Embedded-check: a Code Quality Tool for Automatic Firmware Verification

Rafael Corsi Ferrão (Insper), Igor dos Santos Montagner (Insper), Craig Zilles(UIUC), Mariana Silva (UIUC), Rodolfo Azevedo (Unicamp)

Insper

A small private non-profit institution in São Paulo, Brazil

Scholarships + stipends for 10-15% of students

Cohort-based (no courses outside of major)
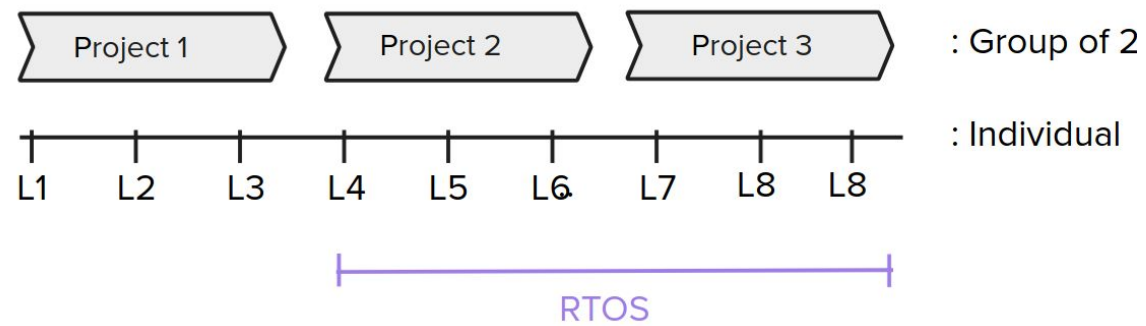
Enrollment: 50 students per semester

Context

Embedded Systems

- Undergrad course
- Offered in the fifth semester of CE
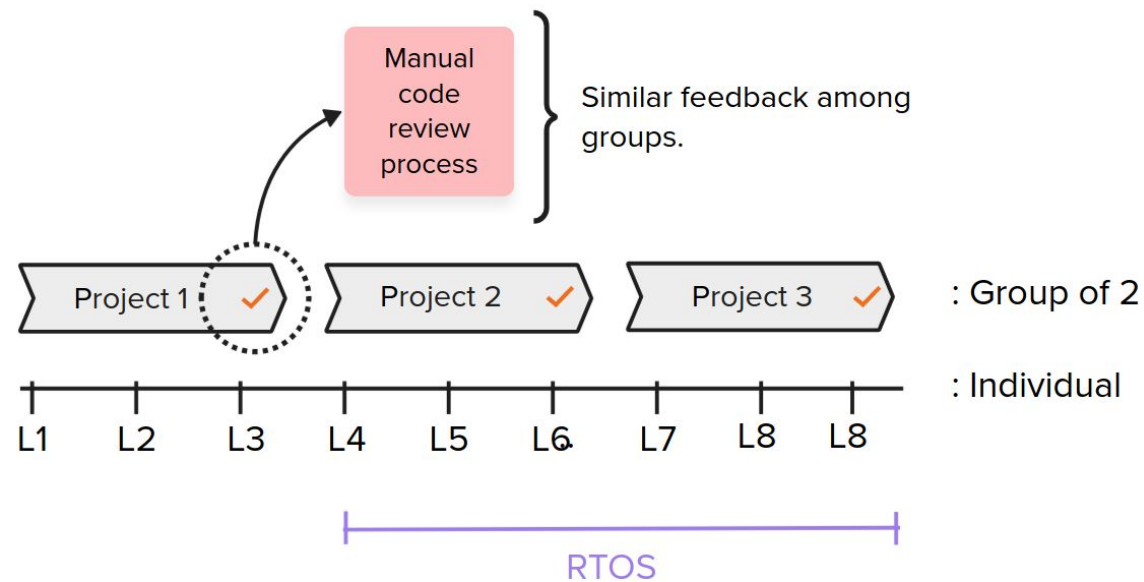- Firmware focus (C language)
- Hands-on

Software ⟷ Design ⟷ Hardware

Course details - Overview

Project 1   Project 2   Project 3   : Group of 2

L1   L2   L3   L4   L5   L6.   L7   L8   L8   : Individual

RTOS

Course details - Code review

Manual code review process

Similar feedback among groups.

