Journal of Software: Evolution and Process

JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw. Evol. and Proc. (2013)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/smr.1596

Metrics for quantifying evolutionary changes in Z specifications

Andreas Bollin*,†

Software Engineering and Soft Computing, Alpen-Adria Universität Klagenfurt, 9020 Klagenfurt, Austria

SUMMARY

This article proposes metrics for quantifying changes throughout the evolution of formal software specifications in long living systems. Formal specifications play an important role in the software development life-cycle by supporting refinement and proof and by providing a basis for comprehension and maintenance activities. However, specifications also undergo evolutionary changes, and these changes are hard to assess because of a lack of suitable measures.

This paper proposes and analyzes a set of measures for estimating aspects of a specification's complexity and quality. The measures are based on existing measures for source code, but they have been redefined in the scope of formal Z specifications. Geared towards Z, they are then evaluated concerning their expressiveness by a case study that comprises more than 65,000 lines of specification text. Finally, the usability of the measures is demonstrated on the Z specification of the Web Service Definition Language during its evolution over a period of about 3 years. Copyright © 2013 John Wiley & Sons, Ltd.

Received 15 February 2012; Revised 3 February 2013; Accepted 6 March 2013

KEY WORDS: evolution; deterioration; slice-based measures; Z specifications

1. INTRODUCTION

Formal specifications can be very useful artifacts at various stages of software development. They enable verification and validation, but they also form the basis for ongoing refinement steps [1]. Although there are several myths around [2, 3], there are also a lot of reasons for using them to improve the quality of the systems to be generated [4–7].

Another, sometimes underestimated, benefit arises when specifications are kept up-to-date during the evolutionary stages of the development. This facilitates ongoing maintenance activities. Because formal specifications are close to the requirements, these documents are usually descriptions at a high level of abstraction and thus accelerate the underlying comprehension process.

The use of formal specifications looks like a silver bullet, but the benefits are also put into question. It is interesting to ponder the reasons for this situation. The old management adage 'You can't manage what you don't measure' might be a hint towards an answer. Fenton and Kaposi ([8], p. 7) already mention it when they write that *[. . .] in the absence of a suitable measurement system, there is no chance of validating the claims of the formal methods community [. . .]*. Hall *et al*. also state that formal methods do have a lot of benefits, but they also admit that it really is *difficult to be more precise* ([9], p. 22).

Since the 1990s, the situation has definitely improved. But, compared with the number of articles regarding software measures, the amount of studies dealing with specification measures is still small. As mentioned in Section 6, Related Works, developers basically do count lines of specification text

*Correspondence to: Andreas Bollin, Software Engineering and Soft Computing, Alpen-Adria Universität Klagenfurt, 9020 Klagenfurt, Austria.
†E-mail: Andreas.Bollin@uni-klu.ac.at

or the number of predicates. But other measures closer to the aspects of the semantics of a specification are not in use. One of the impediments when trying to reuse existing measures is the structural difference between formal specification languages and programming languages. Due to the declarative nature of specifications, concepts such as control flow or data flow are not predominant. But, these concepts are very often used to form the basis for widespread software measures, and so it becomes clear why it is not easy to apply 'traditional' software measures to formal specifications.

This contribution tries to demonstrate that this situation can be changed. With the reconstruction of control and data dependencies [10], 'classical' measures can be mapped to specifications, too. Bollin [11] demonstrated that slice-based coupling and cohesion measures can be reasonably mapped to formal Z specifications. Using them, the existing set of measures can be extended by semantics-based measures, too. What is left (and will be shown in this paper) is to check whether these measures are unique views onto the specifications, to ensure that they are not only proxies for size/quantity-based measures, and that they can be used to talk about possible deterioration effects.

Motivated by this situation and by a study by Meyers and Binkley [12] where they are looking at coupling and cohesion-based software measures, small, medium, and large-size Z specifications have been collected and used for a thorough evaluation of the different measures. The evaluation is intended to be comparable with the study by Meyers and Binkley, and thus, this paper follows the structure of their work for the most part.

In total, more than 65,000 lines of specification text have been taken as a basis for this study. The main steps of this study are:

- The study compares size-based/structure-based and semantics-based measures head-to-head. The comparison shows which of these measures are strongly related and which of them are providing unique views of the specification. Section 4.4 shows that there really are measures that are not just proxies for counting lines (or predicates) in a specification text.
- The paper analyzes the effect of evolutionary changes of the specification with respect to the measures. One longitudinal study, based on the specification of the Web Service Definition Language, is presented. Section 4.5 shows that some of the measures are quite sensitive to changes in the specification.
- Finally, in Section 4.6, the study presents baseline values for the measures, depending on the sizes of the specifications under study.

This paper is structured as follows. Section 2 provides the necessary terminology for the study. Ensuing, a set of existing measures is presented and discussed in Section 3. Then, on the basis of a large set of sample specifications, in Section 4, the measures are evaluated and assessed in respect to their expressiveness. Possible threads regarding validity are discussed in Section 5. Related work is discussed in Section 6, before Section 7 summarizes the most important findings. Finally, the Appendix summarizes the most important operations and definitions used in this work.

2. MEASUREMENT BASIS

The objective of this section is to briefly introduce the elements and structures of a formal specification being measured. Most of the measures are calculated on the basis of specification dependencies and slices, so this section addresses these concepts in some details. Background information, formal definitions, and further examples can be found in the articles on specification slicing and measurement by Mittermeir and Bollin [10, 13, 11]. All the necessary articles can be downloaded from the web page of the ViZ project,* a project at the University of Klagenfurt that aims at creating a tool set for increasing the comprehensibility of Z specifications.

2.1. Specification slicing

Static program slicing was introduced by Weiser [14, 15] to compute those set of program statements that affect program values at some point of interest (called *slicing criterion*). Because the original

*ViZ – Visualization of Z Specifications. Project web page at http://viz.uni-klu.ac.at. Page last visited: Aug. 26th, 2012.

DOI: 10.1002/smr

definition there has been a substantial development of the concept (an introduction can be found in the overview paper by Tip [16]), but basically, static slices are computed by transitively relating program statements according to control and data dependencies.

As control and data dependencies are not predominant concepts in formal specifications, slicing was restricted to programming languages for a while. In 1993, Oda and Araki [17] then transferred the concept of slicing to specifications by defining a static forward slice for a formal specification based on data dependencies. A year later, Chang and Richardson [18] extended this idea by also considering control dependencies. Their approach rests on a simple idea: predicates of a specification that describe an after-state are 'control'-dependent on predicates that do not refer to an after-state. 'Data dependency', on the other hand, is defined as dependency between predicates potentially connected by data flow. By removing all the predicates that are not dependent upon a point of interest (which is one predicate or a set of predicates in the specification), static specification slices are carved out of the specification.

The approach of Chang and Richardson has been refined and extended by Mittermeir and Bollin [10, 13] in 2003. There, the different dependencies have been defined formally, and also, the procedure for carving out the elements and dependencies has been specified in an algorithmic and semantics preserving manner. The approach finally yielded a Java prototype called ViZ that is able to generate various types of partial formal specifications [20], including those of specification slices. The approach of Chang and Richardson is based on the idea that unneeded predicates are removed from a specification until the slice has been generated. On contrary, the approach of Mittermeir and Bollin starts with the point of interest, adding predicates to it by following the different dependencies in the specification. To do so, the specification is mapped to a graph in a first step, and then, in a second step, annotated with different types of dependencies. Alternative approaches, like that of Wu and Yi [21], also make use of a graph-based approach. Figures 1 and 2 visualize this basic idea behind the mapping process that will be explained briefly in Sections 2.1.1 and 2.1.2.

