Towards a Generic Model for Classifying Software into Correctness Levels and its Application to SQL

1st Benard Wanjiru
Software Science
Radboud University
Nijmegen, The Netherlands
benard.wanjiru@ru.nl

2nd Patrick van Bommel
Software Science
Radboud University
Nijmegen, The Netherlands
p.vanbommel@cs.ru.nl

3rd Djoerd Hiemstra
Data Science
Radboud University
Nijmegen, The Netherlands
hiemstra@cs.ru.nl

***Abstract*—Automated grading systems can save a lot of time when carrying out grading of software exercises. In this paper we present our ongoing work on a generic model for generating software correctness levels. These correctness levels enable partial grading of students' software exercises. The generic model can be used as a foundation for building SQL grading systems that check for correctness of SQL queries and can be generalized to different programming languages.**

***Index Terms*—correctness levels, software correctness, automated grading, assessment, partial marks, SQL query grading**

I. Introduction

Software correctness levels are discrete levels that show the level of adherence of software to the given specifications and requirements. The highest level shows that the software exactly matches those specifications and the lowest level shows that the software does not match the specifications at all. Software correctness is an important aspect of software science and in some cases critical [1]. Whenever we build software, we accompany it with a list of requirements that we use to judge the quality of the software. We check whether it conforms to the specifications and requirements. This also directly translates to the code that was used to build the software. As we strive to make sure that the code follows the programming language requirements like the syntax used, we also make sure it executes as required and it carries out the required task.

In a production environment, grading of software can be very strict [1]. A program that fails its intended purpose is unusable, but in educational settings, the rules are more relaxed. Education instructors are mandated to pay attention to the different levels of understanding the students have on a topic [2], [3]. As the instructors spend the time increasing these levels, they must make sure the student can see themselves improving. This instils and keeps the motivation of acquiring software engineering skills of making a correct software product. To accomplish this, the instructor needs to assess the software written by the student, identify the parts the student has done well and those that are lacking, generate constructive feedback that the student will use to improve, and finally award a grade that the student will be satisfied with. If the instructor does not pay attention to the level of understanding of the student shown on the software thereby awarding a very strict grade that does not factor in the student's efforts, it may result in damaging the motivation of the student to keep learning the topic.

Grading of students software exercises in an education setting has mainly been characterized by the instructor going through hundreds of submissions and awarding a grade as they see fit [4]. By the time they are coming to the last submissions, the level of concentration has waned resulting in awarding grades that bring students to their door with a list of complaints. Because of these challenges and more [5], the motivation for building automated grading tools has increased over the years as the number of students that attend the classes keeps on growing [6]–[11]. These tools have been envisioned to have the grading capabilities of an instructor at their best and use the same level of grading for all the student submissions to be graded such that, at the end, both the instructor and the student are satisfied.

Our motivation for this paper is to present a generic model that we have started working on for building automated grading tools. Tools that identify the different students' levels of understanding about a given topic and award a grade based on this level. Tools than can also award partial grades and use the missing points as automated feedback to teach students where they need to improve. We are currently experimenting with the model discussed for an automatic grading system of SQL exercises in the first year's BSc course Information Modelling and Databases at the Radboud University.

II. Research goals

This paper gives a glimpse of our ongoing work at Radboud University focusing on automated grading of students' software exercises. We have started our work with SQL as the main programming language and later generalize to other languages. Our research goals are:

1) Assigning a piece of software a meaningful discrete level or measure that shows how correct it is.
2) Normalizing this measure to have a common reference frame for different programming languages with varying characteristics.
3) Using this measure in automated grading of SQL exercises.

III. The generic model

Automated grading tools having a minimum of two correctness levels, *correct* and *incorrect* have already been proposed [7], [12]. Other tools capable of using more than two correctness levels are also available [8]–[11].The greater the number of correctness levels a tool possesses, the more profound its understanding. As an illustration, [6] uses 8 correctness levels to distinguish among syntax, schema, and semantics errors. The authors employ distinct properties to evaluate correctness for each of these systems, tailored to their particular objectives.

