Qualitative Content Analysis of Programming Errors

Philipp Shah
Technical University of Munich
Arcisstr. 21
80333 Munich, Germany
philipp.shah@tum.de

Marc Berges
Technical University of Munich
Arcisstr. 21
80333 Munich, Germany
berges@tum.de

Peter Hubwieser
Technical University of Munich
Arcisstr. 21
80333 Munich, Germany
peter.hubwieser@tum.de

ABSTRACT
In computer science education, the analysis of source code with errors is of interest as programming errors may give a hint to misconceptions. The analysis of misconceptions can help teachers to improve their exercises and lessons.

The semantic analysis of texts or video sequences could lead to different, subjective interpretation. This problem also affects source code errors, which could contain semantic errors.

In different projects, we were confronted with a lot of incorrect source codes, which were written by students of universities and secondary schools. A first analysis of these errors led to a categorization of errors regarding missing competencies.

To avoid mainly subjective interpretation of source code errors a standardized method for categorizing errors, which could also be applied by a practitioner, has to be developed and justified. Categorizing texts or source code errors is a matter of semantics, because text or code elements have to be interpreted. Thus, a qualitative content analysis is most suitable. In this paper we explain the difference between errors and misconceptions and present our adaption of the qualitative content analysis of Mayring to source code errors.

CCS Concepts
- Information systems → Content analysis and feature selection; Social and professional topics → Computing education; Theory of computation → Categorical semantics;

Keywords
Misconceptions; Programming errors; source code errors, content analysis; qualitative research method; error categorization.

1. INTRODUCTION
Students of computer science sometimes make errors or ‘mistakes’. Some of these errors are just problems with attention or syntax, like a missing semicolon at the end of a Java statement. But, some of them have their origin in misconceptions. For instance, some students state that the manipulation of a counter variable of a loop after the loop is accepted scientific notions.

A misconception is a conception which is not in line with accepted scientific notions [3], e.g. some students are stating that the manipulation of a counter variable of a loop after the loop is done still affects the loop [1]. Or a student tries to call a method from an object by using the class name instead of the object identifier. Holland, Griffiths and Woodman called this category “Object/class conflation” [4].

It is important to mention that not every error or mistake is a misconception. If a student just forgets a bracket at the condition of a while statement in Java he/she might not have a wrong conception of while statements at all.

A compiler or an automatic assessment system is able to find syntactical errors, but will never find a misconception, if, by chance, the misconception is not causing a syntactical error.

2. THEORY
Before introducing the adaption of the methodology we provide a brief overview of the used theories. Following the twofold design of the paper, we first present the difference between misconceptions and errors and afterwards the theory of qualitative text analysis. While the latter one underpins our theoretical approach, the first theory clarifies the practical application of it.

2.1 Misconceptions
A misconception is a conception which is not in line with accepted scientific notions [3], e.g. some students are stating that the manipulation of a counter variable of a loop after the loop is accepted scientific notions.

Holland, Griffiths and Woodman called this category “Object/class conflation” [4].

It is important to mention that not every error or mistake is a misconception. If a student just forgets a bracket at the condition of a while statement in Java he/she might not have a wrong conception of while statements at all.

A compiler or an automatic assessment system is able to find syntactical errors, but will never find a misconception, if, by chance, the misconception is not causing a syntactical error.

2.2 Intersubjective Analysis
Social science provides three different basic approaches to analyze texts and speeches [2]:

(1) Discourse Analytic Approaches, like Conversation Analysis [5], Metaphor Analysis [6], or Text Mining [7] utilize hermeneutic approaches. Discourse analytic approaches start with a linguistic criterion [2], like reconstructing the interaction process of an interview or searching the text for metaphors. After determining the linguistic criterion, the results are interpreted in a
hermeneutical way, with all advantages and disadvantage of that approaches. Text mining is one approach [7], which is an exception, it overcomes the disadvantage of uncoupling the results from a precise research question by integrating elements of modern content analysis.

(2) Modern Hermeneutic Approaches, like Grounded Theory Coding [8], Biography Analysis [9], or Psychoanalytical Text Interpretation [10] are open to every kind of content that a text or a speech contains. These methods could easily uncouple the interpreted text content from a precise research question, like categorizing source code errors. If a researcher or teacher is open for every hint about cs education, these approaches could be applied, otherwise these methods are not precise enough.

(3) Content Analysis has its origin in an analysis of a newspaper in 1893 by John G. Speed [11]. Articles of a newspaper were categorized by thematic categories. The frequency of occurrence of every thematic category were determined and then compared to the frequencies of other newspapers. Modern Content Analysis, like Qualitative Content Analysis [2] or Complex Quantitative Approaches [12] are more than just determining the frequency of categories. The qualitative content analysis is a mixed-methods approach with a strict rule-based procedure and produces intersubjective and reliable [2] results.

