A Teaching Assistant for the C Language

Rui C. Mendes
Centro Algoritmi, Departamento de Informática, University of Minho, Braga, Portugal

José João Almeida
Centro Algoritmi, Departamento de Informática, University of Minho, Braga, Portugal

Abstract

We introduce a C tutor that can help instructors manage classes with many students learning to program in C. Nowadays, it is easy to evaluate code but it is hard to provide good feedback. We introduce a tool to help instructors provide students with feedback concerning their implementation and documentation for honing their programming skills. This tool is implemented in `Python` and is available at `https://github.com/rcm/C_teaching_assistant`.

2012 ACM Subject Classification Applied computing → Computer-assisted instruction; Software and its engineering → Empirical software validation

Keywords and phrases Software metrics, Documentation extractor, Domain specific language, Query language, Report generation

Digital Object Identifier 10.4230/OASIcs.ICPEC.2021.13

Category Short Paper

Supplementary Material *Software*: `https://github.com/rcm/C_teaching_assistant`

Funding This research has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

1 Introduction

Learning to program is a difficult task [8]. There are many concerns involved when teaching the first imperative programming language. Students are often fairly disorganized and have difficulty writing and understanding code. The common difficulty involved in learning to program is that students often write code tentatively by applying several patches until it finally is able to solve the given task. However, when involved in a project, this *ad hoc* methodology is quite detrimental. It is important to provide feedback concerning implementation. This means that students should realize which functions implemented are poorly written and should be improved.

When teaching classes with many students, it is hard to have enough teaching assistants for accompanying them and providing feedback on all fronts: writing code that implements the required specifications, is well documented and is easy to understand and maintain. In order to help instructors provide feedback to students, it makes sense to follow an automatic assessment strategy based on three pillars: functionality, readability and documentation.

The goal of the tool presented in this work is to offer a solution that will help instructors provide automatic assessment on two of these pillars: program readability/maintainability and documentation. With it, instructors will be able to produce reports that will provide feedback and help students fulfill these tasks. This system may also be used for summative assessment.

2 Software metrics

Using a tool capable of testing programs helps instructors understand whether the students were able to implement a solution capable of solving a given problem. This may be done by using many different online judge systems capable of automatically evaluating source

code [10]. However, it is possible to create a program that solves a task while writing code that is hard to read and maintain. In fact, this could be detrimental because it would give students the wrong feedback: as long as it solves the problem, the solution is accepted.

There is a known relationship between code that is difficult to understand and software metrics. This was also reported when creating questions for exams [5]. Some studies suggest that it is important to include software metrics in feedback given to students [2] and have reported including these metrics in Fuzzy rules used for giving advice to students [4]. Both of these studies have used these metrics and incorporated them when evaluating small programming assignments for providing feedback. Our aim is to extend these studies by providing a tool in order to automatically collect these statistics over all functions in a project.

Given all these clues from literature, it makes sense to incorporate information about software metrics when evaluating programming projects. Thus, the authors decided to include some measures like lines of code, cyclomatic complexity [7] or software maintainability index [3] in summative assessment. These were incorporated with other concepts like not defining code in include files, not having warnings and not having global variables. However, when presenting these measures to students, they reported difficulties in understanding how to improve their program in order to improve their evaluation. One of the difficulties reported was in identifying the functions that were responsible for their low results.

One way of helping both instructors and students would be to create a report that details these measures on a per function basis. If students are able to peruse a table that details this information for each function, they will have further clues as to which functions are more complex and should be rewritten. Thus, they can try to refactor the code to address these issues and better understand these concepts. Even thought there are tools readily available that can help gather these software metrics [11], they are not easy to use and only provide feedback per source code file. We aim to provide a tool that helps instructors evaluate students and/or create detailed reports using a large number of software metrics over all functions in a project.

