A Template–Based Approach to Describing Metamorphic Relations

Sergio Segura, Amador Durán, Javier Troya and Antonio Ruiz Cortés
Department of Computer Languages and Systems
University of Seville, Seville, Spain
{sergosegura,amador,jtroya,aruiz}@us.es

Abstract—Metamorphic testing enables the generation of test cases in the absence of an oracle by exploiting relations among different executions of the program under test, called metamorphic relations. In a recent survey, we observed a great variability in the way metamorphic relations are described, typically in an informal manner using natural language. We noticed that the lack of a standard mechanism to describe metamorphic relations often makes them hard to read and understand, which hinders the widespread adoption of the technique. In this paper, we propose a template–based approach for the description of metamorphic relations. The proposed template aims to ease communication among practitioners as well as to contribute to research dissemination. Also, it provides a helpful guide for those approaching metamorphic testing for the first time. For the validation of the approach, we used the proposed template to describe 17 previously published metamorphic relations from different domains and groups of authors, without finding expressiveness problems. We hope that this work eases the diffusion and adoption of metamorphic testing, contributing to the progress of this thriving testing technique.

Keywords—Metamorphic testing, metamorphic relation, templates

I. INTRODUCTION

Metamorphic testing enables the generation of test cases when the expected output of a program execution is complex or unknown [1], [2]. To that purpose, rather than checking the correctness of each individual program output, metamorphic testing checks whether multiple executions of the program under test fulfill certain conditions referred to as metamorphic relations. A metamorphic relation is a necessary property of the program under test that relates two or more input data and their expected outputs, e.g. \( \sin(x) = \sin(-x) \). In the last two decades, hundreds of metamorphic relations have been reported in a variety of domains including web services and applications [3], computer graphics [4], compilers [5], machine learning [6] and cybersecurity [7].

In a recent survey, some of the authors reviewed 119 papers on metamorphic testing published in the last two decades [8]. We observed that most metamorphic relations are informally described using natural language, which may lead to misunderstandings and communication problems among researchers and practitioners. We also found that key information about the relations was often omitted or simply assumed to be known by the reader. Additionally, we found a great variability in the way metamorphic relations are described, which makes them hard to read and understand. We think that this variability could be explained by the degree of expertise on the technique. We observed that experienced researchers tend to clearly identify metamorphic relations including helpful data as identifiers, preconditions or examples. Conversely, newcomers on the technique usually describe the relations informally as a part of the main research text, omitting key information like a precise definition of the program’s inputs and outputs. Finally, some authors have proposed to use formal notation to describe metamorphic relations, but their approaches have not been widely adopted probably due to the difficulty to be understood by all stakeholders [9].

The problem of capturing and expressing information in a way that it is understandable for users with different degree of expertise has been addressed in fields such as requirements engineering [10], experimentation [11] and software metrics [12], [13]. A classical approach to address this problem is the use of templates. A template is a combination of placeholders and linguistic formulas used to describe something in a particular domain, e.g. an experiment. Templates facilitate communication among practitioners, contribute to research dissemination, and provide a helpful guide for beginners.

In this paper, we present a template–based approach for describing metamorphic relations. The proposed template is based on the structure of metamorphic relations observed in the literature, and it is also inspired by related and widely adopted templates in various fields of software engineering [10], [11], [12], [13]. The template is intentionally simple and flexible to foster its adoption by the metamorphic testing community. To this purpose, the template specifies what data should be included in the description of a metamorphic relation, but not how it should be specified: using natural language, formal languages, or a combination of both. To support the evolution of the template (e.g. when feedback from other researchers is received), it has been fully specified in a separated document subjected to version control and accessible on the Web [14]. For the evaluation of our approach, we used the template to describe 17 previously published metamorphic relations from different domains and groups of authors. This helped us to refine and validate the template, which showed to be expressive enough to represent all the subject relations.