Project 1 ✓   Project 2 ✓   Project 3 ✓   : Group of 2

L1   L2   L3   L4   L5   L6.   L7   L8   L8   : Individual

RTOS

Code review - Details

Code review

Project

Students modify code and
ask for validation
- github issues
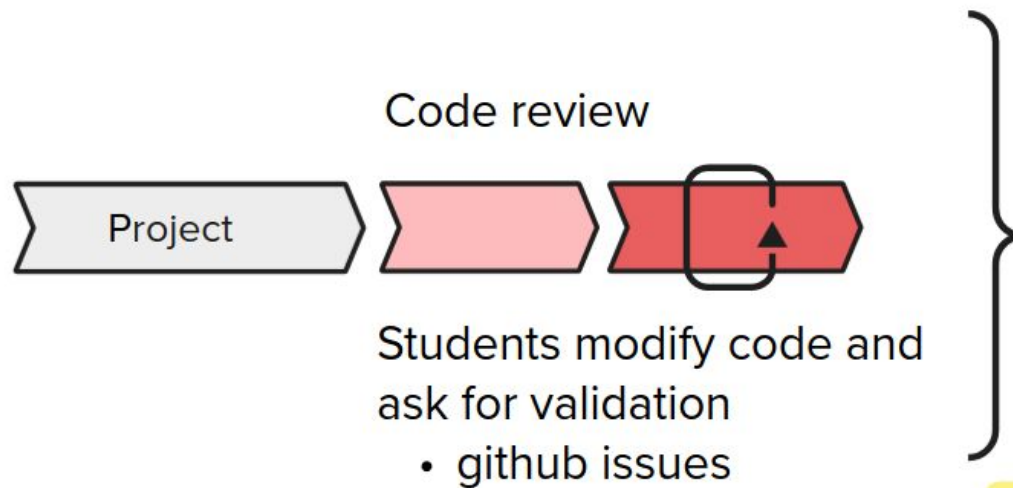
Type	Percentage
Code refactoring	55%
Rules violation	**13%**
Code malfunction	12%
Wrong rubric	7%
Documentation	5%
Solution questions	5%
Complements	3%

N = 96

3 semesters

2021 / 2022

Embedded-check: a Code Quality Tool for Automatic Firmware Verification - Rafael Corsi
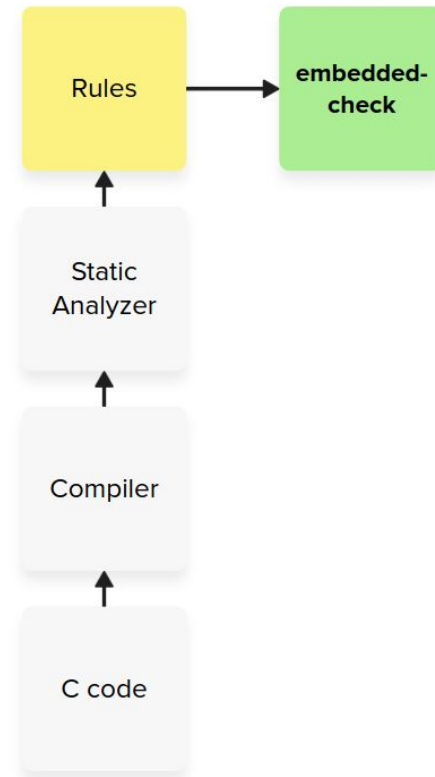
Rules

- Created based on faculty expertise
- Focus in firmware
- Three categories:
  - C Language (2 rules)
  - Embedded Systems (4 rules)
  - FreeRTOS (4 rules)
- Reflects **misconceptions**