3 Concerning documentation

The Bloom taxonomy is well known and can be applied when teaching computer science [9]. The cognitive categories involved in the taxonomy are to remember, understand, apply, analyse, evaluate and create. Some of these concepts can be applied when producing documentation. This task helps students understand and evaluate their code. While they write the documentation, they have to answer some key questions like how to explain what each function does, what it returns, what is the importance of each of the arguments it receives and what is the program flow necessary to implement the functionalities needed by their project. While producing code helps students to apply the knowledge they learned and use it to create a solution for a problem, documentation helps students analyse and evaluate their solution because it forces them to better understand how it works.

Evaluating documentation quality in a project is a cumbersome task because it involves checking that all the concepts in the program were explained: macros, types, functions and their arguments and return values. While a documentation coverage tool does not help instructors understand if the documentation that was produced is clear and well written, it can reduce the burden drastically. As far as the authors know, the automatic evaluation of documentation available when using `Doxygen` is `coverxygen` [6] that checks for documentation coverage. There are some common mistakes concerning student documentation. They usually:

- Forget to document some of the arguments;
- Copy information from other functions and thus document non-existing arguments;

- Forget to update the documentation after changing the function (e.g., adding or removing arguments);
- Document other entities as function arguments, like local variables or constants;
- State that a `void` function returns something.

Many of these mistakes can be checked automatically but there are no tools that help in this task. Coverxygen only gives information about documentation coverage, including datatypes, macros and function coverage but not argument coverage. Thus, while it is possible to use it to check whether all functions were documented, it does not help to identify the problems mentioned above.

4 The C Teaching Assistant

We introduce a tool, the C Teaching Assistant, that helps instructors produce documents that report what problems exist with the students' code or documentation. This tool analyses the users' submissions and gathers information about all the functions they contain. All the fields that can be used on the reports are presented in appendix A.

The tool is invoked by passing it several arguments that represent pathnames containing C code. Each of the arguments passed corresponds to a different project.

```
$ teaching_assistant.py PL[1-9]G[0-9]*
Enter query >
```

The tool recursively searches through all the existing C files (both header files and code files) and creates a table concerning all the information gathered. After consulting all the information, the user can use a domain specific language (DSL) to query the system. The DSL uses a syntax similar to SQL and is described in Listing 1. The queries can either be executed interactively or the resulting table may be converted into many formats including most markdowns, LaTeX or HTML.

Listing 1 The description of the query language. `expression` can contain arbitrary `Python` code and use field names.

```
SHOW <expressions  separated by spaces or semicolons>
[HEADER <fields separated by spaces>]
[COND <conditions using  fields and Python operands and functions>]
[SORT <fields separated by spaces>]
[COLOR <field> : <expression>[; <field> : <expression>]*]
[
GROUP_BY <fields separated by spaces>
[AGGREG <expressions separated by spaces or semicolons>]
]
```

5 Some examples of queries

Listing 2 shows an example that uses some of the measures collected by `Multimetric`, applies some functions to the values presented and uses a color code for signaling when these measures have non recommended values. When using something that returns a Boolean value, the system uses the green and red colors; when using a real number between zero and one, the system uses a palette. It is possible to use `Python` functions that return one color based on the value that was passed as well. The functions `scale_lower` and `scale_upper` will create functions that scale the values to the $[0, 1]$ interval using the lower and upper

bounds. The function `scale_upper` will create a scaler that returns 0 for values lower or equal than the lower bound, 1 for values greater or equal than the upper bound and uses interpolation between these bounds. The function `scale_lower` is reciprocal. This example also sorts the table results by decreasing values of `maintainability_index` and increasing values of `cyclomatic_complexity` and `name`. Figure 1 shows a snippet of what can be seen on the terminal. While it is possible to produce the output with less decimal places, it would involve a more complicated query.