The remainder of this paper is structured as follows. In Section II, we introduce and formalize the concepts of metamorphic relation and metamorphic testing. Section III presents the proposed template. Several examples of metamorphic
relations described with the proposed template are presented in Section IV. Section V describes the validation procedure. The related work is discussed in Section VI. Finally, we summarize our conclusions and the outlook to future work in Section VII.

II. METAMORPHIC TESTING

Metamorphic testing provides a mechanism to test a program when the expected output of a test execution is unknown or hard to compare with the actual output; this is known as the oracle problem [15]. To this purpose, instead of checking individual program outputs, metamorphic testing exploits the relations among the inputs and outputs of multiple executions of the program under test. Each of those relations is referred to as a metamorphic relation. For instance, consider the program avg(a, b), which returns the average value of two integers a and b. The order of the parameters should not influence the result, which can be expressed as the following metamorphic relation: \( \text{avg}(a, b) = \text{avg}(b, a) \). A metamorphic relation consists of the so-called source test cases, e.g. \( \text{avg}(a, b) \), and one or more follow-up test cases derived from the source test cases, e.g. \( \text{avg}(b, a) \). A metamorphic relation can be instantiated into one or more metamorphic tests by using specific input values, e.g., \( \text{avg}(2, 3) = \text{avg}(3, 2) \). If the outputs of a source test case and its follow-up test case(s) violate the metamorphic relation, the metamorphic test is said to have failed, indicating that the program under test contains a bug.

Metamorphic testing was introduced as an approach to reuse existing test cases by Chen et al. back in 1998 [1]. Since then, the research community has realized the potential of the technique to alleviate the oracle problem and to enable the automated generation of test cases. In the last two decades, a vast array of applications and innovative improvements to the automated generation of test cases, especially in\( ^1 \) average, have been proposed. RapidMiner [16] or the NASA Data Access Toolkit [17].

In what follows, we present the formal definitions of metamorphic relation and metamorphic testing, used as the basis for the design of the proposed template.

A. Formal model for a metamorphic relation

In the seminal work by T. Y. Chen et al. [1], a metamorphic relation with respect to a function \( f \) was defined as a logical implication between a relation \( R_i \) defined over a sequence of inputs \( \langle x_1, \ldots, x_n \rangle \) where \( n \geq 2 \), and a relation \( R_o \) defined over the corresponding sequence of outputs \( \langle f(x_1), \ldots, f(x_n) \rangle \), i.e.

\[
R_i \left( \langle x_i \rangle_{i=1..n} \right) \Rightarrow R_o \left( \langle f(x_i) \rangle_{i=1..n} \right)
\]

Although this definition was expressive enough for simple metamorphic relations such as:

\[
x_2 = x_1 + 360^\circ \Rightarrow \sin(x_1) = \sin(x_2)
\]

it has been later updated in order to make it more generally applicable [4], [18], [19]. In these new definitions, a metamorphic relation with respect to a function \( f \) is defined as a relation over a sequence of inputs \( \langle x_1, \ldots, x_n \rangle \) where \( n \geq 2 \) and their corresponding sequence of outputs \( \langle f(x_1), \ldots, f(x_n) \rangle \), i.e.

\[
R \left( \langle x_i \rangle_{i=1..n}, \langle f(x_i) \rangle_{i=1..n} \right)
\]

In this definition, the sequence of \( n \) inputs \( \langle x_i \rangle \) is the concatenation (using \( \bowtie \), the sequence concatenator operator) of two subsequences: the subsequence of \( m \) source test cases inputs \( \langle x_i \rangle \) and the subsequence of \( (m-n) \) follow-up test cases inputs \( \langle x_j \rangle \), i.e.

\[
\langle x_i \rangle_{i=1..n}, \langle x_j \rangle_{j=m+1..n} \]

The follow-up inputs can be determined from the source inputs and their corresponding outputs using follow-up generator functions (also called mappings in [19]), i.e.

\[
\langle x_j \rangle_{j=m+1..n} \Rightarrow G \left( \langle x_i \rangle_{i=1..m}, \langle f(x_i) \rangle_{i=1..m} \right)
\]

For example, in the previous example, the follow-up generator function (using only the source test cases but not their outputs) is \( G(x) = x + 360^\circ \).