C
Language

```c
#include <asf.h>

/**
 * freq: Frequecia em Hz
 * time: Tempo em ms que o tom deve ser gerado
 */
void tone(int freq, int time){
    double periodo_s = (double) 1 / freq;
    int periodo_us = periodo_s * 1e6;

    int n_de_iter = time / (periodo_s * 1000);
    int cont = 0;
    while(cont < n_de_iter) {
        if (!but_pause_flag) {
            pio_set(BUZZER_PIO, BUZZER_PIO_IDX_MASK);
            delay_us(periodo_us / 2);
            pio_clear(BUZZER_PIO, BUZZER_PIO_IDX_MASK);
            delay_us(periodo_us / 2);
            cont++;
        }
    }
}

// Recebe freq em hz e toca essa freq pelo tempo
// definido em TIME_BUZER_TEST (padrao 5s = 5000ms)
void buzzer_test(int freq) {
    double periodo_s = (double) 1 / freq;
    int periodo_us = periodo_s * 1e6;
    int n_iter = TIME_BUZZER_TEST / (periodo_s * 1000);

    for (int i = 0; i < n_iter; i++) {
        pio_set(BUZZER_PIO, BUZZER_PIO_IDX_MASK);
        delay_us(periodo_us / 2);
        pio_clear(BUZZER_PIO, BUZZER_PIO_IDX_MASK);
        delay_us(periodo_us / 2);
    }
}

// ...
// ...
```

foo.h

C
Language

NO (or wrong) include guard

C code in head file

```c
#include <asf.h>

/**
* freq: Frequecia em Hz
* time: Tempo em ms que o tom deve ser gerado
*/
void tone(int freq, int time){
    double periodo_s = (double) 1 / freq;
    int periodo_us = periodo_s * 1e6;

    int n_de_iter = time / (periodo_s * 1000);
    int cont = 0;
    while(cont < n_de_iter) {
        if (!but_pause_flag) {
            pio_set(BUZZER_PIO, BUZZER_PIO_IDX_MASK);
            delay_us(periodo_us / 2);
            pio_clear(BUZZER_PIO, BUZZER_PIO_IDX_MASK);
            delay_us(periodo_us / 2);
            cont++;
        }
    }
}

// Recebe freq em hz e toca essa freq pelo tempo
// definido em TIME_BUZER_TEST (padrao 5s = 5000ms)
void buzzer_test(int freq) {
    double periodo_s = (double) 1 / freq;
    int periodo_us = periodo_s * 1e6;
    int n_iter = TIME_BUZZER_TEST / (periodo_s * 1000);

    for (int i = 0; i < n_iter; i++) {
        pio_set(BUZZER_PIO, BUZZER_PIO_IDX_MASK);
        delay_us(periodo_us / 2);
        pio_clear(BUZZER_PIO, BUZZER_PIO_IDX_MASK);
        delay_us(periodo_us / 2);
    }
}

// ...
// ...
```

foo.h

Embedded Systems

```c
char btn_flag;
volatile int freq;

// Hardware callBack (ISR)
void but_pause_callback()
{
    while (!btn_flag) {}
    btn_flag = 1;
    update_oled_display();
}

void update_freq(void) {
    freq++;
}

int main (void)
{
    board_init();

    while(1) {

        if (btn_flag) {
            update_freq();
            btn_flag = 0;
        }
    }
}
```