A generic model serves the purpose of aiding system developers determine the number of correctness levels needed for their system to be able to effectively evaluate a given piece of software according to their needs. It is based on the following:

1) **Properties** These are the characteristics of the software like syntax and semantics that can be used to evaluate the quality of software.
2) **Outcomes** These are the possible outcomes that a property can take like correct and incorrect.

These are discussed further below.

A. Properties

There are many characteristics of software code that can be used to evaluate its quality like syntax. Style++ [13] checks the *style* of the C++ code written by the students. The concept of correctness can be extended to include *performance* [14] and *complexity* [15] as fundamental components of overall software quality. The properties used when implementing a grading system depends on the goals of the system and the limitations of the development language. What makes sense for the language to check or not to check. Some of these properties are shown in table I. This is not an exhaustive list, but the number of properties can be added or removed at will. The syntax is the most fundamental property to check as this is the starting point to create a piece of software that can run. Semantics property is used to check whether the code written can accomplish the given task. The result is only meaningful for those tasks that have an output. Some tasks can be accomplished by writing the code in different ways. Some ways may contain some aspects or elements that are deemed unnecessary for there may be a simpler way of doing it. In this situation, checking for complexity may be necessary. Computing resources are finite and as we develop software, we need to track how much resources a piece of software consumes. some of these resources are the time it takes to execute, how much memory is consumed and network bandwidth. For those platforms that are resource sensitive checking for efficiency becomes important.

TABLE I
SOME PROPERTIES THAT CAN BE USED TO EVALUATE THE QUALITY OF SOFTWARE.

Property	Description
Syntax	Rules and grammar defined by the language.
Semantics	What the software does.
Result	The output from running the software.
Complexity	Code quality for maintenance purposes.
Efficiency	How much resources the software consumes.

B. Outcomes

When we check for the properties discussed above, we need to give them a measure of quality from a list of categories. Researchers such as [6] use two categories, *correct* and *incorrect*. The number of categories to use depends on how sensitive the system developers want the grading system to be. The more sensitive the system the more categories needed. Some of these categories are shown in table II. The highest category i.e., outcome means that the property meets all the specifications or requirements needed and vice versa. A grading system with only two outcomes is a binary system translating to pass or fail. As with the properties, the list in table II is not an exhaustive list but more outcomes can be added or removed as needed.

C. Setting the context for the generic model

In this section we shall explain our thoughts about the generic model. In section III-A we described properties of software. In the generic model, we will use an underlying set P to contain the relevant properties. In section III-B we described outcomes for properties. In the generic model, we will use an underlying set T to contain relevant outcomes.

The possible outcomes for a specific property is now given by the following function:

$$\mathit{Outcomes} : P \to \mathbb{P}(T) \tag{1}$$

where $\mathbb{P}(T)$ is the powerset of T. For $x \in P$, we then have that $\mathit{Outcomes}(x)$ gives the possible outcomes of property x.

When specific properties with their outcomes have been chosen, the correctness levels can be generated. The number of correctness levels ℓ is based on the Cartesian product as follows:

$$\ell = \prod_{x \in P} |\mathit{Outcomes}(x)| \tag{2}$$

Some of these levels will be unusable as we shall see.

TABLE II
POSSIBLE OUTCOMES FOR THE PROPERTIES SHOWN IN TABLE I. IN THIS CASE THE HIGHEST OUTCOME IS CORRECT OR HIGH AND THE LOWEST OUTCOME ABSOLUTELY INCORRECT OR LOW DEPENDING ON THE PROPERTY. A PROPERTY LIKE COMPLEXITY OR EFFICIENCY WOULD USE THE OUTCOME *high* INSTEAD OF CORRECT AND SO ON.