2.1.1. Specification primes. Figure 1 presents a simple and for demonstration purposes shortened formal Z specification of the Birthday Book specification (BBook) [19] that stores names, related birthdays, and gifts in a 'database' (represented as sets and functions). At line 1, it introduces three basic types namely *NAME*, *DATE*, and *GIFT*. Then, two boxed schemas are provided. The first one (between lines 2 and 9) is called BB (for BBook), and it represents the state of the system. The second one (between lines 10 and 19) is called *Add*, and it represents an operation schema for adding new entries to the set of names. Both boxed schemas are divided into two parts (noticeable by the horizontal lines at lines 6 and 14). It separates the declaration part from the predicate part. The declaration part introduces three basic declarations written on separate lines: a set of known names at line 3, a partial function mapping names to birth dates at line 4, and a partial function mapping names to a set of birthday presents at line 5. The predicate part of the state schema BB contains two predicates (state invariants), ensuring that, for the names stored in the database, the birthday dates are defined, and ensuring that presents are defined for existing entries only. At lines 12 and 13 two identifiers, namely *name*? and *date*?, are defined. The question mark indicates that they are input identifiers in the operation schema. The declaration part of the *Add* schema introduces another feature of the Z specification language: it is possible to include other schemas in the declaration part. At line 11, the state schema BB is included, which means that the declarations (and

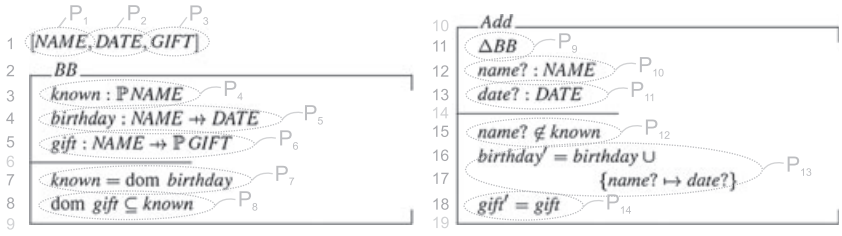

Figure 1. Example of a Z specification (a slightly modified and incomplete version of the Birthday Book specification [19]). The basic building elements are, for considerations in Section 2.1, encircled and labeled with the strings P_1 to P_{14}. Additionally, line numbers are added to the specification.

DOI: 10.1002/smr

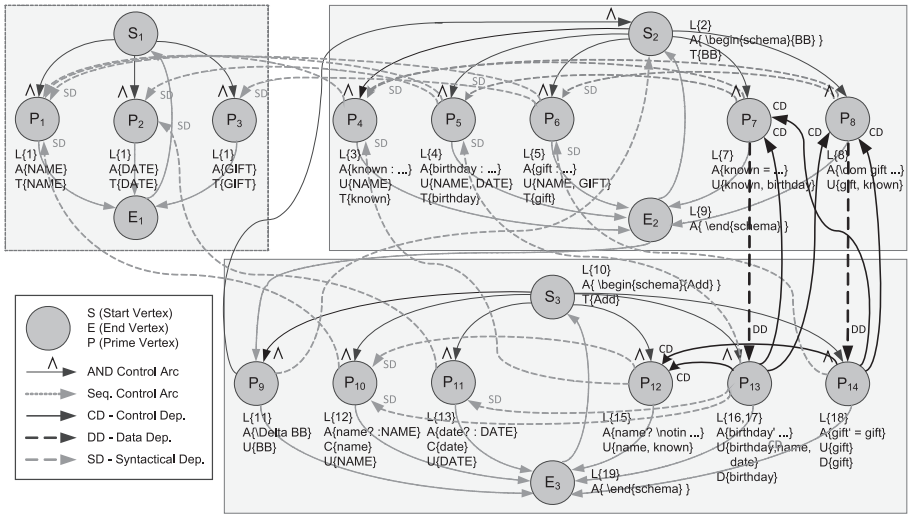

Figure 2. ASRN graph [10] for the Birthday Book specification presented in Figure 1. Predicates and declarations are represented by vertices *P*. They are enclosed between start (*S*) and end (*E*) vertices and are additionally annotated by position/line numbers, the related Z specification text, and the sets describing the use of the identifiers (D..defined, U..used, T..type-defined, C..in/out channel). Logical combinations are expressed by logical control arcs – in our example, AND control arcs.

predicates) of the included schema are now considered to be part of the actual schema. By following a naming convention, 'Δ *BB*' means that the state of BB might change because of the operation schema.[†] In the predicate section, there are three predicates. The first one at line 15 ensures that the name taken as input is not part of the database already. The second and third (at lines 16–18) describe the after-state of the two functions *birthday* and *gift*, indicated by the decorated identifiers. An in-depth introduction to the syntax and semantics of Z can be found in Diller's textbook [22].

The ViZ environment makes use of the CZT parser [23] for identifying the basic elements a specification is built upon: specification primes, comparable with program statements.

Definition 1
Specification prime
A specification prime (also called prime object) represents the basic entity of a specification. It is built out of literals of the specification and forms logical, syntactic, or semantic units.

In that sense, a specification prime constitutes those primitives that form the smallest, basic concepts expressed in a formal specification. It is a *syntactically coherent sequence of literals within a state-based specification, forming semantic entities that can be paraphrased by a short sentence in natural language*. For Z specifications, a prime is either (a) a given set or (b) a free-type definition, (c) a (global) declaration or a (d) (sub)predicate (separated by a logical AND or OR from other predicates).[‡] They are identified by traversing the abstract syntax tree generated by the CZT parser and by looking for those subtrees that represent names, predicates, and expressions. In the example in Figure 1, these primes are the three type definitions (line 1), the six declarations (lines 3–5 and 11–13), and the five predicates (lines 7 and 8, line 15, lines 16–17, and line 18).

2.1.2. Graph-based slicing approach. After the identification of all primes in a specification, they are mapped to a graph as presented in Figure 2. The graph, called Augmented Specification Relationship Net (ASRN), has been introduced to generate a graphical overview of a specification's structure, but it also allows for several annotations that ease the calculation of specification abstractions. The ASRN

[†]For indicating that a state does not change, usually the notation 'Ξ *Schemaname*' is used.
[‡]In the PhD thesis of Bollin [13], there is a lot more information about the definition and the identification of Z specification primes.

DOI: 10.1002/smr

consists of vertices called 'prime vertices' representing the basic elements of the specification (in our case all definitions, declarations, and predicates of the BBook specification). It also contains arcs representing the various types of dependencies between them.[§] The ASRN in Figure 2 has already been annotated with its dependency arcs. For example, there is a control dependency between vertex P_{13} and P_{12} or a data dependency between P_7 and P_{13}

Slices are typically defined for a specific point of interest. This 'point' is, according to similar definitions in the slicing community, called the *Abstraction Criterion*. For Z, it consists of a set of primes to be regarded and the types of dependencies to be taken into account. On the basis of the abstraction criterion, a first, general definition of a specification slice can be provided:

Definition 2
Specification slice
A specification slice is a syntactically and semantically correct specification that is the result of adding those primes to an (initially empty) specification, which are directly or indirectly contributing to the abstraction criterion.

In the case of the ASRN, one has to take a set of vertices as starting point and then to follow the different arcs in the graph. For example, when starting with P_{13} and including all primes that are reachable via control and data dependency arcs (in a forward and a backward manner), then the resulting set of primes is $\{ P_7, P_8, P_{12}, P_{13} \}$. When also including primes that the primes in the slice are syntactically dependent on (by following the syntactical dependency arcs in the ASRN) then the resulting set is again syntactically correct.

2.2. *Slice profiles*

A slice profile is simply the collection of all possible slices for a given piece of software code (or specification text). Various authors demonstrate that this collection of slices can be used to calculate coupling and cohesion measures of procedural programs [24, 25, 12].

Longworth, and also Meyers and Binkley [26, 12], mention a couple of decisions to be taken when calculating slice profiles for ordinary programs. These decisions address the total number of slices to be generated, the type of slices to be calculated, and the question of correct selections of the abstraction criteria. Their considerations also affect the way of producing slices for formal Z specifications. The reason is that also in the case of specifications, the dependencies (and with them the arcs in the ASRN) do have a direction. For coupling and cohesion, as later formally defined for Z specifications in Section 3.3, the following types of slices turned out to be useful [11]:

- Cohesion. For the calculation of the different cohesion values, *transitive forward and transitive backward* slices should be used. This holds because there is no ordering of the predicates in a schema and an artificially introduced 'direction' (comparable with the notion of *VRES* slices defined by Ott and Thuss [24]) would be semantically wrong. Additionally, a single slice for only the 'last' predicate in a schema is not appropriate. There is no 'order of execution' in a schema that tells us which of the predicates to be evaluated first and which are to be evaluated next. So, all predicate primes have to be regarded, and slices have to be generated for all primes that either do change or might change the state. Concerning the ASRN, this means that all primes (vertices) representing predicates have to be taken as the abstraction criterion.
- Coupling. It turns out that the same slice type as introduced by Harman *et al*. [25] should be used. To include all primes that might eventually influence a given prime in another schema, the slice is calculated in a *transitive* manner, and as the calculation of coupling is based on Harman's notion of information flow (which implies a given direction), the slice to be taken is the *transitive backward slice*.

[§]Please note that the transformation between a specification and the ASRN is one-to-one [13]. When talking about a specification, it can be either the textual or the ASRN representation.

DOI: 10.1002/smr

For the calculation of the values, transitive forward and backward slices for Z and slice profiles are needed. The definitions are given as follows:

Definition 3
Transitive forward slice
Let Ψ be a formal Z specification, ψ one schema out of Ψ, and V a set of primes in ψ called *Abstraction Criterion*. $SSlice_f(\Psi,\psi,V)$ is called *transitive forward slice* of ψ for primes V. It contains all the primes q in V and all primes p of the specification ($p \in \Psi$) for which it holds:

- p is (transitively) reachable via data dependency arcs from a prime q in V, or
- a prime q in V is (transitively) reachable from prime p via control dependency arcs.