In practice, and in line with the formal definition given by Chan et al. in [4] and [18], we have observed that metamorphic relations are expressed as logical implications using as an antecedent a relation defined over the source inputs, their outputs and the follow-up inputs (including a source-to-follow-up mapping), and as a consequent a relation over all inputs and outputs, especially follow-up outputs not included in the antecedent. Thus, a metamorphic relation can be formally described as follows:\( ^1 \)

\[
R_i \left( \langle x_k \rangle_{k=1..m}, \langle x_j \rangle_{j=m+1..n}, \langle f(x_k) \rangle_{k=1..m}, \langle f(x_j) \rangle_{j=1..n} \right) \Rightarrow R_o \left( \langle x_i \rangle_{i=1..n}, \langle f(x_i) \rangle_{i=1..n} \right)
\]

B. Formal model for metamorphic testing

Following T. Y. Chen et al. [19], the metamorphic testing of an implementation \( P \) of a function \( f \) with a metamorphic relation defined is the process by which the metamorphic relation is checked but replacing the function by its implementation, i.e.

\[
R_i \left( \langle x_k \rangle_{k=1..m}, \langle x_j \rangle_{j=m+1..n}, \langle f(x_k) \rangle_{k=1..m}, \langle P(x_j) \rangle_{j=1..n} \right) \Rightarrow R_o \left( \langle x_i \rangle_{i=1..n}, \langle P(x_i) \rangle_{i=1..n} \right)
\]

This replacement is also applied in the generation of the follow-up inputs if needed, i.e.

\[
\langle x_j \rangle_{j=m+1..n} \Rightarrow G \left( \langle x_k \rangle_{k=1..m}, \langle P(x_k) \rangle_{k=1..m} \right)
\]

If the relation is not satisfied in all the generated follow-up test cases, i.e. pairs \( (x_j, P(x_j)) \), the metamorphic testing of \( P \) reveals that it is faulty.

\( ^1 \)Although this expression can be simplified to \( R_i(x_i, f) \Rightarrow R_o(x_i, f) \), and even to \( R_i(x_i, f) \) as shown in [19], we prefer to explicitly differentiate between the source and the follow-up test cases, and to remark the logical implication.
III. TEMPLATE FOR METAMORPHIC RELATIONS

In this section, we present the proposed template to describe metamorphic relations based on the formal definition presented in the previous section. For its design, we took inspiration from some related approaches such as the Goal–Question–Metric (GQM) template used in the field of software measurement [20], [12], [13], and in particular from the GQM-based template proposed by Wholin et al. [11] in the context of software experimentation (see the related work Section for details). In syntactic terms, we have opted by a textual format rather than a tabular structure because we find it more natural and easy to include in research papers. To facilitate its integration into formal documents, a LaTeX template is provided as supplemental material2.

The template for describing metamorphic relations is shown below, where the placeholders are depicted between < and > and optional sections are enclosed between square brackets.

In the domain of <application domain>
[where <context definition>]
[assuming that <constraints>]
the following metamorphic relation(s) should hold
  • <metamorphic relation name1>:
    if <relation among inputs/outputs>
    then <relation among inputs/outputs>
    
  • <metamorphic relation name2>:
    if <relation among inputs/outputs>
    then <relation among inputs/outputs>

The template placeholders have the following meaning:

application domain
This is the application domain in which the metamorphic relations apply. For example: general domains such as search engines, code obfuscators or machine learning; specific versions of software tools such as Weka 2.1; software services like Google search, etc.

context definition
The context definition includes all necessary definitions of concepts, variables, notations, etc. used in the definition of the metamorphic relations and that are essential for their proper understanding. The section containing this placeholder is considered as optional because depending on the complexity of the metamorphic relations and the degree of formalization, could not be strictly necessary.

constraints
In this optional section, some constraints can be specified indicating necessary conditions for the metamorphic relation to be applicable.

metamorphic relation name
This is the name of the metamorphic relation being defined. Ideally, it could be a meaningful name, but a simple label is also acceptable in order to distinguish it from other metamorphic relations defined in the same template.

relation on inputs & outputs
These are logical implications in which both the antecedent (the if placeholder) and the consequent (the then placeholder) are relations defined over the function inputs and outputs.

