UnitJudge: a novel online automatic correction system for long programming practices by means of unit tests

Iván García-Magariño, Isabel Pita, Javier Arroyo, Marta López Fernández, Javier Bravo-Agapito, Clara Segura, Raquel Lacuesta Gilaberte

Complutense University of Madrid & University of Zaragoza

Jornadas de Innovación Docente 2023

Introduction

- Automatic Evaluation of Programming Practices
- Existing online judges rely in input and output streams
- Immediate Feedback
- Usually judges misses checking short pieces of code such as functions, methods or classes
- UnitJudge: novel online judge for long practices based on unit tests

Related Work

- (Cheang,2003) Online Judge for Data Structures and Algorithms, imported from competitive programming
- Jutge.org (Petit, 2018): data structures, AI, programmed in Haskell
- Acepta El Reto(Gomez-Martin 2017) 300+ exercises in Spanish, structured in categories
- DomJudge: used here in Faculty
- Missing: Testing separately different small pieces of code of a complex practice (normally based on input/output streams for small exercises)

UnitJudge

- Online Judge for evaluating long programming practices
- Evaluate small pieces of code (e.g. functions, methods, classes)
- Use Unit Test concepts
- Provides specific comment for each failure to guide students
- Supports two programming languages
  - C++ (own UnitJudge testing framework)
  - Java (uses JUnit with their annotations)
- Programmed using PHP, MySQL, Java and C++
- Online service with private access by professors and students

Main menu of UnitJudge in the professor role

Main Menu

Welcome to UnitJudge

Role: professor. Username: igarciam

Actions only for professors:

- UnitJudge Templates for Professors:
  - C++ Template for UnitJudge
  - Java Template for UnitJudge
- Create Practice
- Submissions:
  - Submission Results
  - Submission Files
- Handling Student Users:
  - Create Users for Students
  - Existing Students with Their Passwords

Student action:

- Submit solution

Main Menu

Creation of a practice

Upload only the **MainTests**(.cpp/Java) file.

Upload your file: Browse... No file selected.

Name of the Practice:

Language: C++

Upload

MainTests file of the C++ template of UnitJudge

```cpp
#include <iostream>
#include "UnitJudge.h"
// INCLUDES OF THE HEADER (.h) FILES OF THE PRACTICE
#include "HelloPractice.h"

using namespace std;


// INCLUDE HERE ALL THE TESTS THAT YOU NEED WITH THE ASSERT COMMAND.
// REMEMBER TO INCLUDE UnitJudge &uj AS REFERENCE PARAMETER  AND CALL
// uj.assert(condition, errorMsg) IN ALL NECESSARY COMPROBATIONS
void testGreeting(UnitJudge& uj) {
    string greeting = sayHello("Juan");
    uj.assert(greeting == "Hello, Juan", "sayHello should have returned 'Hello, Juan'");
}


int main()
{
    UnitJudge uj;
    uj.deactivateCout();
    // INCLUDE THE CALLS TO ALL THE TESTS IN THE MAIN
    testGreeting(uj);

    // It prints out the number of errors and the messages so the PHP UnitJudge
    // system can provide the output to students.
    uj.activateCout();
    cout << uj.getNumErrors()<<" "<<uj.getMessages();
    return 0;
}
```