3. Related Work
As our research has two parts we present related work on misconceptions and content analysis.

3.1 Misconceptions
In computer science education there are a lot of publications about misconceptions, but they are often a collections of misconceptions without a standardized way of gathering them; e.g. in 2005 Ragonis and Ben-Ari [13] stated about 58 misconceptions and difficulties.

Holland et al. [4] were some of the first authors who collected misconceptions using an object oriented programming language (Smalltalk). Fleury [14] searched for misconceptions of students using Java as a programming language and referred to [4]. Ragonis and Ben-Ari's [13] study about misconceptions was based among others on Holland et al. [4] and Fleury [14].

Danielsiek, Paul and Vahrenhold [15], who referenced to Holland et al. [4], and Ragonis and Ben-Ari [13], investigated misconceptions on complex data structures like binary search trees and heaps.

Gal-Ezer and Zur [16] and later Shah et al. [1] succeeded in connecting intuitive rules, which had negative effect in science and mathematics [3], to misconceptions in computer science. They discovered [16] and verified [1] among others the misconceptions “two programs containing the same statements (even if in different order) are equally efficient” following the intuitive rule “same A, same B” [3], which was discovered by Stavy and Tirosh.

Xinogalos [17] provides a broad overview on the current state of the investigation of misconceptions related to objects and classes.

Berges examined novice programmers using concept maps to gain misconceptions. [18].

3.2 Text Analysis in Computer Science Education
In computer science education different kinds of text analysis are applied. Mostly, to analyze the content of interviews of students or experts.

Different authors, like Fincher, Havenga, and Hanks, used grounded theory, an approach of modern hermeneutic. In 2007, Fincher and Tenenberg [19] analyzed answers about “how teaching practices transfer” from teachers sent by e-mails. Their first question to an e-mail list was ignored. After that, they changed their way of asking and analyzed all received answers in a hermeneutic way. The analysis contained discourse analytic elements without stating them explicitly.

In 2008 Havenga, Mentz and Villiers investigated in 2008 the research question “What are the differences between the ways that successful and unsuccessful programmers apply their knowledge, skills and strategies in an object-oriented programming task?” [20]. For this purpose, they collected computer programs together with written thinking processes and analyzed it with qualitative and quantitative methods. The written thinking processes were analyzed using grounded theory. Unfortunately, there was no standardized and systematical analysis of the program codes and its semantics. The program codes were used to check the interpretation of the thinking processes, rather than being a further source of interpretation.

Later, in 2011, Havenga, Villiers and Mentz [21] investigated “how the thinking processes and strategies used by high-performing student programmers foster the development of correct computer programs”. In their research, they also used a mixed-methods approach including grounded theory, which was similar to the research design before in [20]. They assigned scores to the program codes “in the same way [they] would mark (score) it for a semester mark” [21] and treated the scores as quantitative data. Only the content of the recorded thinking processes was analyzed with grounded theory in a qualitative way.

In 2009 Hanks, Murphy, Simon, McCauley and Zander analyzed what kind of advice CS students at the end of a course would offer to other students who might take the course in the future [22]. The written answers were coded using a grounded theory approach.

In 2016 Berges et al. analyzed a huge amount of programming errors pointing to missing competences. They categorized students’ errors of an introductory programming course with regard to further analysis of the error. For that purpose, they applied the qualitative content analysis introduced by Mayring (see Section 2.2) [23].

4. Qualitative Content Analysis of Programming Errors
The qualitative content analysis [2] for texts and video sequences incorporate different steps in a fixed order. These steps are:

- Determination of the units of analysis,
- paraphrasing of content-bearing text passages,
- generalization and bundling of paraphrases, and
- collation of the new statements as a category system.
The researcher ought to write each decision into a coder manual; otherwise, they could not guarantee a reproducible analysis.

Before starting the analysis, the researcher has to determine the coding units (recording unit and context unit) \[2\]. The recording unit is the minimal component of text that can be categorized, whereas the context unit is the maximal component. For ordinary text documents possible units are: syllables, words, sentences, or paragraphs.

In contrast to the approach to analyze natural language texts as described above, there might be differences to the analysis of text written in a programming language. This includes programming errors, which are written in a programming language. Programming languages are deterministic context-free languages \[24\], which are formal languages. For that reason, we have to convert the coding units from ordinary texts (natural languages) to program code (formal languages). Possible units are: token, expressions, statements, code blocks, methods, classes, or packages. Furthermore, adaptions to the process of qualitative content analysis have to be made and are described in detail in the next sections.