IV. EXAMPLES

In this section, some examples of use of the proposed template are provided, illustrating different definition approaches. In all the examples, we have tried to follow as much as possible the names and the definition style used by the original authors.

A. Simple descriptions

In this section, we show how the template can be used to describe metamorphic relations in a simple way. The relation below (named MR1 as in the original paper) was presented in the context of cybersecurity by Chen et al. [7]. Roughly speaking, it states that the obfuscated version of equivalent programs should also be functionally equivalent. Note that the description has neither context definition nor constraints, as these are optional sections.

In the domain of cybersecurity (code obfuscators)
the following metamorphic relation(s) should hold
  • MR1:
    if two different source programs, P1 and P2, are functionally equivalent
    then their obfuscated versions, O(P1) and O(P2), should also be functionally equivalent and, therefore, the compiled obfuscated executable programs, C(O(P1)) and C(O(P2)), should have equivalent behavior, i.e. the same outputs for the same inputs.

The following metamorphic relation (MRorder) was proposed by Lindvall et al. [21] to address acceptance testing of NASA’s Data Access Toolkit (DAT). DAT is a large database of telemetry data collected from different NASA missions, including an advanced query interface to search and mine available data. Metamorphic testing was used by formulating the same query in different equivalent ways, and asserting that the resulting datasets were equal to each other.

2https://gestionproyectos.us.es/projects/mr-template/wiki
In the domain of the NASA’s Data Access Toolkit (DAT) where

- \( q_1, q_2, \ldots, q_n \) with \( n \geq 2 \) are equivalent search queries for the DAT system, i.e., they have the same parameters and parameter values, and return the same result set
- \( R(q_i) \) is the result set returned by query \( q_i \).

the following metamorphic relation(s) should hold

- **MRorder**: if the order of the parameters in any of the queries \( q_1, q_2, \ldots, q_n \) is changed, then \( R(q_1) = R(q_2) = \ldots = R(q_n) \), i.e. all queries return the same result set.

B. Descriptions with constraints

Some metamorphic relations may include constraints that limit their applicability to a subset of the input or output space. For instance, Zhou et al. [3] proposed several metamorphic relations to address metamorphic testing of online search engines. To avoid inaccuracies caused by the search engine optimizations (such as results being omitted to improve response time), the applicability of the relations was restricted to searches that returned between 1 and 20 results. The description of one of these relations (named MPReverseJD as in the original paper), including the previous constraint, is shown next.

In the domain of Google search assuming that

- the number of results of the source search query is greater than zero and less than or equal to 20 in order to avoid inaccuracy caused by empty and large result sets

the following metamorphic relation(s) should hold

- **MPReverseJD**: if the search terms of a given query are set in reverse order, then the result of the new query must be similar to the results of the former one applying Jaccard similarity.

C. Descriptions with a common context definition

The following instance of the proposed template shows several metamorphic relations sharing the same context definition. We found that this is a common scenario in the metamorphic testing literature. These relations were proposed in the context of metamorphic testing of machine learning classifiers by Xie et al. [6]. Note that, as in the original paper, the names of the metamorphic relations include both an identifier and a descriptive sentence.

In the domain of machine learning classifiers where

- \( S \) is the training data set.

- \( t_i \) is a source test case, i.e., a data sample \( \langle a_0, a_1 \ldots a_{m-1} \rangle \)
- \( l_i \) is the class label obtained as the output of \( t_i \).
- an uninformative attribute is one that is equally associated with each class label.

the following metamorphic relation(s) should hold

- **MR-2.1 Addition of uninformative attributes**: if in the follow-up input, an uninformative attribute is added to each sample in \( S \) and to \( t_i \), then the output of the follow-up test case should still be \( l_i \).
- **MR-5.1 Removal of classes**: if in the follow-up input, we remove one entire class of samples in \( S \) of which the label is not \( l_i \), then the output of the follow-up test case should still be \( l_i \).