Embedded Systems

Missing volatile for ISR variable

Wrong use of volatile

Wrong use of global variables

Delay in serving interruption

while, for, stdio, display call

```c
char btn_flag;
volatile int freq;

// Hardware callBack (ISR)
void but_pause_callback()
{
    while (!btn_flag) {}
    btn_flag = 1;
    update_oled_display();
}

void update_freq(void) {
    freq++;
}

int main (void)
{
    board_init();

    while(1) {

        if (btn_flag) {
            update_freq();
            btn_flag = 0;
        }
    }
}
```
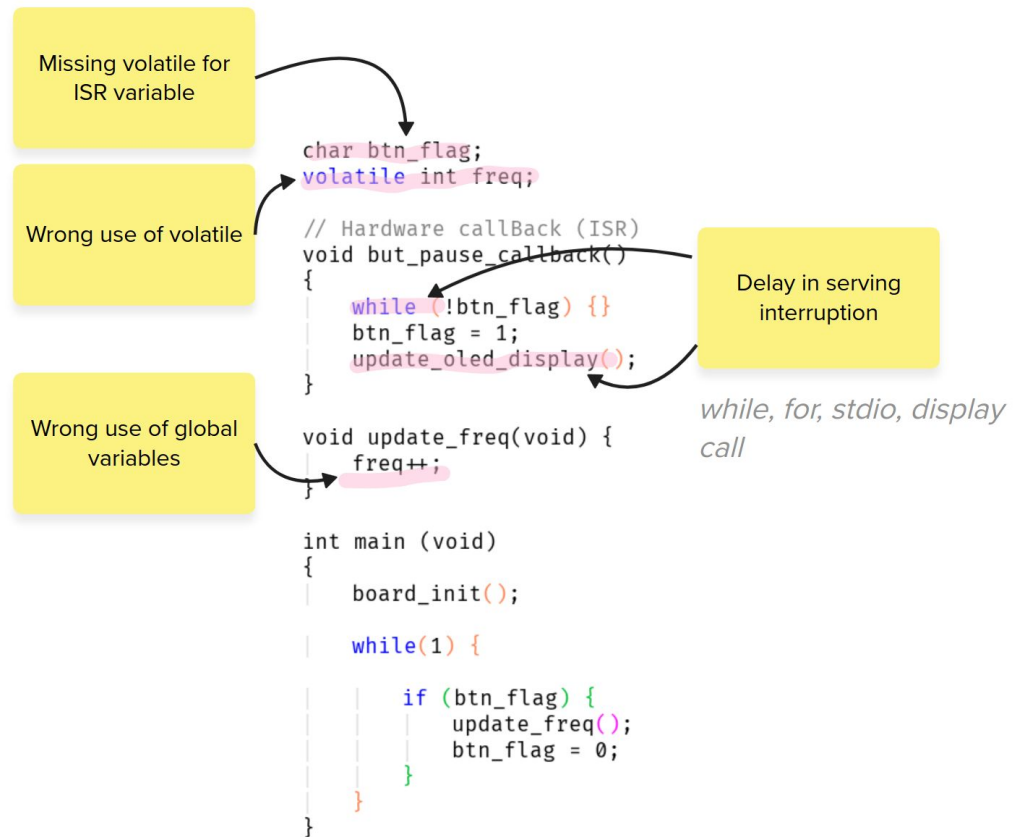
RTOS
(FreeRTOS)

```c
volatile char btn_flag;

// Hardware callBack (ISR)
void btn_pause_callback()
{
    btn_flag = 1;
}

void timer_irs(){
    xSemaphoreGive(xSemaphore, 0);
}

void task_1(void){
    int timer = 0;
    while(1) {
        if (xSemaphoreTake(xSemaphore, 0)) {
            timer++;
            xQueueSendFromISR(xQueue, &timer, 0);
        }
    }

}

void task_2(void){
    while(1) {
        if (btn_flag) {
            delay_ms(100);
            update_oled_display();
            btn_flag = 0;
        }
    }
}
```

RTOS (FreeRTOS)

Wrong use of global variables

Shall use RTOS resources such as Queue and Semaphores

Missing FromISR

Wrong use of FromISR

```c
volatile char btn_flag;

// Hardware callBack (ISR)
void btn_pause_callback()
{
    btn_flag = 1;
}

void timer_irs(){
    xSemaphoreGive(xSemaphore, 0);
}

void task_1(void){
    int timer = 0;
    while(1) {
        if (xSemaphoreTake(xSemaphore, 0)) {
            timer++;
            xQueueSendFromISR(xQueue, &timer, 0);
        }
    }
}

void task_2(void){
    while(1) {
        if (btn_flag) {
            delay_ms(100);
            update_oled_display();
            btn_flag = 0;
        }
    }
}
```