The analysis steps are divided into steps that do not have an impact on the semantic of the error (paraphrasing) and those where interpretation is needed (generalization and categorization).

Figure 1 illustrates the different analysis steps and how they are aligned.

![Figure 1: Steps of the Qualitative Content Analysis](image)

**4.1 Paraphrasing**

In the first paraphrasing step, all source code components, which are not content-bearing for the error (e.g. comments, empty constructors, methods without errors), have to be eliminated. Paraphrasing does not change the semantic of the error. This step is directly adapted from the original theory of Mayring. Here, no adjustment is necessary.

If the analysis is done only for an exchange between practitioners, the paraphrasing step is finished. Otherwise, a second paraphrasing step has to be applied. The remaining source code has to be translated to a uniform stylistic level and into a grammatically standardized form. Among others, this could be the enforcement of code conventions or additional syntax elements (e.g., a leading `this` in front of all attribute names in non-declaration parts). The second paraphrasing step has to be adjusted in comparison to the original process proposed by Mayring.

The following examples illustrate the standardization:

- `i++ \to i = i + 1` (if equal in the given context)
- `getAge()` \to `getAge()`

**4.2 Generalization**

Programming languages are already abstract. For that reason, generalization results in a change of language. We propose an abstract description of the error in natural language.

The generalization is the first step in which the researcher or teacher is not acting on a syntactical level, and because of that, a precise description of the generalization has to be recorded into a coder manual. In addition to the rules that led to the generalized error description, an anchor example should be provided to clarify the decisions taken.

In the qualitative content analysis of natural language, the generalization does not enforce a change in language. As mentioned above, we had to adjust this in comparison to the original theory.

**4.3 Categorizing**

In the categorization step, the generalized error descriptions have to be interpreted and, afterwards, rewritten in a more abstract way to build categories for similar errors. Here, again, the rules for the interpretation have to be written down into a coder manual – integrating examples. Based on the content of the coder manual, the recoding for quality checks is done. This is why this step has to be done very carefully. In comparison to the original step of Mayring, no adjustment had to be done.

**4.4 Checks**

Following the qualitative content analysis for texts, the matching between the categories and the program code errors has to be validated. To gain reliability \[2\] and objectivity, intra-coder checks and inter-coder agreements have to be calculated. Thus, different researchers or teachers in computer science education should be able to replicate the same or at least similar categorization and interpretation of programming errors by using the same coder manual.

Usually, inter-coder reliability is assessed by applying a suitable quantitative measure taken from the literature. In his overview, von Eye \[25\] describes three different coefficients. The simplest one just expresses the raw agreement by the ratio of the agreements relative to the sample size. The other two coefficients are known as Cohen’s kappa and the kappa-coefficient of Brennan and Prediger (for a detailed description see \[26\]). Both coefficients calculate the ratio of agreements relative to the sample size and adjust the value by a ratio of agreement by chance. Another coefficient, called Krippendorff’s alpha was proposed by Krippendorff \[27\]. It calculates the ratio of observed disagreement relative to the expected disagreement.

All these coefficients are based on three assumptions. First, the rated objects (usually documents) have to be independent. Second, the coders operate independently, and third, they assume the categories to be independent, mutually exclusive and exhaustive.

**5. Illustration**

To illustrate the single steps, we present two different examples. They only illustrate the methodology. Obviously, the generalization and categorization is content of further discussion that is not in the focus of this paper. In general, the proposed errors are only candidates of what could have gone wrong. We
present both, errors that imply wrong “behavior” (run-time errors) and errors that occur during compilation.

The first example leads to an ambiguous error categorization which implies further investigation that is out of the focus of this paper. It illustrates the application of the methodology on a run-time error. The second example leads to an unambiguous categorization of a compile-time error. The resulting category matches a common misconception that is described in literature.

In a first step the recording unit as well as the context unit have to be defined for both examples:

- coding unit: token
- context unit: class.

5.1 Run-Time Error

The first example is a Java class DUCK, which represents a DUCK for a simulation program of a lake. In this example a duck should have a name, a weight and an age and is able to swim, walk, make a combination move, and is able to provide information about itself. Unfortunately, the methods print wrong information.

This example is from a student of secondary school, at the beginning of the 10th grade in Germany.