Outcome	Description
Correct	Property meets full specifications.
High	Property meets full specifications.
Minor Incorrect	Property has minor errors that can be discarded.
Major Incorrect	Property has errors that threaten the quality.
Absolutely Incorrect	Property has fatal errors.
Low	Property has fatal errors.

IV. The choice for specific properties

Platforms in which a piece of software is written vary with differing characteristics. This means, the number of properties will correspond to the platform and how thorough the developers of the systems want it to be. For example, a grading system for evaluating a piece of code written in SQL might not use the efficiency property. Database management systems are designed to optimize query execution without the user having to worry about efficiency [16]. For any platform, the following procedure shows how to instantiate the generic model:

1) Choose the properties P that you want applied in the grading system and the possible outcomes T.
2) For each $x \in P$, choose *Outcomes*(x).
3) Determine inappropriate levels and remove them.

We now give a specific instantiation of the model for an SQL platform. At Radboud university, we are experimenting with an automated grading system that can carry out partial grading of students' exercises.

1) Properties: The properties that we have found to be useful are $P = \{$*Syntax, Semantics, Result*$\}$,

Syntax - Syntactic errors are mainly due to typing errors as a result of lack of practice or carelessness [17]. A statement with a misspell of a single character should be differentiated from a statement with wrong syntax in several clauses when awarding grades. This is possible by carrying out syntax analysis. The errors found can be used to generate constructive feedback for the student. It has been shown that students improve on their answers when they are given such feedback [18].

Semantics - Semantic analysis mainly involves testing equivalence of SQL queries [19]. It involves checking whether a student's query has the same meaning as the instructor's reference query. With semantic analysis we can also identify different answering patterns and approaches [20]. This can help to identify difficult SQL areas so that we can improve the quality of teaching to help the students learn better.

For an incorrect answer, with semantic analysis we can identify how far off the student is from what was asked. Whether they completely have no idea or had the idea well formulated in their mind but could not fully translate it to the code.

Result - Analyzing the result is another way of testing the equivalence of SQL queries [8]. SQL is a language that allows statements with different query formulations, but which accomplish the same thing. For those tasks that return a result, analyzing it can help to narrow down on the grading of how correct the code was. If the student's answer has different syntax from the one of the instructor, checking the result against multiple databases can aid in identifying if they have the same meaning. We can analyze incorrect results to get a glimpse of the student's level of understanding about the concept in question.

We do not consider complexity and efficiency as database management systems feature query optimizations [16].

2) Outcomes: One example of achieving the different levels of correctness is to check how many edits it would take an incorrect query to match the correct one. Another example could be to compare the results of the student's query with the results of various other wrong queries. The possible outcomes we have found for our SQL platform are:
$T = \{$*Correct, Minor Incorrect, Major Incorrect*$\}$,

- **Correct** - This is the highest level of an outcome in any scenario. In our case it means the student has fully grasped the concepts of the properties discussed above. For example, they have a very good understanding of SQL syntax.
- **Minor incorrect** - A student might make simple misspell errors even though they have complete understanding of what is being asked. With this outcome, we can appreciate a student's effort and knowledge.
- **Major incorrect** - This is the lowest outcome in our setting. In this case the student has no understanding of the concepts of the properties checked.

To illustrate how this example could work, we could use all the three properties P. And for each property we could use all the three possible outcomes T.

3) Correctness levels: The list of correctness levels for this case is as shown in table III. $L10$ and $L19$ are unusable. $L8$ can be avoided by checking the result against varying database data. Table IV shows an example of how correctness levels can be assigned to queries. The queries are sourced from some answers given by students on a quiz at Radboud university in the class information modelling and databases. *Q1* and *Q2* are fully correct. *Q3* has an extra invalid character , *D. Q3* is as a result of students copying DuckDB prompt character , *D. Q4* searches for a string ending with *Nijmegen* instead of containing the word.

V. Discussion

Software correctness levels are important for an effective grading system. Software grading is done in many areas, and it begins at the educational level when we are training students and equipping them with skills to build correct systems.

