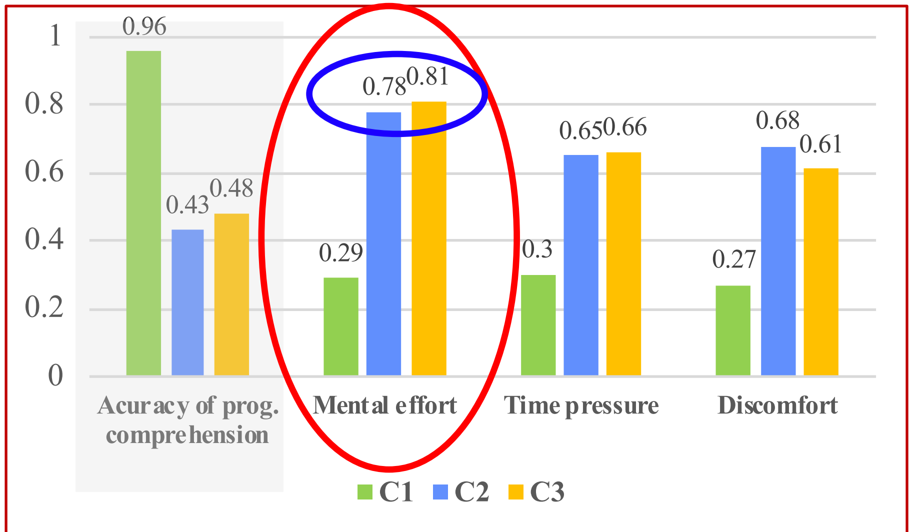
Multiple comparison tests using Kruskal-Wallis test

Green squares represent groups where the mean values of the corresponding feature are significantly different ($p<0.01$)

Encouraging results, but… to allow code annotation we need precision and **accuracy in both time and space**

HRV results

Subjective code complexity measured using NASA TLX



- **HRV results and NASA TLX provide consistent view of programmers' cognitive load.**
- **Code metrics do not not map (always) to programmers' cognitive. Metrics are not enough to guide testing effort.**

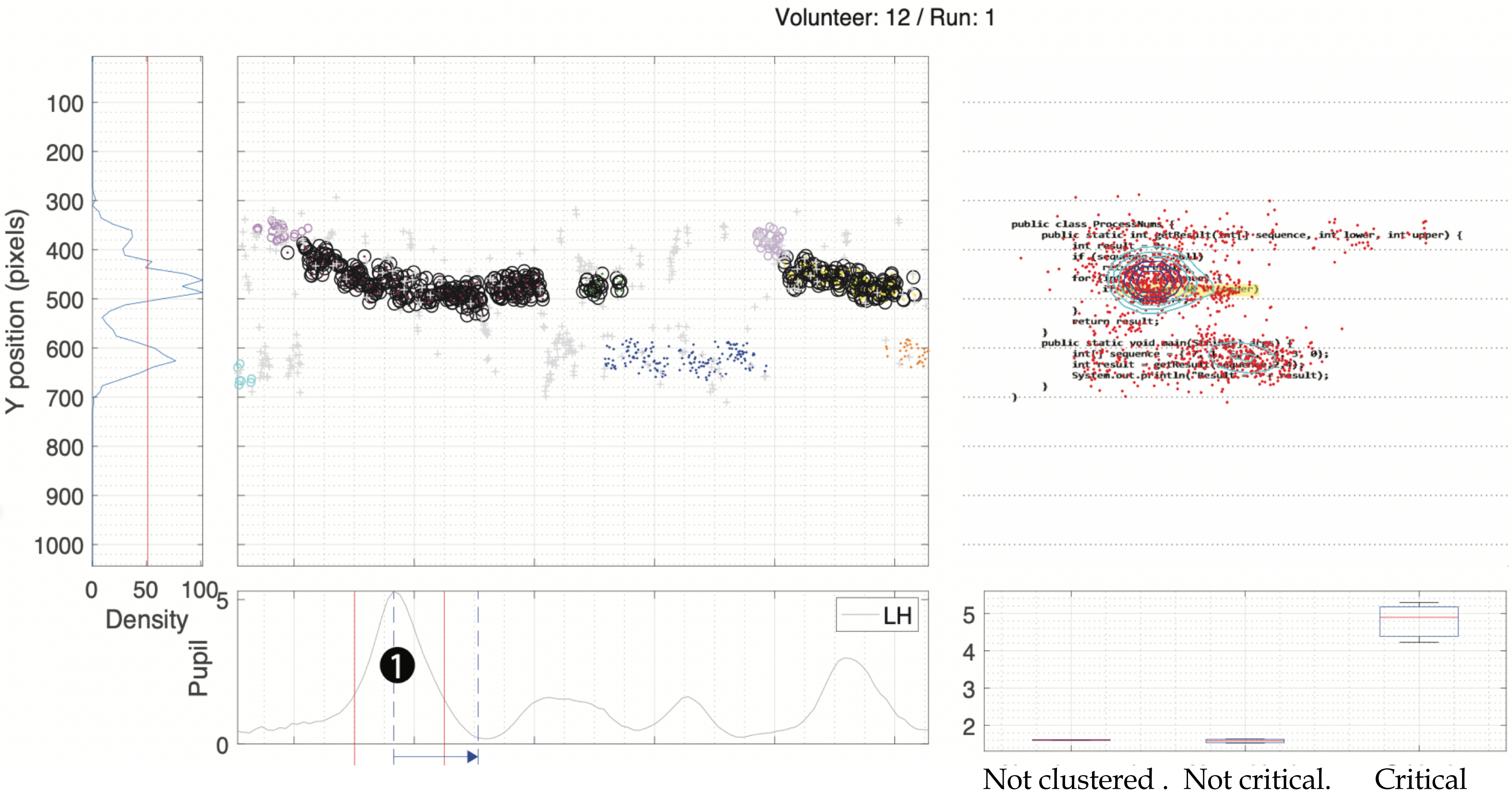
Program	Lines of code	Nested Block Depth	No. params.	Cyclomatic complexity
C1	13	2	3	3
C2	42 (12+30)	3	3	3 + 6
C3	49	5	4	15