Definition 4
Transitive backward slice
Let Ψ be a formal Z specification, ψ one schema out of Ψ, and V a set of primes in ψ called *Abstraction Criterion*. $SSlice_b(\Psi,\psi,V)$ is called *transitive backward slice* of ψ for primes V. It contains all the primes q in V and all primes p of the specification ($p \in \Psi$) for which it holds:

- p is (transitively) reachable via control dependency arcs from a prime q in V, or
- a prime q in V is (transitively) reachable from prime p via data dependency arcs.

Forward and backward slices are easily calculated when making use of the ASRN. A transitive forward slice $SSlice_f(BBook,Add,\{P_{12}\})$ of the BBook specification in Figure 2 would yield the set of primes containing P_{12} (as it is part of the abstraction criterion) and P_{13} and P_{14} (as these primes are control-dependent upon prime P_{12}). On the other hand, a transitive backward slice $SSlice_b(BBook,Add,\{P_{13}\})$ would contain the following primes: P_{13} (as it is part of the abstraction criterion) and P_7 and P_8 (as P_{13} is control-dependent upon them).

Definition 5
Transitive forward/backward slice
Let Ψ be a formal Z specification, ψ one schema out of Ψ, and V a set of primes in ψ called *Abstraction Criterion*. $SSlice_{fb}(\Psi,\psi,V)$ is called *transitive forward backward slice* of ψ for primes V. It consists of all primes that are either in $SSlice_f(\Psi,\psi,V)$ or in $SSlice_b(\Psi,\psi,V)$.

In the case of the BBook specification in Figure 2, a transitive forward/backward slice $SSlice_{fb}(BBook,Add,\{P_{13}\})$ would contain the following primes: P_{13} (as it is part of the abstraction criterion), P_{12} (as P_{13} is control-dependent upon it), and P_7 and P_8 (as P_{13} is both, control-dependent and data-dependent, upon them).

Definition 6
Slice profiles
Let Ψ be a formal Z specification and ψ one schema out of Ψ. Additionally, let V be the set of primes v in the ASRN representing predicate primes in ψ. The collection of all transitive forward backward slices ($SSlice_{fb}(\Psi,\psi,\{v\})$ with $v \in V$) is called *forward backward slice profile* $SP^{fb}(\Psi,\psi)$. The collection of all transitive backward slices ($SSlice_b(\Psi,\psi,\{v\})$ with $v \in V$) is called *backward slice profile* $SP^{b}(\Psi,\psi)$.

Both types of slice profiles just differ in the type of slices used for their calculation. To improve the legibility of the text, both types of slice profiles ($SP^{fb}(\Psi,\psi)$ and $SP^{b}(\Psi,\psi)$) will from now on just be called *slice profile*. Only when necessary, the full name will be given.

Definition 7
Size, slice intersection, smallest slice, largest slice, slice union
Let Ψ be a formal Z specification and ψ one schema out of Ψ. Let V be the set of primes v in the ASRN representing predicate primes in ψ. Additionally, let $SP^{x}(\Psi,\psi)$ be the *slice profile* of schema ψ (where 'x' stands either for '*fb*' or '*b*'). Then it holds:

DOI: 10.1002/smr

(a) $SP^x_i(\Psi,\psi)$ denotes one specific slice out of the collection of slices in $SP^x(\Psi,\psi)$ $(1 \le i \le |V|)$.
(b) The number of slices in $SP^x(\Psi,\psi)$ is called the size of the slice profile $|SP^x(\Psi,\psi)|$.
(c) $SP^x_{min}(\Psi,\psi)$ denotes the smallest slice in the slice profile $SP^x(\Psi,\psi)$.
(d) $SP^x_{max}(\Psi,\psi)$ denotes the largest slice in the slice profile $SP^x(\Psi,\psi)$.
(e) The intersection of all slices in the slice profile is called *Slice intersection* $(SP^x_\cap(\Psi,\psi))$.
(f) The union of all slices in the slice profile is called *Slice union* $(SP^x_\cup(\Psi,\psi))$.

Table I (right part) presents a simple example of a slice profile for the *Add* operation schema of the BBook specification (left part). At first, *forward backward slices* are generated for the two predicates at lines 16/17 and 18. The resulting slices are then part of the slice profile $SP^{fb}(BBook,Add)$, consisting of the slice containing the lines 7, 8, 15, 16, 17, and the slice containing the lines 7, 8, 15, and 18. The intersection of the sets of primes yields the slice intersection $SP^{fb}_\cap(BBook,Add)$ (containing the primes at lines 7, 8, and 15), and the union of all the primes yields the Slice Union $SP^{fb}_\cup(BBook,Add)$ (containing all the primes).

3. SPECIFICATION MEASURES

As will also be discussed in the related work Section 6, most of the studies focus on size/quantity-based measures. They are easy to calculate and give a good glimpse of parts of the complexity of the underlying document. However, with the ASRN and its explicitly available dependencies, further possibilities exist. The graph, with its additional information, enables structure as well as semantics-based considerations that are standard now for programming languages. The definitions in the following section are given for formal Z specifications. It would go beyond the scope of this paper, but for other declarative, state-based formal specification languages, a mapping should be possible, too. The measures

Table I. Example of a simple Z specification (a slightly modified and incomplete version of the Birthday Book specification [19]) including its related slice profile $SP^{fb}(BBook,Add)$, the slice intersection $SP^{fb}_\cap(BBook,Add)$, and the slice union $SP^{fb}_\cup(BBook,Add)$. Slices and slice profiles are explained in more details in Section 2.2. A bar in the right columns marks those predicates (one or several lines) that are part of the slice profile, the slice intersection, or the slice union. A cross marks those primes (one or several lines) that are part of the slicing criterion.

Line	Birthday Book Specification (*BBook*)	$SP^{fb}(BBook,Add)$		$SP^{fb}_\cap$	$SP^{fb}_\cup$
1	$[NAME, DATE, GIFT]$				
2	*BB*				
3	$known : \mathbb{P}\, NAME$				
4	$birthday : NAME \pfun DATE$				
5	$gift : NAME \pfun \mathbb{P}\, GIFT$				
6					
7	$known = \mathrm{dom}\ birthday$	|	|	|	|
8	$\mathrm{dom}\ gift \subseteq known$	|	|	|	|
9					
10	*Add*				
11	ΔBB				
12	$name? : NAME$				
13	$date? : DATE$				
14					
15	$name? \notin known$	|	|	|	|
16	$birthday' = birthday \cup$	×			|
17	$\{name? \mapsto date?\}$	×			|
18	$gift' = gift$		×		|
19					

DOI: 10.1002/smr

are defined upon the ASRN, and all that has to be performed is to map the basic elements (predicates) to the graph-based structure, and then to annotate the graph with occurring dependencies.

3.1. Classes of specification measures

Because of simplicity of assessing values, two classes of measurement are commonly used since the 1970s; those based on the physical size of the source text and those based on the analysis of data and control flow, thus describing quantitative properties. In addition to these classes, qualitative measures have been developed as a third class to describe the compactness of components or functions of a system.

Class (1) The first set of measures deals with size or structure-based complexity considerations. There, quantity (or size)-based measures only consider the literals without a deeper interpretation of the text itself. A widespread metrics of this type is the number of lines of code. But, as lines of specification text are not necessarily very informative, it is more convenient to count the number of predicates or expressions.

Class (2) Structural metrics, on the other hand, usually refer to the relations between the individual sections. Two well-known (and also widely discussed) metrics are that of cyclomatic complexity *v(G)* going back to McCabe [27], and the *Definition-Use* (DU) count metric of Tai [28]. Structural metrics are often based on control and/or data flow relationships. As these dependencies are made explicit in the ASRN, they can also be calculated for Z specifications.

Class (3) Quality considerations are said to be crucial during a software project, but the term 'quality' strongly depends on the perspective one is looking upon it. Within the scope of this contribution, we focus on quality measures that describe the compactness of components or functions. Quite often they are expressed by the diametrically opposed properties of coupling and cohesion, and this study also assesses these types of measures.

Size and structure-based measures will be considered briefly in Section 3.2 and are discussed in more detail by Bollin [29]. A summary of quality-based measures is given in Section 3.3. A more detailed discussion is presented by Bollin [11].