Shall use RTOS delay not software delays

embedded-check

- A static analyser tool capable of verifying all rules
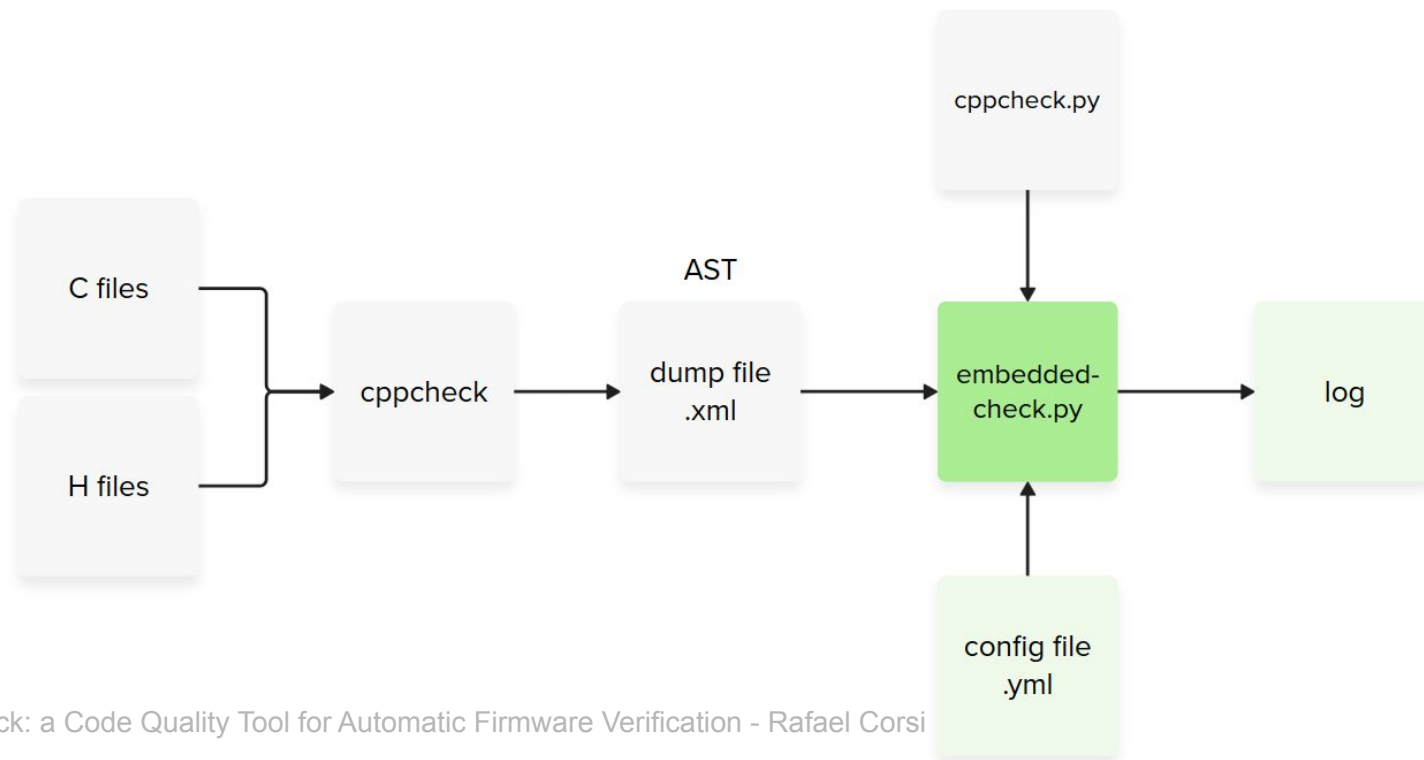
# embedded-check