D. Formal descriptions

The proposed template can be used to describe metamorphic relations using both natural and formal languages, or a combination of both. To illustrate this, some of the relations presented by Chen et al. [7] and Zhu et al. [3] are described below using a formal style, including also natural language in order to make the formal expressions easier to understand. Note that two of the relations (MR\(_1\) and MPReverseJD) were informally described in previous sections. This shows how the template can be used to describe the same relations with an informal or a formal style, and with a different level of detail.

In the domain of cybersecurity (code obfuscators) where

- \( p, p_1 \) and \( p_2 \) are computer programs.
- \( \Omega \) is a program obfuscation function.
- \( \Omega(p)[t_i] \) is the obfuscation of \( p \) at a given time \( t_i \).
- \( \equiv \) is the program functional equivalence relation.

the following metamorphic relation(s) should hold

- **MR\(_1\)**: if \( p_1 \equiv p_2 \), i.e. \( p_1 \) and \( p_2 \) are functionally equivalent, then \( \Omega(p_1) \equiv \Omega(p_2) \), i.e. the obsfuscations of \( p_1 \) and \( p_2 \) are also functionally equivalent.

- **MR\(_2\)**: if \( \{ t_i \}_{i=1..n} \) are different times, then \( \forall i : 1..n-1 \bullet \Omega(p)[t_i] \equiv \Omega(p)[t_{i+1}] \), i.e. the obfuscation process does not depend on the obfuscator environment (time of execution in this case).

In the domain of Google search where

- **site**: is a Google search operator that specifies domains, e.g. site:nbc.com.
• $q_1$ and $q_2$ are queries represented as sequences of conjunctive search criteria, i.e. $q_i = \langle c_j \rangle_{j=1..n}$
• an exact word or phrase is a search criterion, e.g. “side effect of antibiotics in babies”.
• $site:d$ is a search criterion.
• $R(q)$ is the result set of web pages returned by a given query $q$, i.e. $R(q) = \{ p_k \}_{k=1..m}$
• $\#R(q)$ is the size of $R(q)$.
• $\cap$ is the sequence concatenation operator.
• $rev$ is the reverse sequence function, i.e. $q = \langle c_j \rangle_{j=1..n} \Rightarrow rev(q) = \langle c_j \rangle_{j=n..1}$

assuming that

0 < $\#R(q_1)$ ≤ 20

the following metamorphic relation(s) should hold

MPSite:

if $q_2 = q_1 \cap site:d$ where $d$ is the domain of one of the web pages in $R(q_1)$
then $R(q_2) \subseteq R(q_1)$, i.e. the results of $q_2$ must be a subset of the results of $q_1$

MPReverseID:

if $q_2 = rev(q_1)$, i.e. $q_2$ is the reverse of $q_1$
then $R(q_2) \approx R(q_1)$, i.e. the results of $q_2$ are similar to the results of $q_1$ applying Jaccard similarity.

As a further example, the following is a formal description of one of the metamorphic relations presented by Segura et al. [22] in the context of feature model analysis tools.

In the domain of feature model analysis tools

• $M$ is a feature model.
• $\Pi(M)$ is a function returning the set of products of a feature model $M$.
• $\#$ is the cardinality function on sets.

the following metamorphic relation(s) should hold

MRMandatory:

if $M'$ is derived from $M$ by adding a mandatory feature $f_m$ as a child feature of $f_p$
then $\#\Pi(M') = \#\Pi(M) \land \forall p \in \Pi(M) \bullet$

$f_p \notin p \Rightarrow p \in \Pi(M') \land$

$f_p \in p \Rightarrow (p \cup \{f_m\}) \in \Pi(M')$

V. VALIDATION

For the validation of our approach, we used the proposed template to describe several metamorphic relations found in the literature, trying to identify expressiveness problems. To this purpose, we selected 10 metamorphic testing papers, from 35 different authors and 8 different application domains, from which 17 metamorphic relations were selected to be described using our approach. We may remark that these relations were randomly selected with the only purpose of having a representative pull of metamorphic relations, and not because we identified any specific limitations in them. Table I depicts the list of selected papers including publication year, short list of authors, title, application domain, and reference. Five of the papers were published in journals, and five in conferences or workshops. Some of the selected metamorphic relations were presented in the previous section to illustrate the use of the template. Owing to space limitation, the full set of relations are described in a technical report [14].