3.2. Complexity metrics

Complexity can be seen as the difficulty of describing or understanding parts/pieces of artifacts, in our case formal Z specifications. It would be tempting to define a single value for complexity. However, complexity cannot be reduced to just one dimension. When interested in defining measures that influence the human 'costs' for the creation and understanding of a specification, then *conceptual and logical complexity* are to be considered. The following measures for these types of complexity have been suggested by Bollin [29] and their definitions are summarized in Table II:

- *Conceptual complexity* (CC) $CC(\Psi)$ of a specification Ψ. This measure equals the total number of prime vertices in the ASRN.
- *Logical complexity* $v'(\Psi)$ of a specification Ψ. On the basis of the number of control dependencies, lower and upper bounds for (logical) decisions in a specification are counted. The lower bound value l is 1 plus the number of primes that are terminal vertices of control dependence arcs in the ASRN. The upper bound value u equals 1 plus the total number of control dependencies in the ASRN. The lower bound of the measure counts the different decision statements in a specification whereas the upper bound counts all logical dependencies.

Table II. Size-based and structure-based measures for Z specifications Ψ.

Measure	*Definition*
Conceptual complexity: $CC(\Psi)$	$\#\{v : Prime \mid \Psi \text{ contains } v\}$
Logical complexity: $v'(\Psi) = (l,u)$	$l = 1 + \#\{v : \Psi \mid \exists\, a : Arc \,@\, \cdot\, a.type = CD \wedge a.dest = v\}$
	$u = 1 + \#\{a : Arc \mid \Psi \text{ contains } a \wedge a.type = CD\}$
Definition-use count: $DU(\Psi)$	$\#\{a : Arc \mid \Psi \text{ contains } a \wedge a.type = DD\}$

DOI: 10.1002/smr

- The *Du count* metric $DU(\Psi)$ of a specification Ψ. By counting the number of data dependencies the maximum number of data relationships is identified.

3.3. Qualitative measures

Weiser proposed several slicing metrics [15] but did not relate them to cohesion. He assumed a relation to cohesion, but did not examine this in detail. Longworth [26] extended this idea and Ott and Thuss [24] formally defined them by making use of slice profiles. The following measures of Ott and Thuss are redefined for formal Z specification schemas by Bollin [11] and also make use of slice profiles. For a specification Ψ, containing a schema ψ (for which n specification slices can be calculated), it then holds:

- *Tightness* $\tau(\Psi,\psi)$ is defined as the ratio of the number of primes in the intersection of all slices over the total number of prime elements in the schema.
- *Coverage (Cov)* ($Cov(\Psi,\psi)$) relates the number of primes in all n specification slices (in $SP^{fb}(\Psi,\psi)$) to the number of primes in ψ.
- *Minimum coverage* ($Cov_{min}(\Psi,\psi)$) is calculated by taking the number of primes in the smallest slice and relating it to the number of primes in the schema.
- *Maximum coverage* ($Cov_{max}(\Psi,\psi)$) is calculated by taking the number of primes in the largest slice and relating it to the number of primes in the schema.
- *Overlap (O)* $O(\Psi,\psi)$ considers the size of the slice intersection and determines how many primes are common to all n slices in $SP^{fb}(\Psi,\psi)$.

Table III provides the definitions of these measures. As the sizes of the slices are always related and restricted to the size of the schema, the resulting value is between [01]. Please note that, to reduce calculation complexity and to avoid the 'noise' of declarations as mentioned by Ott and Thuss [24], the measures are defined such that only primes representing predicates in the ASRN are considered. The cardinality operator || for a schema (or set of primes) is therefore defined as follows:

Definition 8
Cardinality
Let Ψ be a formal Z specification and S be any set of primes out of Ψ. Then, the cardinality of S is defined as follows:

$$\mid S_\Psi \mid \; = \; \#\{p : S | p.type = P \wedge p \text{ represents } a \text{ predicate prime in } \Psi\}$$

To improve the readability of the definitions in Section 3.3, the following abbreviation is introduced: when the context (the specification Ψ that the primes in S are part of) is clear, then the cardinality $|S_\Psi|$ is simply written as $|S|$.

Table III. Cohesion measures for a schema ψ in a Z specification Ψ as introduced by Bollin [11].

Measure	*Definition*
Tightness: $\tau(\Psi,\psi)$	$\frac{\mid SP_\cap^{fb}(\Psi,\psi) \;\cap\; \psi \mid}{\mid \psi \mid}$
Coverage: $Cov(\Psi,\psi)$	$\frac{1}{n}\sum_{i=1}^{n} \frac{\mid SP_i^{fb}(\Psi,\psi) \;\cap\; \psi \mid}{\mid \psi \mid}$ (with $n = \mid SP^{fb}(\Psi,\psi) \mid$)
MinimumCoverage: $Cov_{min}(\Psi,\psi)$	$\frac{1}{\mid \psi \mid} \left\mid SP_{min}^{fb}(\Psi,\psi) \;\cap\; \psi \right\mid$
MaximumCoverage: $Cov_{max}(\Psi,\psi)$	$\frac{1}{\mid \psi \mid} \left\mid SP_{max}^{fb}(\Psi,\psi) \;\cap\; \psi \right\mid$
Overlap: $O(\Psi,\psi)$	$\frac{1}{n}\sum_{i=1}^{n} \frac{\mid SP_\cap^{fb}(\Psi,\psi) \;\cap\; \psi \mid}{\mid SP_i^{fb}(\Psi,\psi) \;\cap\; \psi \mid}$ (with $n = \mid SP^{fb}(\Psi,\psi) \mid$)

DOI: 10.1002/smr

The calculation of the value of coupling follows the definitions of Harman *et al*. [25]. It has also been mapped to formal Z specifications by Bollin [11]. The approach is simple: first, the interschema flow between two schemas is specified and then taken as the basis for the calculation of the flow between all the schemas in the specification. This leads to the following set of measures (summarized in Table IV) defined for formal Z specifications:

- *Interschema flow* (Ψ,ψ_s,ψ_d) that measures the number of primes of the slices in ψ_d that are in ψ_s,
- *Interschema coupling* $C(\Psi,\psi_s,\psi_d)$ that computes the normalized ratio of this flow in both directions, and
- *Schema Coupling* $\chi(\Psi,\psi_i)$ of a schema which is the weighted measure of *Interschema coupling* of ψ_i and all n other schemas in Ψ.

3.4. Deterioration of Z specifications

Ott and Thuss showed that slice-based measures are sensitive to changes and that they have the potential to quantify deterioration processes [30]. This sensitiveness was also demonstrated on sample Z specifications by Bollin [11], and the idea is now to make use of the measures to estimate the effect of maintenance actions applied to formal Z specifications.

The basic idea goes back to a simple perception: the fewer thoughts there are in one schema, the clearer and the sharper is the set of predicates. So, when a schema deals with many things in parallel, a lot of (self-contained) predicates are to be covered. This has an influence on the set of slices that are to be generated. When there is one 'crisp' thought specified in the schema, then the slice intersection covers all the predicates. When there are different thoughts specified in it, then the intersection is expected to get smaller (as each slice only regards dependent primes). A progress towards a single thought should therefore appear as a convergence between the size of the schema and the size of its slice intersection, an increasing divergence could indicate some deterioration of the formal specification.

Definition 9
Deterioration
Let Ψ be a formal Z specification, ψ_i one schema (out of n schemas) in Ψ, and $SP^{fb}_{\cap}(\Psi,\psi_i)$ its slice intersection. Then *Deterioration* $\delta(\Psi)$ expresses the difference between the average schema size and the average size of the slice intersections. It is defined as follows:

$$\delta(\Psi) = \frac{\sum_{i=1}^{n} \mid \psi_i \mid - \left| SP^{fb}_{\cap}(\Psi,\psi_i) \cap \psi_i \right|}{n} \qquad (\text{with } n = \#\{\psi_k : \text{Schema} | \psi_k \text{ in } \Psi\})$$

The value of deterioration, when standing alone, is not very informative – it is just an absolute number depending on the context. More information is yielded by looking at the differences in the

Table IV. Coupling and flow-related measures for a schema ψ_i in a Z specification Ψ (consisting of $n+1$ specification schemas).

Measure	*Definition*
Interschema flow: (Ψ,ψ_s,ψ_d)	$\frac{\left\lvert \left(SP^{b}_{\cup}(\Psi,\psi_d) \cap \psi_s\right) \right\rvert}{\lvert\psi_s\rvert}$
Interschema coupling: $C(\Psi,\psi_s,\psi_d)$	$\frac{F(\Psi\psi_s,\psi_d)\times\lvert\psi_s\rvert \;+\; (\Psi\psi_d,\psi_s)\times\lvert\psi_d\rvert}{\lvert\psi_s\rvert+\lvert\psi_d\rvert}$
Schema coupling: $\chi(\Psi,\psi_i)$	$\frac{\sum_{j=1}^{n} C(\Psi\psi_i,\psi_j) \times \lvert\psi_j\rvert}{\sum_{j=1}^{n} \lvert\psi_j\rvert}$ (for n schemas $\psi_j(\neq\psi_i)$ in Ψ)

DOI: 10.1002/smr

deterioration values between consecutive versions of the specification. For this case, the notion of a *Relative Deterioration* is defined.