Listing 2 Functions and some software engineering measures color coded by their severity. The query language allows the use of semicolons instead of spaces to be able to break lines in order for the query to be more readable.

```
SHOW name; cyclomatic_complexity; maintainability_index;
    scale_lower(1,10)(cyclomatic_complexity);
    scale_upper(60,100)(maintainability_index);
    0.5*scale_lower(1,10)(cyclomatic_complexity) +
    0.5*scale_upper(60,100)(maintainability_index)
HEADER name complexity maint_idx cmplxty_grd maint_grd assessment
SORT -maintainability_index cyclomatic_complexity name
COLOR
    complexity : scale_lower(1,10)(complexity);
    maint_idx : maint_idx > 80; cmplxty_grd : cmplxty_grd;
    maint_grd : maint_grd; assessment : assessment
```

Listing 3 shows two examples of queries containing aggregation. These can use any `Python` function over fields and, in the current implementation of this tool, uses some functions declared in the `statistics` module. The first example reports some statistics concerning the number of effective lines of code for each project and function return type. The second example presents the mean number of useful lines of code and counts the number of functions with less than 10 lines of code for all projects. This example is not very readable but is there to illustrate the reason why the `AGGREGATE` keyword can receive tokens separated by semicolons and that it can use arbitrary `Python` code.

Listing 3 Examples using aggregation.

```
SHOW name project return loc
GROUP_BY project return
AGGREG len(name) min(loc) mean(loc) max(loc)

SHOW name project return loc
GROUP_BY project
AGGREG mean(loc); (lambda L: len([x for x in L if x < 10]))(loc)
```

The system can also be used inside other programs. It is just a matter of first using the function `extract_all_functions` with a project and subsequently calling either the lower level function `function_query` that returns a list of lists containing the table or using the function `query` and pass it the structure returned from `extract_all_functions` and a string containing the query as illustrated in the examples given above (cf. Listing 4). Function `query` can also be passed an optional argument `fmt` that specifies which format (used by the module `tabulate` [1]) to use for the table.

Listing 4 Using inside other `Python` scripts.

```
from teaching_assistant import *
info = extract_all_functions("/home/rui/repos/PL2G01")
```

```
result = query(info, """SHOW name return args
SORT name""")
print(result)
result = query(info, ["SHOW name project","SORT name"], fmt = "html")
```

Listing 5 shows an example of generating a report. In order to use this functionality, the command is executed with the `-t` keyword.

```
teaching_assistant -t example_report PL[1-9]G[0-9]*
```

Listing 5 Example of generating a report.

````
# List of problematic functions
These functions are too complex and should be rewritten:

```
SHOW project name cyclomatic_complexity maintainability_index
COND cyclomatic_complexity > 10 or maintainability_index < 80
SORT project -maintainability_index cyclomatic_complexity name
COLOR
    cyclomatic_complexity : scale_lower(0,50)(cyclomatic_complexity);
    maintainability_index : scale_upper(0,100)(maintainability_index)
```

# Bad documentation
The following functions have no documentation:

```
SHOW name
COND not comment
```
These functions have a *@return* keyword for *void* functions:

```
SHOW name comment
COND comment and return == 'void' and '@returns' in comment
```

The following functions have problems with the definition of
their arguments. They either:
- Forget to document some of the function arguments, or
- Document things that are not function arguments

```python
def arg_doc_problems(args, comment):
    actual = set(args.keys())
    reported = set(re.findall(r"@param\s+(\S+)", comment, re.M))
    return actual != reported
def arg_names(args):
        return list(args.keys())
```