The validation process was iterative. First, the selected papers were divided and distributed among three of the authors. Each author randomly selected between one and three metamorphic relations from each paper and described them using the template. Then, several meetings were arranged to discuss the raised issues such as the exact linguistic forms used or how expressing multiple relations sharing the same context (where section of the template, see Section III). Some of these issues where addressed by introducing minor changes in the template. The process was then repeated by updating the described relations and debating about them, until no expressiveness problem was found and a strong consensus was reached among all the authors.

VI. RELATED WORK

In this section, we review some related templates in the context of software engineering.

The Goal–Question–Metric (GQM) method proposes a template to support software measurement [20], [12], [13]. The approach is based on the idea that for an organization to measure any aspect of software development it must first specify the goals, including the object for which it is defined, and the interested viewpoint, e.g. improve the timeliness of change request processing from the project manager’s viewpoint. Then, the organization must trace those goals to a number of questions to characterize how the achievement of a specific goal is going to be performed, e.g. what is the current change request processing speed? Finally, a number of metrics must be defined in order to answer each question quantitatively, e.g. average change processing time.

Durán et al. [10] proposed several templates and linguistic patterns to assist engineers when gathering and documenting user requirements. Templates were given in a tabular format. For each type of requirement (e.g. information requirements, use cases, etc.) a number of fields were recommended such as author, purpose, description, time interval or importance degree. Additionally, some parameterized linguistic patterns were also proposed and introduced as a part of the templates, e.g. “The system shall store the information corresponding to <relevant concept>”. Their approach is integrated into the requirements management tool REM [27].

Wholin et al. [11] proposed using the GQM template to describe experiments in software engineering. In particular, the authors proposed using the goal template by Basili and Rombach [20] as follows:

Analyze <object(s) of study> for the purpose of <purpose>
with respect to their <quality focus> from the point of view of the <perspective> in the context of <context>.

The template presented in this paper is mainly inspired by the GQM template proposed by Wholin et al. in the context of experimentation. We chose a textual format because it reads more naturally than tables and it is easier to integrate in research papers. However, it would be straightforward to translate it into a tabular format. In contrast to domain-specific approaches as the one presented by Durán et al. [10] in the domain of requirement engineering, the proposed template aims to be as generic as possible, making it suitable to describe metamorphic relations from multiple domains using different notations and levels of abstraction.

VII. CONCLUSION AND FUTURE WORK

Metamorphic testing is becoming a well-established discipline with new applications, techniques, and empirical studies rapidly proliferating. However, the lack of standard mechanisms to describe metamorphic relations is becoming an obstacle for the dissemination of research results, and eventually for its adoption by industry. Inspired by how related fields have approached this problem, in this paper we have proposed a template for the description of metamorphic relations. The template is intentionally simple and flexible, allowing researchers to describe metamorphic relations in their own way, but adhering to a basic structure in order to make relations easy to read and understand. The template is fully specified in a separated document subjected to version control in order to support its evolution. We trust that this template eases the adoption and dissemination of metamorphic testing, contributing to the progress of this promising testing technique.

Several challenges remains for future work. A more rigorous validation of the proposed template should involve empirical studies with human subjects. For instance, evaluating whether people with different expertise can understand the template and use it to describe metamorphic relations consistently. Also, the mapping from template-based metamorphic relations to executable code or test cases is a challenging topic that will require further research.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their helpful comments and suggestions. This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project BELI (TIN2015–70560–R), and the Andalusian Government project COPAS (P12–TIC–1867).

REFERENCES