MainTests file of the Java template of UnitJudge

```java
/* This class helps the teacher to prepare the tests for the UnitJudge
System. Do not change the name of the class.
Provide your students a template of the Practice (as HelloPractice)
in which the main aspects are included and MainTests with reduced set of tests.
Upload the MainTests to the UnitJudge, in which you can include more
tests to obtain more reliable results.
Upload the judge
 * @author Iván García-Magariño
*/
public class MainTests {
    // DO NOT CHANGE THIS MAIN METHOD
    // This method prints the number of failures and description of failures
    public static void main(String args[]){
        Result result = JUnitCore.runClasses(MainTests.class);
        String strForJudge ="";
        strForJudge += result.getFailureCount()+" "; // Number of results
        strForJudge += result.getFailures(); // Description of the failures
        System.out.println(strForJudge);
    }

    // INCLUDE HERE AS MANY TESTS AS NECESSARY WITH @Test BEFORE EACH METHOD.
    // SOME EXAMPLES ARE INCLUDED BELOW

    // Example of evaluation of practice. You can test implementation
    // right and wrong implementation of this practice. Be careful
    // with the names of packages, importing any package if necessary
    @Test
    public void testFromPractice(){
        HelloPractice hello= new HelloPractice();
        String name = "Juan";
        String greeting = hello.sayHello(name);
        assertEquals("Hello, Juan", greeting);
    }
```

Access of professors to submission results of students

Practice: 

Filter

Practice	Username	Result	Time	Messages
	igarciam	COMPILER ERRORS	2022-10-17 12:02:23	El sistema no puede encontrar el archivo especificado.
Hello	igarciam	CORRECT	2022-10-18 13:37:06	
Hello	professor	CORRECT	2022-10-07 15:52:24	
OcaV1	igarciam	CORRECT	2022-10-17 13:10:59	
OcaV1	mlopezfe	CORRECT	2022-10-11 16:04:42	
OcaV1S1	FPD15	FAILED	2022-10-18 23:08:52	esPuente falla en casilla 6 esPuente falla en casilla 12 esPuente falla en casilla 16 esPuente falla en casilla 22 esDados falla en casilla 26 esLaberinto falla en casilla 32 esDados falla en casilla 36 esLaberinto falla en casilla 42 esDados falla en casilla 43 esDados falla en casilla 53 siguientePuente falla en casilla 6 siguientePuente falla en casilla 12 siguienteDado falla en casilla 26 siguienteDado falla en casilla 53
OcaV1S1	igarciam	CORRECT	2022-10-18 14:04:07	
PruebaInicial	professor	COMPILER ERRORS	2022-10-10 13:22:20	MainTests.cpp:5:27: fatal error: HelloPractice.h: No such file or directory #include "HelloPractice.h" ^ compilation terminated.

Experimentation

- Subject “Fundamentals of Programming“
  - We gave them “.h” file
- Sample:
  - replies from 29 students
  - 20 male students (69%) and 9 female students (31%)
  - Mean age was 18.1 years old with a SD of 0.55
  - 93.1% of the students belonged to the Double Grade of Computer Science and Mathematics and 6.9% belonged to the Grade of Data Engineering and Artificial Intelligence