Definition 10
Relative deterioration
Let Ψ_{n-1} and Ψ_n be two consecutive versions of a formal Z specification Ψ. Then, the relative deterioration ($\rho(\Psi_{n-1},\Psi_n)$) (with n being the sequence number of specifications, $n > 1$) is calculated as the relative difference between the deterioration of Ψ_{n-1} and Ψ_n. It holds:

$$\rho(\Psi_n) = 1 - \frac{\delta(\Psi_n)}{\delta(\Psi_{n-1})}$$

The value of relative deterioration is greater than zero when there is a convergence between schema size and slice intersection, and it is negative, when the differences between the sizes become bigger, indicating some probably unintentional deterioration.

The usability of the measure of deterioration might be debated, but it is only an indicator for the isolation of concerns in schemas. From the perspective of a developer, putting several concerns together might lead to an increase in the efficiency of the implementation. However, from the perspective of stakeholders that have to understand (and to agree upon), dozens of pages of specification text (so from the comprehension perspective), the separation of concerns has a lot of merits. In his keynote at FM'2012, Alan Wassyng ([31], p.8) also addressed this issue when he pledged for 'more prescriptive' methods, stating that our efforts as software engineers have to lead to development environments that can 'really be used' and understood by practitioners. In that sense, deterioration is addressing exactly the type of *quality* that should not be forgotten.

4. THE STUDY

After introducing the experimental basis and subjects in Sections 4.1 and 4.2, the study is split into three parts. In the first part (Section 4.4), a set of specifications is taken as a basis to assess the usefulness of the measures. The objective is to find out whether each of the measures represents unique characteristics of the specification or not. In the second part (Section 4.5), a set of specifications, revisions of the Web Service Definition Language Specification (WSDL) 2.0, is assessed in a longitudinal study. The objective of this step is to find out how well the measures are suited for predicting deterioration effects. As a concurrent versioning system (CVS) has been used for its further development from version 1.0, the 139 versions leading to WSDL 2.0 are a great candidate for examining the behavior of the measures. In the third part (Section 4.6), the results of the measures are examined again. The objective is to find out whether a baseline of measures can be found, and, if so, how a 'typical' metrics distribution looks like.

4.1. Experimental subjects

Large and publicly available specifications are rare. However, for the assessment of the measures a representative set of specifications is necessary. For this study several books, papers and projects have been searched and the resulting set of specifications has then been extended by 'real-world' specifications such that of the aforementioned WSDL. Table V provides a list of all specifications analyzed so far. The BBook [19], *Elevator* [18], and *Access Control System* [32] specifications are classical paper and textbook examples. The *Petrol station*, *Cinema*, *Library* system, *BankAccount*, *Cycalize*, *VendingMachine*, and *ParcelService* specifications have been built by students during labs at Klagenfurt University. The rest of the specifications are: sample specifications from the test case framework called 'Fastest' [33] (On-Board Application Software System, *Seguridad*, *EngineCache Scheduler*, *Teleservice*, the EXP-OBDH Communication Protocol, *Telecommand*, *FlowxSystem*, and the ECSS-E-70-41A Standard), the *ICT Window Manager* [34], which is part of the scheduler for

DOI: 10.1002/smr

Table V. Sample specifications used in this study. The table summarizes the type (T .. Textbook, S .. Students solution, R .. 'Real-world' specification), scale (*Lines* .. total lines, *LOC*(Ψ) .. lines of specification text, *A4* .. size in A4 pages), the number of schemas (experimental subjects *Subj.*), its conceptual complexity (CC), the number of predicates (*Preds*) forming the abstraction criteria ,and the total number of declarations (*Decl*).

Spec. Ψ	*Type*	*Lines*	*LOC*(Ψ)	*A4*	*Subj.*	*CC*	*Preds*	*Decl*
Library	S	43	37	2	2	33	13	11
Birthday Book	T	64	40	2	6	34	8	6
Petrol	S	130	88	3	10	65	34	12
Access Control	T	60	45	2	6	41	12	16
Cinema	S	175	95	4	9	74	26	15
OBS	R	377	106	7	13	104	108	12
Cycalize	S	245	118	7	10	96	30	22
Seguridad	R	278	130	4	18	124	56	29
VendingMachine	T	264	169	7	25	146	60	28
BankAccount	S	199	173	5	15	181	153	40
SchedulerJW	R	291	186	6	21	166	76	33
ParcelService	S	334	191	7	9	136	24	47
Elevator	T	241	193	6	18	185	264	9
SteamBoiler	T	331	252	9	36	342	181	36
EngineCache	R	384	259	8	34	225	86	63
CISC	R	746	343	12	67	368	120	60
ICLDataDic	R	945	352	20	30	278	173	60
ICT WM	R	539	359	12	41	280	154	50
OSScheduler	T	423	365	12	47	455	204	34
RoomPlanner	T	643	381	13	19	254	111	47
Scheduler	R	650	394	12	35	330	218	40
Caviar	R	1500	456	27	92	467	189	49
Unix	R	1199	474	29	58	362	112	79
EOCP	R	914	488	11	53	538	322	52
Hysteresis	R	6594	604	93	19	303	69	58
Teleservice	R	954	679	12	40	686	338	193
CVSServer	T	1515	766	34	37	425	135	76
Telecommand	R	1489	963	22	78	1073	491	149
Radiation Therapy	R	2482	1172	36	131	1038	704	120
WSDL 2.0	R	16,649	814	33	98	814	3784	210
Mondex	R	4857	1371	102	132	868	1535	116
Corba	R	3370	1616	81	130	1268	425	286
FlowxSystem	R	6128	1941	94	284	2129	2148	286
Tokeneer	R	6742	2013	117	170	1575	2141	96
ECSE7041	R	3293	2063	73	134	1461	823	287
Sum	—	**65,190**	**20,193**	924	**1938**	**17,040**	**15,633**	—
WSDL (47 CVS vers.)	R	571,559	37,348	—	2978	3076	85,941	—
Total	—	**636,749**	**57,541**	—	**4916**	**20,116**	**101,574**	—

the Linux 2.0 kernel, the *Radiation Therapy Machine* [35], the *Mondex* electronic purse [36], the *Caviar System* ([37], pp.71–98), the *Hysteresis* specification [38], the *ICL Data Dictionary* ([37], pp.99–119), the *OS Scheduler* ([4], pp.330-344), the *CISC TemporaryStorage* ([37], pp.203–218), and the *Unix File System* ([37], pp.41–70). The *Steamboiler* (([39], pp.109–128)), *RoomPlanner*, *CVSServer*, and *Corba* specifications have been taken from the example set of the Isabelle/HOL-Z project version 3.0 (beta) [40], and the *Scheduler* specification has been taken from the test-bench of the CZT project [41]. The *Tokoneer* specification was created by Praxis and *SPRE* as a demonstration object for the NSA [42].

For conducting this study, a Java-Z framework called ViZ [20] was used. ViZ supports concept location within formal Z specifications. It takes a formal Z specification as input by making use of the CZT parser for Z specifications [43], transforms it to an ASRN, and then calculates the necessary slices. For this study, it has been equipped with batch-functionality to access the 139 versions of the WSDL specification and to export the measures into a comma-separated file for further statistical processing. The only manual intervention necessary was adapting a few lines in every WSDL specifications to match the actual Z standard as implemented in the CZT plug-in.

DOI: 10.1002/smr

The WSDL specification is available in 139 versions in the CVS, but most of the revisions vary in the amount of comments and natural text describing the specification only. In the specification code itself, 47 of them differ, and therefore, they become perfect candidates for this study. The 47 versions result in 37,348 lines of specification text and are the basis for 85,941 slices. In total, 4916 Z schemas and 57,541 lines of specification text have been analyzed, summing up to more than 101,000 predicates; slices have been calculated for.

4.2. Statistics

Within the scope of this contribution, three different statistical tests were used to assess the data: the Pearson's Correlation Coefficient, the Spearman's Rank Correlation Coefficient, and the Kendall's Tau Correlation Coefficient.

The Pearson's correlation coefficient (R_P) measures the degree of association between the variables, assuming normal distribution of the values ([44], p. 212). Although this test might not necessarily fail when the data is not normally distributed, the Pearson's test only looks for a linear correlation. It might indicate no correlation even if the data is correlated in a nonlinear manner.

As past experiences about the distribution of the values are missing, one has to assume the data being not normally distributed. To handle this case, the Spearman's rank correlation coefficient (R_S) has been chosen ([44], p. 219). It is a nonparametric test of correlation and assesses how well a monotonic function describes the association between the variables. This is performed by ranking the sample data separately for each variable.