```
SHOW name filetype filename arg_names(args) documented(comment)
COND arg_doc_problems(args,comment)
```
````

name	complexity	maint_idx	cmplxty_grd	maint_grd	assessment
is_empty	0	100	1	1.0	1.0
penultimo	0	100	1	1.0	1.0
pop	0	100	1	1.0	1.0
top	0	100	1	1.0	1.0
has_type	1	100	1.0	1.0	1.0
main	1	100	1.0	1.0	1.0
duplica	2	100	0.8888888888888888	1.0	0.9444444444444444
enesimo	2	100	0.8888888888888888	1.0	0.9444444444444444
ler_linha	2	100	0.8888888888888888	1.0	0.9444444444444444
new_stack	2	100	0.8888888888888888	1.0	0.9444444444444444
popp	2	100	0.8888888888888888	1.0	0.9444444444444444
rodatres	2	100	0.8888888888888888	1.0	0.9444444444444444
trocadois	2	100	0.8888888888888888	1.0	0.9444444444444444
copian	3	100	0.7777777777777778	1.0	0.8888888888888888
parse2	3	100	0.7777777777777778	1.0	0.8888888888888888
verifica_carater	3	100	0.7777777777777778	1.0	0.8888888888888888
nott	5	100	0.5555555555555556	1.0	0.7777777777777778
push	3	99.92049769252367	0.7777777777777778	0.9980124423130917	0.8878951100454348
parse	3	97.99452174359152	0.7777777777777778	0.949863043589788	0.8638204106837829
converte_para_char	5	96.52391934110824	0.5555555555555556	0.913097983527706	0.7343267695416308
converte_para_double	6	96.1542362975608	0.4444444444444444	0.90385590743902	0.6741501759417322
converte_para_long	6	96.07022845886661	0.4444444444444444	0.9017557114716652	0.6731000779580548
decrementa	8	91.56801924883034	0.2222222222222222	0.7892004812207585	0.5057113517214904
incrementa	8	89.06398158095722	0.2222222222222222	0.7265995395239304	0.4744108808730763
print_stack	3	84.6730866972618	0.7777777777777778	0.6168271674315449	0.6973024726046613
xorr	10	81.5303145267608	0.0	0.5382578631690201	0.26912893158451007
e	10	80.66933794117628	0.0	0.5167334485294071	0.25836672426470353
ou	10	79.94165314864951	0.0	0.4985413287162377	0.24927066435811884
modulo	6	78.20989263681746	0.4444444444444444	0.45524731592043644	0.4498458801824404
soma	19	63.62646972307609	0	0.09066174307690229	0.045330871538451147
multiplica	20	62.87710754667034	0	0.07192768866675844	0.03596384433337922
expoente	20	62.54920051887836	0	0.06373001297195895	0.03186500648597947
subtrai	20	62.4927589726792 26	0	0.062318974316980656	0.031159487158490328
dividir	11	61.40891224648385	0	0.035222806162096276	0.017611403081048138
tokenizador	51	47.186473217293056	0	0	0.0

Figure 1 Result of running query on listing 2 on a given project.

The example is more or less self explanatory. It starts by creating a table of the functions that are too complex. Then it displays a table of the functions that don't have documentation. It shows the name of the functions and the condition uses a `Python` statement that checks if the string containing the comment is empty. Then it checks for functions that erroneously document the return value for `void` functions and those whose documentation either does not document all the arguments or has documentation for arguments that do not belong to the current function. This was achieved by creating two `Python` functions inside a Python environment called `arg_names` and `arg_doc_problems`. The first one just returns a list with the argument names from the dictionary than contains the argument names as keys and their type as value. The second one takes the dictionary of arguments and the comment with the documentation, extracts all the `@param` keywords therein and checks if both sets are equal.

The output can then be passed to a system that will take its output and generate another format. In this case, we used `pandoc` that can generate many different formats including HTML and LaTeX. The following statement illustrates how to generate a report in HTML.

```
teaching_assistant -t report.txt PL[1-9]G[0-9]* | pandoc -o report.html
```

6 Conclusions

This tool helps instructors create reports concerning students' code. Instructors can easily create reports about all projects with this tool since it enables a lot of sophistication. With a little thought, it is capable of both grading students or provide reports to help instructors understand the problems with the students' code and better help them address it. Since most of the keywords accept `Python` code, it is very easy to extend to one's needs. As the `tabulate` model can generate tables in many formats including most markdowns, HTML and LaTeX, the tables produced can be easily incorporated in many types of reports.