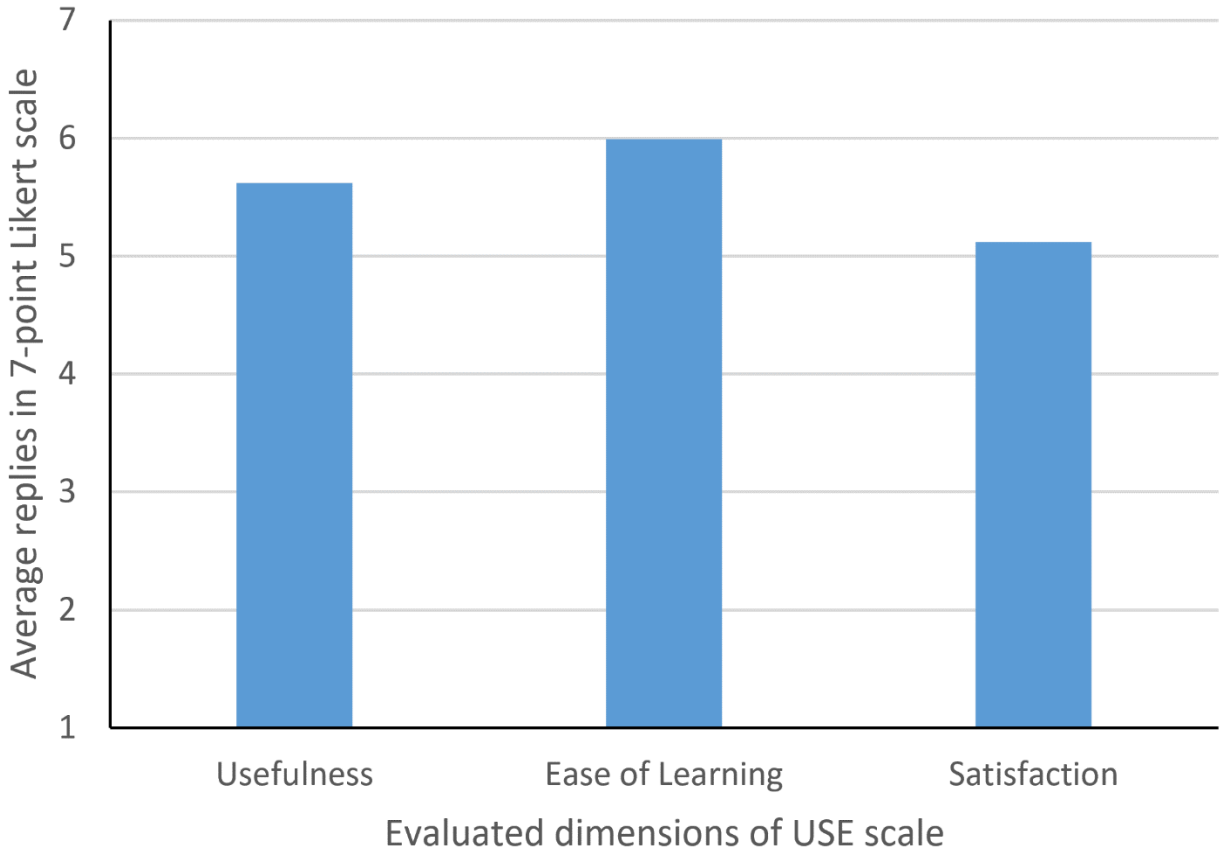
Exhaustive tests for the first week of the first version of the practice about the Game of the Goose