Manual grading involves applying a mental chart of levels of software correctness awarding a grade appropriately to each submission. A correct submission is awarded a full grade. A submission that misses on all the concepts is be awarded zero or close to zero points. Lastly, points are distributed to other works depending on how close to correct the work is. The challenges of this grading includes inconsistency as the instructor's concentration wanes, taking too much time to go through all the submissions, late feedback to the students and many more. Due to these challenges, the benefits of an automated grading system become apparent. The major requirement of this system is that it should perform grading as the instructor at their best mental focus and consistently apply it to all the work in a matter of minutes.

The generic model presented generates discrete levels of software correctness, enabling an effective partial grading system. Users must define the properties to be checked and

TABLE III
CORRECTNESS LEVELS OF VARYING SYNTAX, SEMANTICS AND RESULT. LEVELS WITH * ARE NOT POSSIBLE.

Level	Syntax	Semantics	Result
L1	Correct	Correct	Correct
L2	Minor Incorrect	Correct	Correct
L3	Major Incorrect	Correct	Correct
L4	Correct	Minor Incorrect	Correct
L5	Minor Incorrect	Minor Incorrect	Correct
L6	Major Incorrect	Minor Incorrect	Correct
L7	Correct	Major Incorrect	Correct
L8	Minor Incorrect	Major Incorrect	Correct
L9	Major Incorrect	Major Incorrect	Correct
L10*	Correct	Correct	Minor Incorrect
L11	Minor Incorrect	Correct	Minor Incorrect
L12	Major Incorrect	Correct	Minor Incorrect
L13	Correct	Minor Incorrect	Minor Incorrect
L14	Minor Incorrect	Minor Incorrect	Minor Incorrect
L15	Major Incorrect	Minor Incorrect	Minor Incorrect
L16	Correct	Major Incorrect	Minor Incorrect
L17	Minor Incorrect	Major Incorrect	Minor Incorrect
L18	Major Incorrect	Major Incorrect	Minor Incorrect
L19*	Correct	Correct	Major Incorrect
L20	Minor Incorrect	Correct	Major Incorrect
L21	Major Incorrect	Correct	Major Incorrect
L22	Correct	Minor Incorrect	Major Incorrect
L23	Minor Incorrect	Minor Incorrect	Major Incorrect
L24	Major Incorrect	Minor Incorrect	Major Incorrect
L25	Correct	Major Incorrect	Major Incorrect
L26	Minor Incorrect	Major Incorrect	Major Incorrect
L27	Major Incorrect	Major Incorrect	Major Incorrect

TABLE IV
CORRECTNESS LEVELS APPLIED TO A FEW QUERIES. THE CORRECT QUERIES ARE ASSIGNED LEVEL L1.

Q	Query	Level
1	SELECT * FROM Scientist WHERE university LIKE '%Nijmegen%';	L1
2	SELECT id, name, university, year FROM Scientist WHERE university LIKE '%Nijmegen%';	L1
3	D SELECT * FROM Scientist WHERE university LIKE '%Nijmegen%';	L2
4	SELECT * FROM Scientist WHERE university LIKE '%Nijmegen';	L4

their corresponding outcomes, which should align with the characteristics of the programming language in use. For example, SQL code execution time is irrelevant as database management systems feature query optimizations, and thus would not be used to grade SQL exercises. The number of outcomes chosen affects the sensitivity of the grading system, with a larger number allowing for more differentiation of correctness levels. A binary grading system with two outcomes per property would only allow for full marks or none, while a system with more outcomes provides more granularity in grading.

Automated grading systems can have positive and negative impacts on students, but our model addresses concerns around fairness and accuracy by providing greater granularity in outcomes and properties.

REFERENCES

[1] N. G. Leveson and K. A. Weiss, "Chapter 15 - software system safety," in *Safety Design for Space Systems*, G. E. Musgrave, A. S. M. Larsen, and T. Sgobba, Eds. Burlington: Butterworth-Heinemann, 2009, pp. 475–505.