Kendall's robust correlation coefficient (R_K) can be used as an alternative to the Spearman's test ([45], p. 200). It is also nonparametric and investigates the relationship among pairs of data. However, it ranks the data relatively and is able to identify partial correlations.

When there is no likelihood of confusion, then R will be used to refer to either R_P, R_S, or R_K.

In the remainder of this work, the correlation R is interpreted as follows:

- When $|R| \in [0.8,\ 1.0]$, it is interpreted to indicate a *strong association*.
- When $|R| \in [0.5,\ 0.8)$, it is interpreted to indicate a *moderate association*.
- When $|R| \in [0.0,\ 0.5)$, it is interpreted to indicate a *weak association*.

In addition to the values of the correlation R, also the significance level (p) of the value is provided (checking, within the scope of the null hypothesis, that the probability of the value of R is bigger or equal to the observed value of R). The values in the following tables are rounded to the third decimal place (which means that a value of $p = 0.0005$ would become $p = 0.001$).

4.3. Charts and tables

Apart from statistical tests, this contribution also provides a series of plots and tables. The plots help in gaining a better feeling in the distribution of and relations between the measures. As the basic idea is to stay comparable with the study done by Meyers and Binkley [12], scatter plots ([45], p.199) are used to amend the statistical overviews. They depict the relationship between two variables, in our case, the measures introduced in Sections 3.2 and 3.3. Every scatter plot contains n data points (representing the n Z schemas ψ_i, $i = 1n$).

The tables, on the other hand, summarize the results of the statistical tests for all n variables. They also include the p-values for testing the hypothesis of no correlation against the alternative that there is a nonzero correlation. In addition to the results of the tests, a classification of the statistical significance (*Sig.*) is provided on the basis of the Pearson correlation values R_P.

4.4. Comparison of metrics

A first step is to take a closer look at the evaluation of the measures with respect to their uniqueness. This includes the following issues to be checked:

- The relationship between size and structure-based measures (*CC*, *Logical Complexity*, and DU *count*) and the measure of *Lines of Specification Text*,

DOI: 10.1002/smr

- The relationship between slice-based measures (*Tightness*, *Minimum* and *Maximum Cov*, *Cov*, *O*, and *Coupling*) and one size-based measure (*CC*), and finally,
- The correlations between different slice-based measures (*Tightness*, *Minimum* and *Maximum Cov*, *Cov*, *O*, and *Coupling*).

4.4.1. Issue I1. The first issue is to look at the relationship between the size and structure-based measures and the measure of lines of specification text. As the specification is mapped to the ASRN one-to-one (and the measures are all calculated on the basis of the graph), it might be possible that especially the previously defined complexity measures are just proxies for the number of lines in the text. The ASRN contains a vertex for every prime, and theoretically, between each pair of vertices representing predicates there might be a control and/or data dependency.

Figure 3 presents a scatter plot that compares the lines of specification text on the *X*-axis (per Z schema ψ_i) and related size-based and structure-based measures. As might have been expected, there is a clear trend for the size-based measure (CC, upper left), indicating a high correlation. But the plot is not so clear for structure-based measures. The bigger the specifications are, the more dependencies will usually exist, but the plots in Figure 3 demonstrate that the plot broadens for both complexity measures – they are not strongly related to the size of the specification.

Table VI summarizes the results of the correlation tests. As expected, CC is strongly related to the number of lines of specification text ($R = 0.933$ for the Pearson test). However, the two other measures (logical complexity and Def/Use count) do have moderate correlations only.

As all three tests yield a (weak to) moderate relation, the structural measures are very likely not just proxies for the number of specification lines.

The correlation tests (and the scatter plot) indicate that CC does not provide new insights when compared with the lines of specification text ($LOC(\Psi)$). Nevertheless, the use of CC in the remainder of this work can be justified by the following three arguments:

- The number of lines of specification text depends on the style used when writing down the predicates. By looking at the differences between the values of CC and lines of specification text in Table V, one can see that the values differ up to 25% – which is quite a lot.

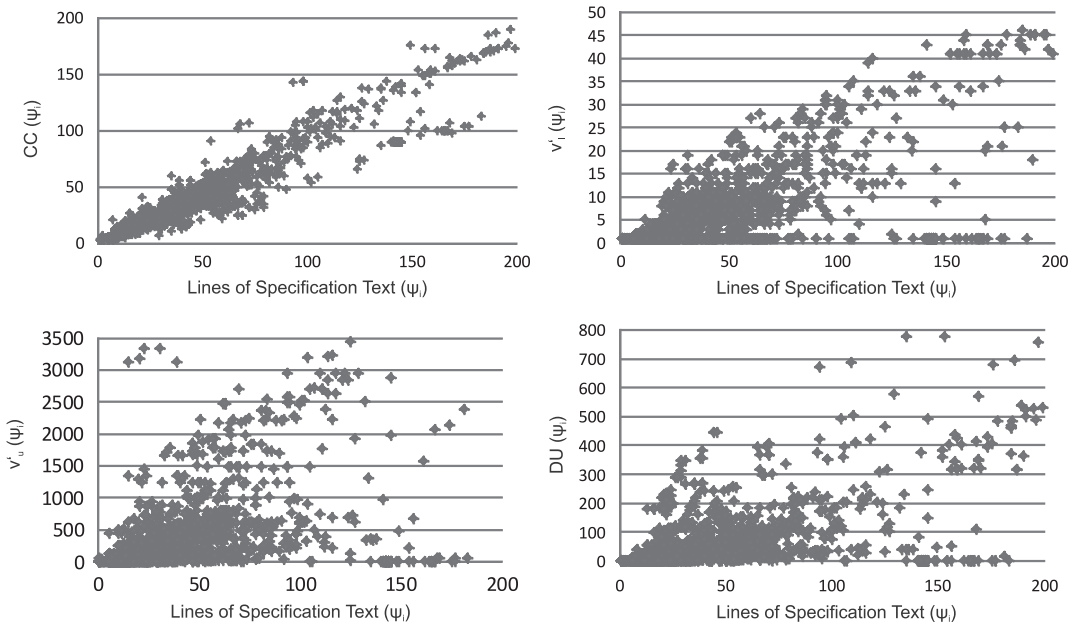

Figure 3. Metrics comparison of size-based and structure-based measures (*Conceptual complexity*, *Logical complexity*, and *Definition-use count*) against the measure of *Lines of Specification Text* ($n = 1938$).

DOI: 10.1002/smr

Table VI. Pearson's, Spearman's, and Kendall's correlation (R_P, R_S, R_K including p-values for testing the hypothesis of no correlation against the alternative that there is a nonzero correlation) for the metrics comparison displayed as scatter plot in Figure 3. The classification of the statistical significance (*Sig.*) is based on the Pearson correlation values R_P.

Lines of specification text correlation ($n = 1938$)							
		Pearson		Spearman		Kendall	
Sig.	*Measure*	R_P	p	R_S	p	R_K	p
Strong	$CC(\Psi)$	0.933	.000	0.964	.000	0.858	.000
Moderate	$\nu'_l(\Psi)$	0.656	.000	0.563	.000	0.470	.000
(0.5, 0.8)	$\nu'_u(\Psi)$	0.668	.000	0.580	.000	0.458	.000
	$DU(\Psi)$	0.659	.000	0.557	.000	0.435	.000

- Conceptual complexity represents the amount of declarations and predicates in a specification (which are very likely refined later on to program code). So, CC might be slightly more precise as basis for implementation-related predictions.
- The calculation of the other semantics-based values goes back to counting primes. When there are correlations, then it is very likely that the effect will be more pronounced when looking at primes and not at the number of lines of specification text.