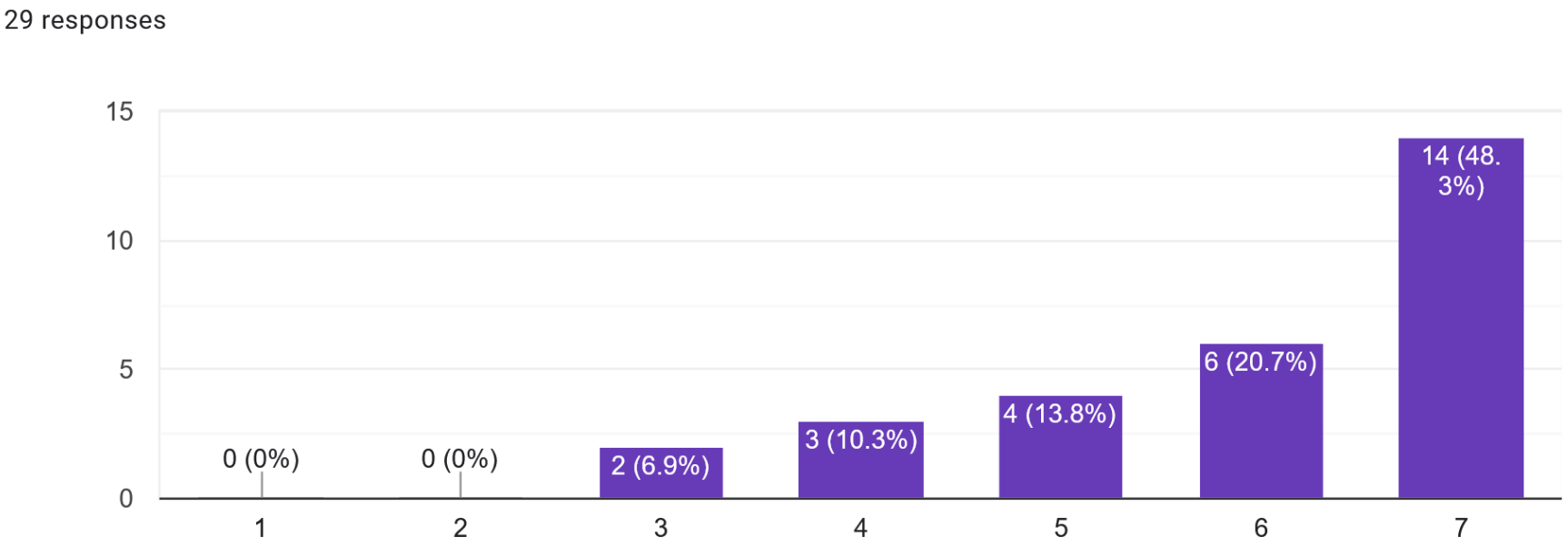
```
// Comprueba exhaustivamente todas las funciones con nombre esXXX
void testEsXXX(UnitJudge& uj) {
    for (int casilla = 2; casilla <= J_NUM_CASILLAS; casilla++) {
        uj.assert(esOca(casilla) == jEsOca(casilla), "esOca falla en casilla "+ to_string(casilla));
        uj.assert(esPuente(casilla) == jEsPuente(casilla), "esPuente falla en casilla " + to_string(casilla));
        uj.assert(esDados(casilla) == jEsDados(casilla), "esDados falla en casilla " + to_string(casilla));
        uj.assert(esLaberinto(casilla) == jEsLaberinto(casilla), "esLaberinto falla en casilla " + to_string(casilla));
        uj.assert(esMuerte(casilla) == jEsMuerte(casilla), "esMuerte falla en casilla " + to_string(casilla));
        uj.assert(esPosada(casilla) == jEsPosada(casilla), "esPosada falla en casilla " + to_string(casilla));
        uj.assert(esPrision(casilla) == jEsPrision(casilla), "esPrision falla en casilla " + to_string(casilla));
        uj.assert(esPozo(casilla) == jEsPozo(casilla), "esPozo falla en casilla " + to_string(casilla));
        uj.assert(esMeta(casilla) == jEsMeta(casilla), "esMeta falla en casilla " + to_string(casilla));
    }
}

// Comprueba exhaustivamente todas las funciones con nombre siguienteXXX
void testSiguienteXXX(UnitJudge& uj) {
    for (int casilla = 2; casilla <= J_NUM_CASILLAS; casilla++) {
        if (jEsOca(casilla))
            uj.assert(siguienteOca(casilla) == jSiguienteOca(casilla), "siguienteOca falla en casilla " + to_string(casilla));
        if (jEsPuente(casilla))
            uj.assert(siguientePuente(casilla) == jSiguientePuente(casilla), "siguientePuente falla en casilla " + to_string(casilla));
        if (jEsDados(casilla))
            uj.assert(siguienteDado(casilla) == jSiguienteDado(casilla), "siguienteDado falla en casilla " + to_string(casilla));
        if (jEsLaberinto(casilla))
            uj.assert(siguienteLaberinto() == jSiguienteLaberinto(), "siguienteLaberinto falla");
        if (jEsMuerte(casilla))
            uj.assert(siguienteMuerte() == jSiguienteMuerte(), "siguienteMuerte falla");
    }
}

int main()
{
    UnitJudge uj;
    uj.deactivateCout();
    // INCLUDE THE CALLS TO ALL THE TESTS IN THE MAIN
    testEsXXX(uj);
    testSiguienteXXX(uj);

    // It prints out the number of errors and the messages so the PHP UnitJudge
    // system can provide the output to students.
    uj.activateCout();
    cout << uj.getNumErrors()<<" "<<uj.getMessages();
    return 0;
}
```