The main advantages of this tool is its simple syntax, it can use arbitrary `Python` code and automatically substitutes the values of fields, it can use all the statistics gathered by `Multimetric`. Tables may be generated with colors in a very straightforward fashion, they

can be generated in many formats and more advanced queries using aggregation can be used. The fact that it is possible to use markdown to create the report and have templating possibilities is also quite useful since it is possible to create small functions inside the text file for addressing a situation and embed tables inside the report.

Its main disadvantages is that is depends on other tools, like `Multimetric` and `Universal C tags`, it is currently not very graceful when encountering parsing errors and only works in Linux. It is also not capable of creating automatic advice. Future work will focus on some of the disadvantages mentioned above and to extend this tool to the `Python` language.

References

1 Sergey Astanin. Tabulate. `https://pypi.org/project/tabulate/`. Accessed: 2021-04-09.

2 Rachel Cardell-Oliver. How can software metrics help novice programmers? In *Proceedings of the Thirteenth Australasian Computing Education Conference-Volume 114*, pages 55–62, 2011.

3 Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 1994.

4 Francisco Jurado, Miguel A Redondo, and Manuel Ortega. Using fuzzy logic applied to software metrics and test cases to assess programming assignments and give advice. *Journal of Network and Computer Applications*, 35(2):695–712, 2012.

5 Nadia Kasto and Jacqueline Whalley. Measuring the difficulty of code comprehension tasks using software metrics. In *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136*, pages 59–65, 2013.

6 Xavier Marcelet. Coverxygen. `https://pypi.org/project/coverxygen/`. Accessed: 2021-04-09.

7 Thomas J McCabe. A complexity measure. *IEEE Transactions on software Engineering*, SE-2(4):308–320, 1976. `doi:10.1109/TSE.1976.233837`.

8 Simon, Andrew Luxton-Reilly, Vangel V. Ajanovski, Eric Fouh, Christabel Gonsalvez, Juho Leinonen, Jack Parkinson, Matthew Poole, and Neena Thota. Pass rates in introductory programming and in other stem disciplines. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '19, page 53–71, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3344429.3372502`.

9 Errol Thompson, Andrew Luxton-Reilly, Jacqueline L Whalley, Minjie Hu, and Phil Robbins. Bloom's taxonomy for cs assessment. In *Proceedings of the tenth conference on Australasian computing education-Volume 78*, pages 155–161, 2008.

10 Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. A survey on online judge systems and their applications. *ACM Computing Surveys (CSUR)*, 51(1):1–34, 2018.

11 Konrad Weihmann. Multimetric. `https://pypi.org/project/multimetric/`. Accessed: 2021-04-09.

A List of keywords

This appendix shows a list of the keywords that can be used by the tool. The software engineering statistics that are provided by `Multimetric` include measures that are only available for other languages and thus are omitted from this list. The keywords are the following:

name Function name

folder The path of the project containing the file where the function was defined

project The basename of `folder`

filename The path of the file where the function was defined

filetype Whether the function was defined in a C file or include file

return Function return type
args Dictionary of function arguments and their types
comment Comment before the function that may be the Doxygen documentation
vars Dictionary of local variables and their types
stat Software engineering statistics provided by `Multimetric` for each function
- **comment_ratio** Percentage of commments
- **cyclomatic_complexity** Cyclomatic complexity according to McCabe
- **halstead_bugprop** Number of delivered bugs according to Halstead
- **halstead_difficulty** Difficulty according to Halstead
- **halstead_effort** Effort according to Halstead
- **halstead_timerequired** Time required to program according to Halstead
- **halstead_volume** Volume according to Halstead
- **lang** Programming language
- **loc** Lines of code
- **maintainability_index** Maintainability index
- **operands_sum** Number of used operands
- **operands_uniq** Number of unique used operands
- **operators_sum** Number of used operators
- **operators_uniq** Number of unique used operators