4.4.2. Issue I2. The second issue is to take a closer look at the correlation between the size of the specification (now represented by $CC(\psi_i)$) and the values of cohesion and coupling. Figure 4 presents a

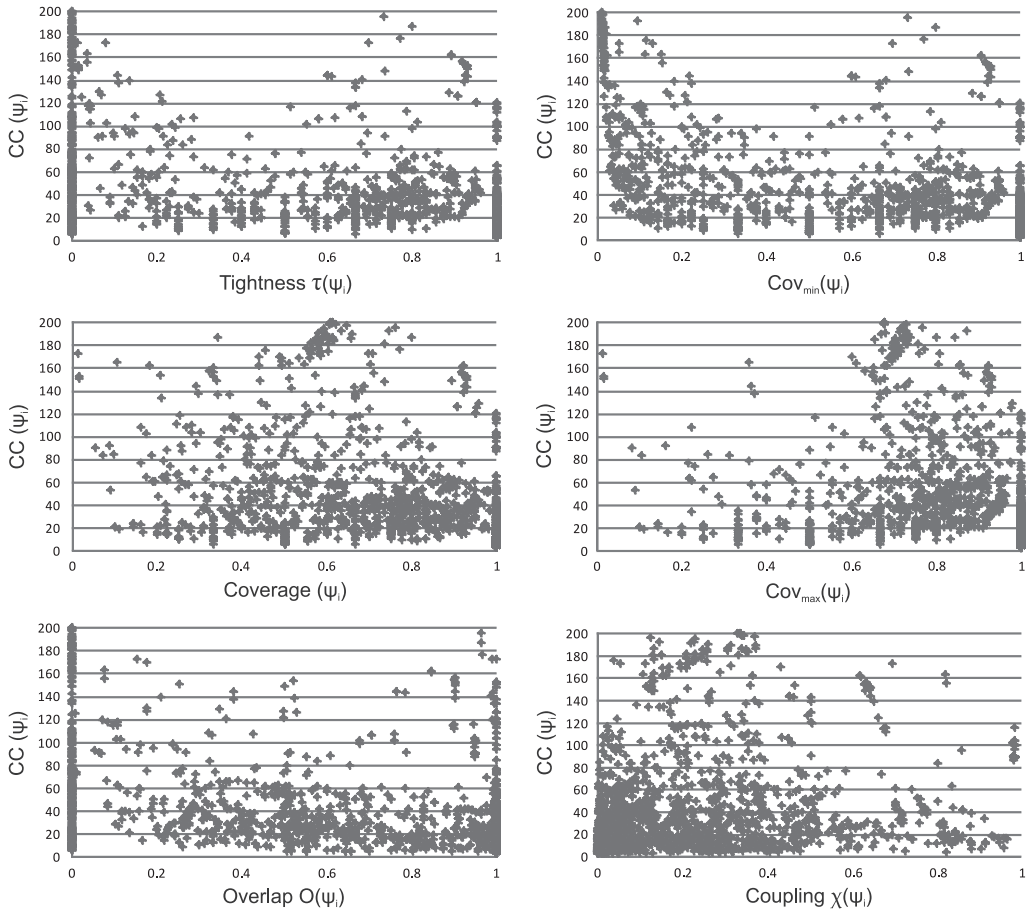

Figure 4. Metrics comparison between size-based and slice-based measures ($n = 1938$).

DOI: 10.1002/smr

scatter plot for the six different measures. None of them indicates a direct or strong correlation, but, the plots do reveal other interesting properties.

- It is apparent that the measures for coupling and cohesion have their bulks in different areas of the diagrams. The values for cohesion are in most cases between 0.5 and 1.0, whereas the values for *Coupling* are between 0.0and 0.5. Figure 5 illustrates this for the values of *Coupling* and *Cov*. As explained later, there is some weak to moderate relation between them. It is good to see that also for formal specifications there seems to be some trend towards values of high cohesion and low coupling.
- Four of the five values of cohesion (τ, Cov_{min}, Cov_{max}, O) do have concentrations of the values at 0 and 1. The reasons go back to the density and the types of the specifications. There are specifications that contain unrelated state spaces without predicates. The values for cohesion are then equal to 0. And some of the specifications are very dense (with relations between all its primes). Every prime is then element of the intersection slices. The values for cohesion are thus equal to 1. Basically, these situations happen with textbook examples (which try to explain the syntax of Z in a condensed manner), but also in the *Caviar* and the *Unix*-system specification. In these specifications, up to 24% of at least one of the values are equal to 0 or 1. However, in 40% of these cases, not all of the values of *Coupling* do have the same value then, and an assessment of these schemas is still possible.

Apart from these two observations, the plots between 0 and 1 do not reveal a clear correlation. Table VII summarizes the results of the Pearson, Spearman, and Kendall tests. The tests show that with raising sizes of the specifications, the values for cohesion are decreasing, whereas the value for *Coupling* increases. This is not surprising, as with larger specifications, the chance for relationships between *all* of the schemas in a specification decreases. The correlation values of the Spearman test are slightly higher than the values of the other two tests, indicating a tendency to a nonlinear correlation. However, the results are still in the weak association class and thus the measures are treated as basically unrelated in the remainder of this work.

4.4.3. Issue I3. The last issue in evaluating the expressiveness of the values is to look at the interrelations between and within structure-based and slice-based measures. Figure 6 presents some scatter plots for the values. The results of the correlation tests are summarized in Tables VIII (for the slice-based measures) and 9 (for structure-based and semantics-based measures).

The correlation values for the slice-based measures (Table VIII) differ widely. Especially the value of *Tightness* is highly correlated to the other measures, and according to the test results, it is very likely that *Tightness* is a proxy for some of the other cohesion-related measures. In fact, the definitions of *Tightness*, *Cov*, and *O* are quite similar. Not surprisingly, this relation can also be seen in Figure 6. The least correlated measures are *Coupling* and *O*, and they thus are excellent candidates for the measures we need in the remainder of this work.

All *p*-scores are less than or around 0.001 – with two exceptions: firstly, at the correlation between *Coupling* and *Tightness* where p is higher than $0,05$ (0.477 for the Spearman, and 0.412 for the

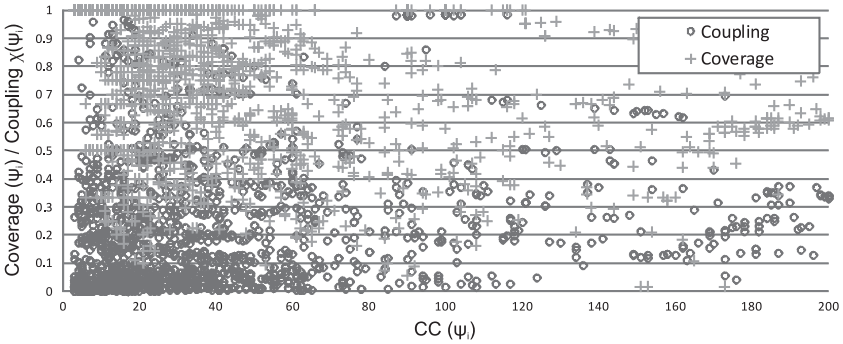

Figure 5. Distribution of two slice-based measures (*Coupling* and *Coverage*) for all samples under test ($n = 1938$).

DOI: 10.1002/smr

Table VII. Pearson's, Spearman's, and Kendall's correlation between CC and all slice-based measures displayed as scatter plot in Figure 4.

Conceptual complexity correlations ($n = 1938$)							
		Pearson		Spearman		Kendall	
Sig.	*Measure*	R_P	p	R_S	p	R_K	p
Weak	*Tightness*	−.296	.000	−.436	.000	−.330	.000
(0.0, 0.5)	$Cov_{\min}$	−.360	.000	−.498	.000	−.378	.000
	Coverage	−.221	.000	−.387	.000	−.281	.000
	$Cov_{\max}$	−.103	.000	−.308	.000	−.226	.000
	Overlap	−.314	.000	−.445	.000	−.331	.000
	Coupling	0.234	.000	0.311	.000	0.211	.000

Kendall tests), and secondly, at the correlation between *Coupling* and $Cov_{\min}$ (where p is 0.29 for the Spearman test and 0.298 for the Kendall test). This indicates that the results for these two tests are not statistically significant. A closer look at the scatter plot in Figure 7 does not explain the reason(s) for this situation. The plot seems to confirm the result of the Pearson correlation test, which tell us that there is, if at all, just a slightly positive relation.