In this case, the coder could decide to deviate from the official code convention of Java. In this example, other code conventions were taught to the students; classes are often written with capital letters and methods are starting with a capital letter. If the coder decides to follow the convention of the school instead to official code convention, he/she has to write it down into a coder manual.

```java
class DUCK
{
    private String name;
    private double weight;
    int age;

    public DUCK(String n, double w, int a)
    {
        n = name;
        w = weight;
        a = age;
    }

    public void Swimming()
    {
        System.out.println(name + " : splash, splash..");
    }

    public void Walking()
    {
        System.out.println(name + " : waddle, waddle..");
    }

    public void Info()
    {
        System.out.println("My name is " + name + ", My weight is " + weight + " gram and I'm " + age + " years old.");
    }

    public void Combination()
    {
        Walking();
        Swimming();
    }
}
```

Paraphrasing of the first example

In the paraphrasing step, several changes are made. We decided to stick to the general Java coding standards. So, the class name (line 1), as well as the method names (for example: lines 21-24) were rewritten in the proper type case. Furthermore, in line 5 a private is added to the attribute declaration and several “this” are added to attribute calls (for example: lines 8-10). The semantic of the error is not changed by this rewriting!

```java
class Duck
{
    private String name;
    private double weight;
    private int age;

    public Duck(String n, double w, int a)
    {
        n = this.name;
        w = this.weight;
        a = this.age;
    }

    public void swimming()
    {
        System.out.println(this.name + ": splash, splash.");
    }

    public void walking()
    {
        System.out.println(this.name + ": waddle, waddle.");
    }

    public void info()
    {
        System.out.println("My name is " + this.name + ", My weight is " + this.weight + " gram and I'm " + this.age + " years old.");
    }

    public void combination()
    {
        walking();
        swimming();
    }
}
```

Generalization of the first example

A confusion of the simple assignment operator occurs inside of the constructor. The values of the attributes were assigned to the parameters.

Categorizing of the first example

In this case, two different interpretation of the error and thus two different categorizations are possible.

Confusion of attribute name and parameter name.

or

Confusion of the effect of the simple assignment operator.

The decision for interpretation has to be explained in the coder manual and discussed with a second coder if necessary.
5.2 Compile-Time Error
The second example originates from the same context as the first one. In this example, we derive a categorization that can be mapped to a common misconception described in literature. Here, we present an error that occurs during compile time. Nevertheless, the compiler message pointing on the mixture of static and non-static context will not help a student in 10th grade to solve their problem. Instead, a hint on the underlying misconception might help. Additionally, the investigation of compiler message is not expedient in terms of competence description.

```java
class DUCK {
    private String name;
    private double weight;
    private int age;

    public DUCK(String n, double w, int a) {
        this.name = n;
        this.weight = w;
        this.age = a;
    }

    public void Swimming() {
        System.out.println(name + " : waddle, waddle.. ");
    }

    public void Walking() {
        System.out.println(name + " : splash, splash.. ");
    }

    public void Info() {
        System.out.println("My name is " + name + ". My weight is " + weight + " gram and I'm " + age + " years old.");
    }

    public void Combination() {
        DUCK.Walking();
        DUCK.Swimming();
        DUCK.Swimming();
    }
}
```

Paraphrasing of the second example
Here, only type case changes had to be made. For example, the class name is rewritten from DUCK to Duck.

```java
class Duck {
    private String name;
    private double weight;
    private int age;

    public Duck(String n, double w, int a) {
        this.name = n;
        this.weight = w;
        this.age = a;
    }

    public void Swimming() {
        System.out.println(name + " : splash, splash.. ");
    }

    public void walking() {
```

5.3 General Application
Basically, our work was inspired by the investigation of programming errors to find possible misconception. For describing the errors and misconceptions, a standardized methodology is necessary as different researchers and practitioners need an intersubjective basis to communicate.

Another application can be seen in the second example. The standardized methodology enables to compare errors and misconceptions in different sources. Even for a comparison with other research papers on programming misconceptions and errors is possible.

Last, the assessment of programming tasks is a big challenge. A standardized methodology and especially a well written coder manual can provide the opportunity to do this beyond syntactical checks or unit tests. Here, the presented methodology has a high implication for secondary education.

6. Conclusion
This paper introduced the adaptation of a widely known methodology for qualitative content analysis on source code errors. The four steps “determination of the units of analysis”, “paraphrasing”, “generalization of paraphrases”, and “collation of the new statements as a category system” could be partly adapted in a direct way or had to be slightly adjusted to fit the context of analyzing programming errors and the surrounding program code. In two examples, we illustrated the introduced methodology and presented some suggestions for the application. This qualitative analysis could be used for interpreting and categorizing source code errors in a reliable way. In future we will apply this method to source code errors of a big data base of 12,000 projects.
REFERENCES