Easy to extend

```python
def rule_1_3(self):
    """
    Rule 1_3: Do not use volatile where is not need
    """

    erro = 0
    var_erro_list_id = []

    for ass in self.code.all_vars_with_ass:
        if ass['variable'].Id in var_erro_list_id:
            continue

        # exclue ISR access vars
        if ass['variable'].Id in self.code.isr_global_vars_id:
            continue

        if ass['variable'].isExtern:
            continue

        if ass["variable"].isVolatile:
            var_name = ass["variable"].nameToken.str
            func_name = ass["className"]
            self.print_rule_violation(
                'rule_1_3',
                f"variable {var_name} in function {func_name}",
            )
            var_erro_list_id.append(ass['variable'].Id)
            erro = erro + 1
    return erro
```

Does it works?

- No manual errors missed by embedded-check

- Detected issues 2.6x more than manual (150 vs. 42)

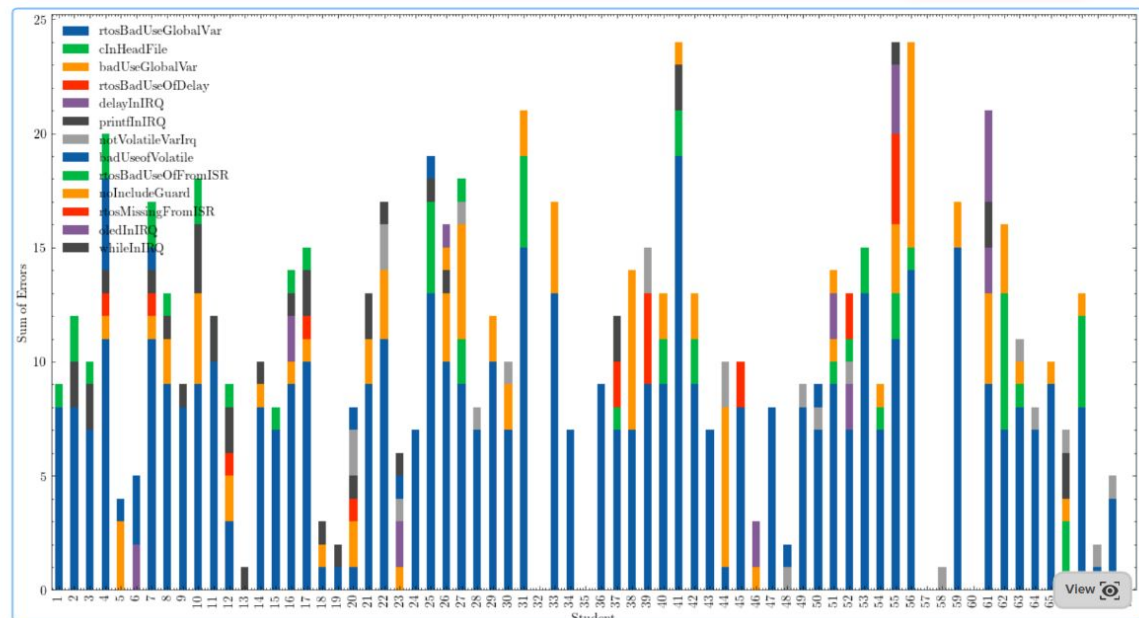- Manual detected 28% repos with errors X 86% automatic
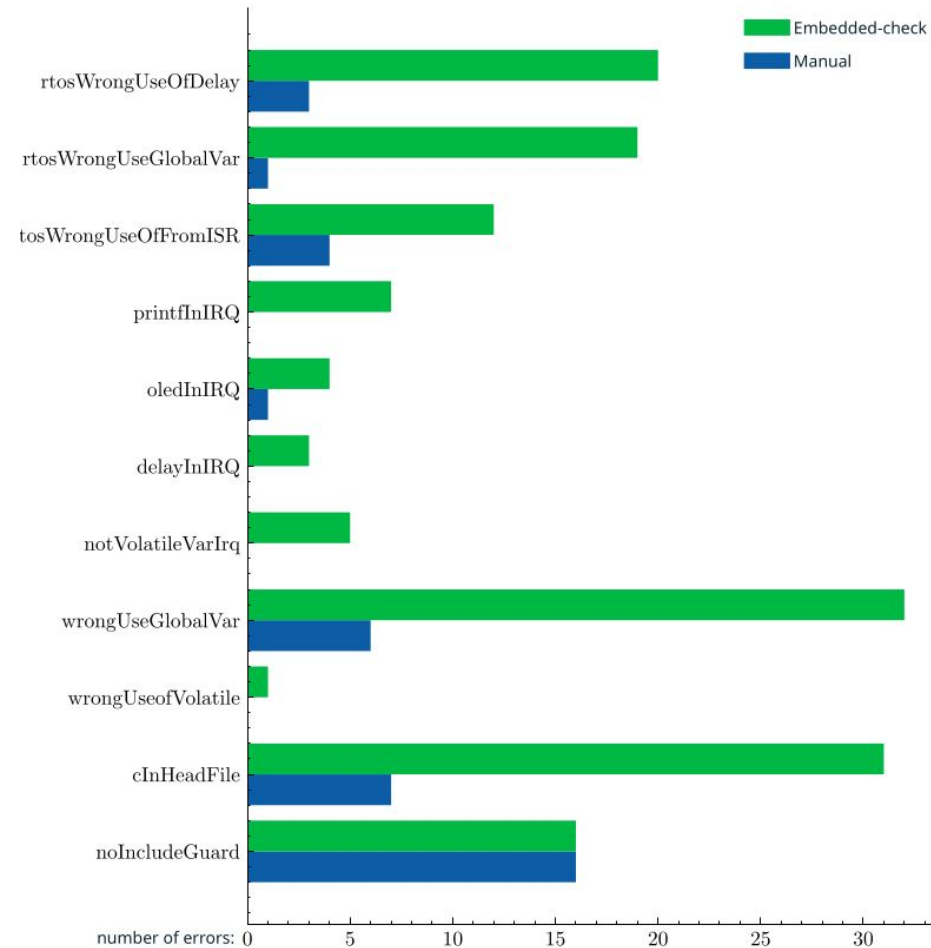
How about the labs?