Quantitative Results

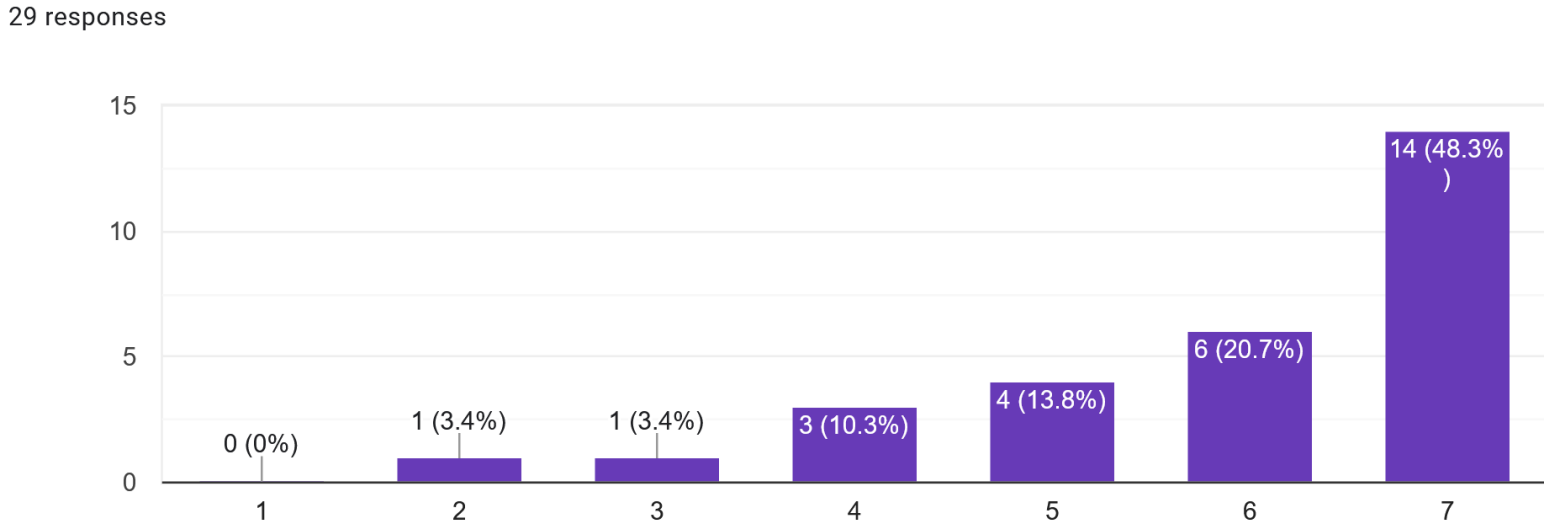
- USE scale, dimensions: (a) usefulness, (b) ease of learning and (c) satisfaction

Dimension	Average (7-point Likert)	SD
Usefulness	5.62	1.43
Ease of Use	5.99	1.26
Satisfaction	5.12	1.73