Cognitive load measured using HRV

	Control vs any code	C1 vs C2	C1 vs C3	C2 vs C3
Sensitivity	**0.97** ± 0.06	**0.96** ± 0.14	**0.96** ± 0.20	**0.46** ± 0.38
Specificity	1 ± 0	0.81 ± 0.25	0.85 ± 0.24	0.45 ± 0.42

Accuracy of pupillography (time and space)

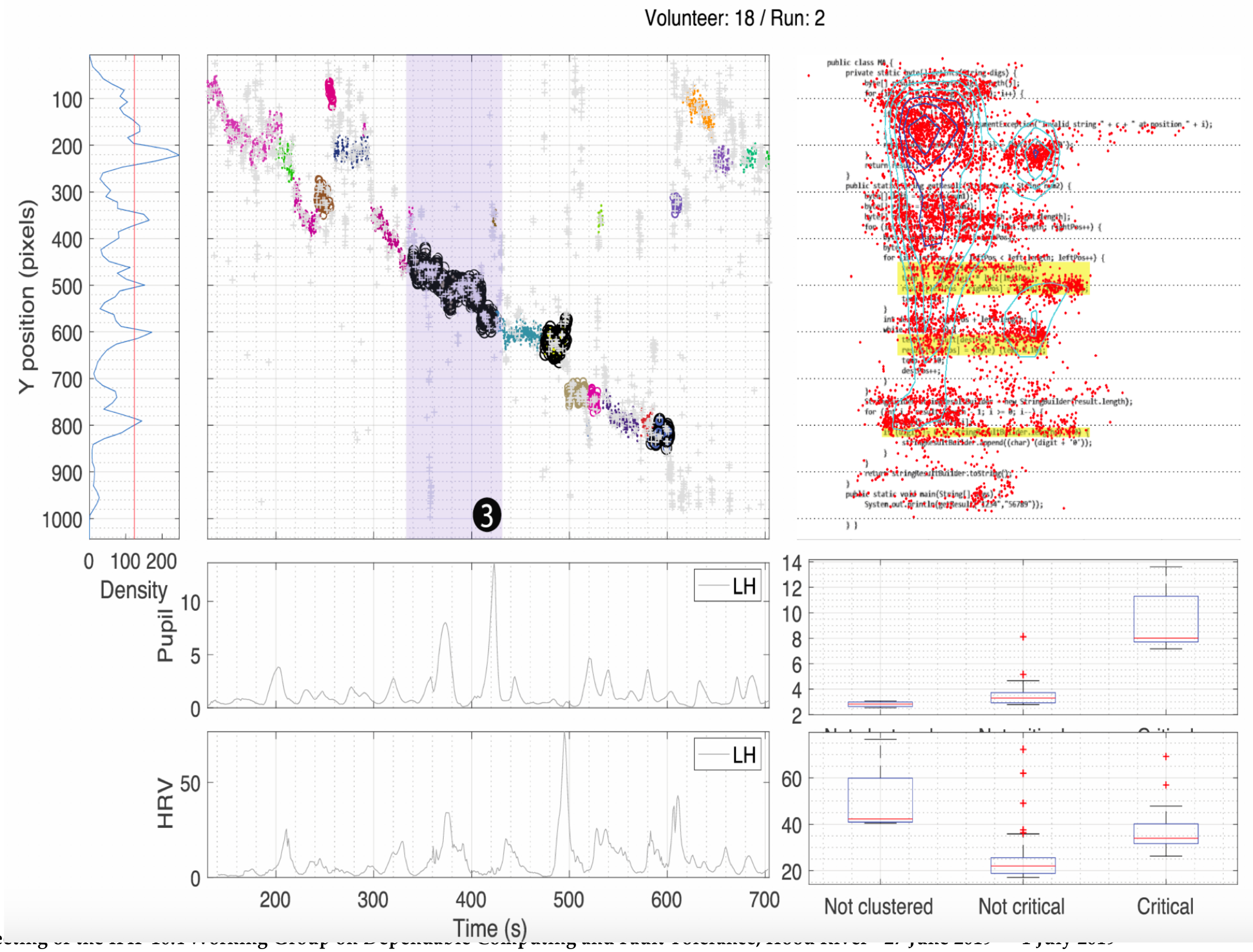
(just a glimpse of very recent results…)



Henrique Madeira, DEI-FCTUC, 2019

Accuracy of pupillography (time and space)

(just a glimpse of very recent results…)



ical. Critical

Henrique Madeira, DEI-FCTUC, 2019

Summary

- Biofeedback Augmented Software Engineering
  - A new research approach with many, many, many research questions
  - **Key →** accurate biofeedback code annotation at code line level representing metadata on the cognitive state of the programmer
  - Many potential utilizations
- Can we monitor cognitive load using available (and simple) biofeedback technology such as pupillography and HRV (and eye tracking)?
  - Apparently YES
  - Not yet fully clear if the precision in time and domain space is good enough to annotate code at code line and token level
  - Pupillography is moderately susceptibility to noise (causes not related to code development) that need to be evaluated
  - Pupillography + HRV + eye tracking should be used in conjunction