Did not receive any
code feedback!

1250 detected errors

54.4%
Embedded
Systems

45.5%
RTOS

11.9
violations
per student
(std=0.7)

Common Errors

Rule	Projects (%)	Labs (%)
	N=150	N=1250
rtosWrongUseOfDelay	15.4	7.0
rtosWrongUseGlobalVar	13.1	55.4
rtosWrongUseOfFromISR	5.0	2.2
rtosMissingFromISR	0.0	0.2
printfInIRS	2.6	6.5
oledInIRS	2.6	1.4
delayInIRS	4.4	4.8
whileInIRS	0.0	0.2
notVolatileVarIRS	1.6	2.2
wrongUseGlobalVar	21.4	9.4
wrongUseVolatile	1.3	4.2
cInHeadFile	19.8	6.2
noIncludeGuard	12.8	0.4

Conclusions

The tool can replace manual feedback for checking the rules and it is easy to extend and can be use in different scenarios:

- **Post-Submission Analysis:** Identifies prevalent errors and assesses curriculum changes.
- **Manual Code Review Assistance:** Supports and improves efficiency in manual reviews.
- **Continuous Integration:** Ensures consistent code quality in CI systems for submissions.

github.com/rafaelcorsi

rafael.corsi@insper.edu.br

Rafael Corsi Ferrão (Insper), Igor dos Santos Montagner (Insper), Craig Zilles(UIUC), Mariana Silva (UIUC), Rodolfo Azevedo (Unicamp)