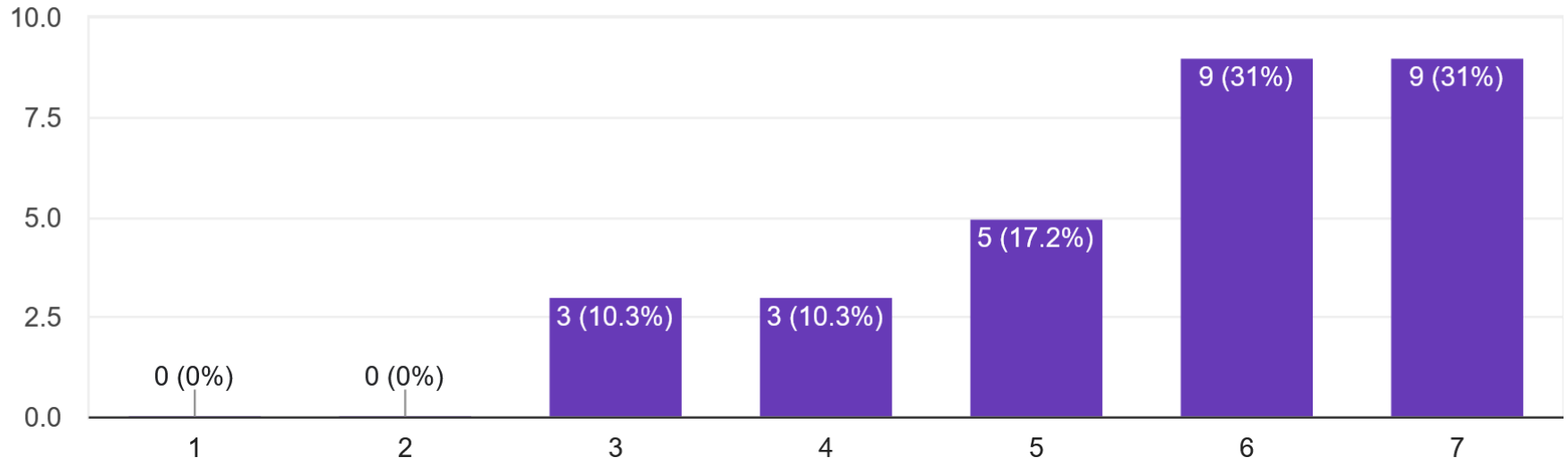


Distribution of responses in 7-point Likert scale to “It helps me to be more effective in developing a correct practice”



Distribution of responses in 7-point Likert scale to “I learned to use UnitJudge quickly”

29 responses



Distribution of responses in 7-point Likert scale to “I am satisfied with UnitJudge”

Qualitative Results

Comment	Frequency
Appreciated aspects	
It helps me to find all the errors easily fastening my debugging process	5
The comments provided by UnitJudge help me to find the reasons behind the errors in most cases	4
I like to have the immediate feedback from UnitJudge to know if I am programming the practice properly	3
Improvable aspects	
The "time out" problem does not provide feedback to help the reason and place of my error (e.g. place of infinite loop)	5
Interface could be improved confirming information (e.g. results of previous submissions or names of uploaded files)	2
It would be great if I only needed to upload one file instead of serveral ones, avoiding to upload the file provided by the professor	2

Most Frequent Comments of Participant Students

Conclusions

- UnitJudge: useful for long practices in fundamentals of programming (5.6 out of 7)
- Based on unit tests, for separately testing pieces
- Have guided students in learning process of programming achieving student's satisfaction (5.1 out of seven)
- UnitJudge is easy to use (6.0 out of 7)
- Facilitates debugging of practices (5 people mentioned it)
- Understand reasons behind errors (4 people mentioned (other judges do not provide reasons)
- Inmediate feedback is useful (3 people commented)

Publications

- García-Magariño, I., Pita, I., Arroyo, J., López Fernández, M, Bravo-Agapito, J., Segura, C., & Lacuesta Gilaberte, R. (2023) UnitJudge: a novel online automatic correction system for long programming practices by means of unit tests. In: The 10th International and the 16th National Conference on e-Learning and e-Teaching (ICELET 2023) . IEEE. https://doi.org/10.1109/ICeLeT58996.2023.10139865
- Alloza-García, S., García-Magariño, I., Bravo-Agapito, J., & Lacuesta Gilaberte, R. (2023) IDSOJ: An Intrusion Detection System in Online Judges for evaluating programming exercises. In: The 10th International and the 16th National Conference on e-Learning and e-Teaching (ICELET 2023). IEEE. https://doi.org/10.1109/ICeLeT58996.2023.10139878

Ongoing and Future Work

- Using it on Tecnnology of Programming II (2nd course subject focused on OOP)
- Students can now upload zip files so they can structure programs in different folders/packages
- Plan to provide explanation for time-out errors (5 students missed it)
  - E.g. providing output of their programs, so they can see if any infinite loop.