[2] A. Saglam-Arslan and Y. Devecioglu, "Student teachers' levels of understanding and model of understanding about newton's laws of motion." in *Asia-pacific Forum on science learning & Teaching*, vol. 11, no. 1, 2010.

[3] M. Krell, A. Upmeier zu Belzen, and D. Krüger, "Students' levels of understanding models and modelling in biology: Global or aspect-dependent?" *Research in Science Education*, vol. 44, pp. 109–132, 08 2013.

[4] D. R. Sadler, "Indeterminacy in the use of preset criteria for assessment and grading," *Assessment & Evaluation in Higher Education*, vol. 34, no. 2, pp. 159–179, 2009.

[5] B. Cheang, A. Kurnia, A. Lim, and W.-C. Oon, "On automated grading of programming assignments in an academic institution," *Computers & Education*, vol. 41, no. 2, pp. 121–131, 2003.

[6] S. Dekeyser, M. de Raadt, and T. Y. Lee, "Computer assisted assessment of sql query skills," in *Proceedings of the Eighteenth Conference on Australasian Database - Volume 63*, ser. ADC '07. AUS: Australian Computer Society, Inc., 2007, p. 53–62.

[7] A. Bhangdiya, B. Chandra, B. Kar, B. Radhakrishnan, K. V. M. Reddy, S. Shah, and S. Sudarshan, "The XDa-TA system for automated grading of sql query assignments," *2015 IEEE 31st International Conference on Data Engineering*, pp. 1468–1471, 2015.

[8] B. Chandra, A. Banerjee, U. Hazra, M. Joseph, and S. Sudarshan, "Automated grading of sql queries," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019, pp. 1630–1633.

[9] G. Dambić, M. Fabijanić, and A. L. Ćošković, "Automatic, configurable and partial assessment of student sql queries with joins and groupings," in *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, 2021, pp. 837–842.

[10] M. Fabijanić, G. Dambić, and J. Sasunić, "Automatic, configurable, and partial assessment of student sql queries with subqueries," in *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, 2022, pp. 542–547.

[11] J. Kjerstad, "Automatic evaluation and grading of sql queries using relational algebra trees," Master's thesis, Norwegian University of Science and Technology, 2020.

[12] S. Nalintippayawong, K. Atchariyachanvanich, and T. Julavanich, "Dblearn: Adaptive e-learning for practical database course — an integrated architecture approach," in *2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2017, pp. 109–114.

[13] K. Ala-Mutka, T. Uimonen, and H.-M. Järvinen, "Supporting students in c++ programming courses with automatic program style assessment," *JITE*, vol. 3, pp. 245–262, 01 2004.

[14] N. R. Tallent and J. M. Mellor-Crummey, "Effective performance measurement and analysis of multithreaded applications," ser. PPoPP '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 229–240.

[15] F. G. Wilkie and B. Hylands, "Measuring complexity in c++ application software," *Software: Practice and Experience*, vol. 28, 1998.

[16] S. Chaudhuri, "An overview of query optimization in relational systems," in *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, ser. PODS '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 34–43.

[17] A. Ahadi, J. Prior, V. Behbood, and R. Lister, "Students' semantic mistakes in writing seven different types of sql queries," ser. ITiCSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 272–277.

[18] C. Kleiner, C. Tebbe, and F. Heine, "Automated grading and tutoring of sql statements to improve student learning," in *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 161–168.

[19] S. Chu, C. Wang, K. Weitz, and A. Cheung, "Cosette: An automated prover for sql," in *CIDR*, 2017.

[20] S. Yang, G. L. Herman, and A. Alawini, "Analyzing student sql solutions via hierarchical clustering and sequence alignment scores," in *1st International Workshop on Data Systems Education*, ser. DataEd '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 10–15.