But, a closer look into the core data reveals the matter: in Z specifications, operation schemas are usually combined into form bigger operations, finally (hierarchically) covering all the operations on the states. These operations (about 10% of all the schemas) do not add any new predicates or declarations, but just include all other existing operations and states. They are really big and (because of the introduced redundancy) distort the calculation of the correlation values. Omitting them would basically

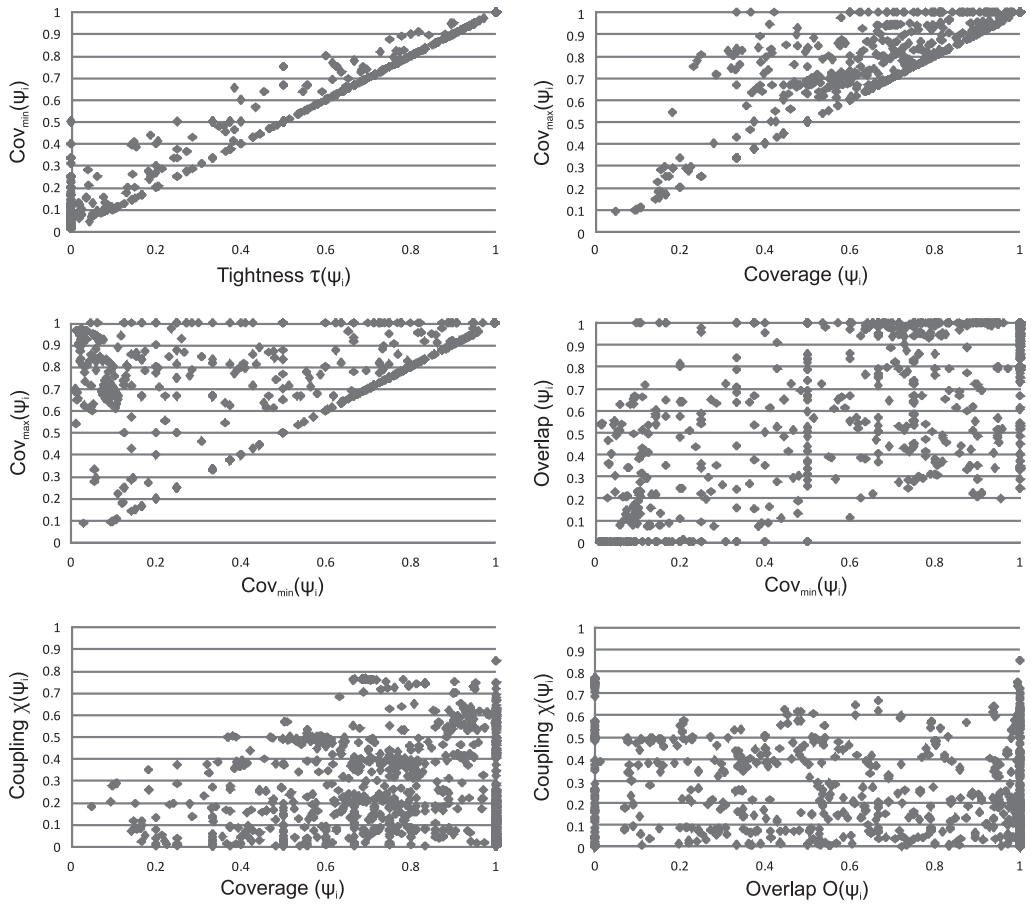

Figure 6. Metrics comparison between slice-based and structure-based measures ($n = 1938$).

DOI: 10.1002/smr

Table VIII. Pearson, Spearman, and Kendall for the correlation of slice-based measures.

Slice-based measures correlations ($n = 1938$)								
Sig.			Pearson		Spearman		Kendall	
	Measure 1	*Measure 2*	R_P	p	R_S	p	R_K	p
Strong	*Tightness*	$Cov_{\min}$	0.975	.000	0.981	.000	0.924	.000
[0.8, 1.0]	*Tightness*	*Coverage*	0.891	.000	0.947	.000	0.839	.000
	$Cov_{\min}$	*Coverage*	0.881	.000	0.937	.000	0.812	.000
	Coverage	$Cov_{\max}$	0.831	.000	0.852	.000	0.738	.000
	Tightness	*Overlap*	0.808	.000	0.726	.000	0.596	.000
Moderate	$Cov_{\min}$	*Overlap*	0.779	.000	0.700	.000	0.554	.000
(0.5, 0.8)	*Tightness*	$Cov_{\max}$	0.648	.000	0.760	.000	0.596	.000
	Coverage	*Overlap*	0.638	.000	0.607	.000	0.476	.000
	$Cov_{\min}$	$Cov_{\max}$	0.624	.000	0.754	.000	0.613	.000
Weak	$Cov_{\max}$	*Overlap*	0.358	.000	0.368	.000	0.287	.000
(0.0, 0.5)	*Coupling*	*Coverage*	0.208	.000	0.080	.000	0.064	.000
	Coupling	$Cov_{\max}$	0.203	.000	0.084	.000	0.065	.000
	Coupling	*Tightness*	0.125	.000	0.016	**.477**	0.013	**.412**
	Coupling	$Cov_{\min}$	0.101	.000	−.024	**.290**	−.017	**.298**
	Coupling	*Overlap*	0.077	.001	−.094	.000	−.061	.000

lead to the same correlation values, but with p-scores below 0.005. During the design of the experiments, it was decided not to prune the set of specification schemas. As combined operation schemas are typical for Z specifications, such an omission of schemas would have falsified the set of experimental subjects.

The situation is clearer for the combinations of structure-based and slice-based measures. Table IX shows that there is only a weak association between structure-based and slice-based measures. Table X reveals that there is a strong correlation between the set of structure-based measures. However, concerning these tables, two things are particularly noticeable.

At first, all p-scores in both tables are around 0.0 – with one exception: the p-score for the correlation test between $v'_l(\psi)$ and *Coupling* in Table IX is 0.337 for the Spearman test and 0.146 for the Kendall test. Secondly, the Pearson, the Spearman, and the Kendall correlation values for the structure-based measures diverge more than in the tests before. Again, a closer look at the data reveals the reason: 41 (2.11%) of all schemas are enormously big and do contain 10,000 and more control dependencies, whereas 75% of the schemas contain less than 500 control dependencies. The rest of the schemas have between 500 and 4000 control dependencies. Figure 8 visualizes the difference by making use of different intervals: at the left there is the scatter plot and the trend line for schemas of 'average size' ($n = 1650$), and to the right, there is the scatter plot for the values of $v'_l \leq 14000$ ($n = 1932$). As the Spearman test is based on the rank order of the values, it is not sensitive to the difference of an order of magnitude – which also explains the difference in the R values. The correlation values for the pruned set of schemas would all be between 0.648 and 0.791, so indicating a moderate correlation only.

For describing the complexity of a specification's structure, it could be sufficient to take just a look at either one value of logical complexity or the Def/Use count. However, for schemas of 'normal' size,

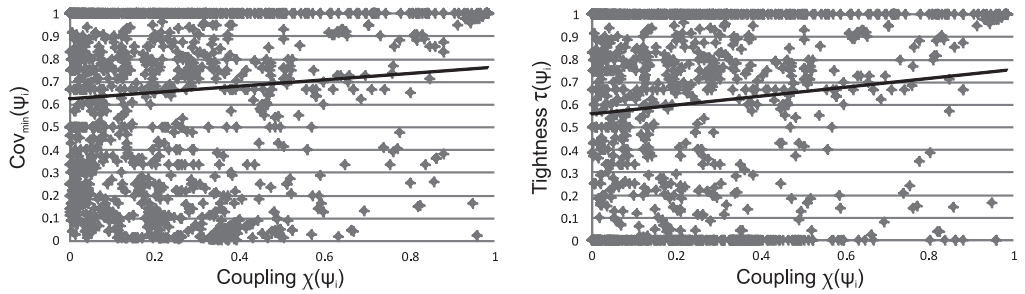

Figure 7. Metrics comparison between slice-based measures against the value of *Coupling* including their trend lines ($n = 1938$).

DOI: 10.1002/smr

Table IX. Pearson, Spearman, and Kendall for the correlation of slice-based and size-based measures.

Size-based and slice-based measures correlations ($n = 1938$)								
Sig.	*Measure 1*	*Measure 2*	Pearson		Spearman		Kendall	
			R_P	p	R_S	p	R_K	p
Weak (0.0, 0.5)	$\nu'_l(\psi)$	*Tightness*	−.341	.000	−.301	.000	−.241	.000
	$\nu'_l(\psi)$	$Cov_{\min}$	−.414	.000	−.356	.000	−.279	.000
	$\nu'_l(\psi)$	*Coverage*	−.257	.000	−.287	.000	−.221	.000
	$\nu'_l(\psi)$	$Cov_{\max}$	−.155	.000	−.239	.000	−.193	.000
	$\nu'_l(\psi)$	*Overlap*	−.368	.000	−.204	.000	−.235	.000
	$\nu'_l(\psi)$	*Coupling*	0.022	**.337**	0.033	**.146**	0.051	.002
	$\nu'_u(\psi)$	*Tightness*	−.362	.000	−.360	.000	−.275	.000
	$\nu'_u(\psi)$	$Cov_{\min}$	−.438	.000	−.430	.000	−.322	.000
	$\nu'_u(\psi)$	*Coverage*	−.265	.000	−.315	.000	−.236	.000
	$\nu'_u(\psi)$	$Cov_{\max}$	−.171	.000	−.275	.000	−.210	.000
	$\nu'_u(\psi)$	*Overlap*	−.361	.000	−.306	.000	−.231	.000
	$\nu'_u(\psi)$	*Coupling*	0.054	.017	0.307	.000	0.220	.000
	$DU(\psi)$	*Tightness*	−.343	.000	−.355	.000	−.270	.000
	$DU(\psi)$	$Cov_{\min}$	−.411	.000	−.421	.000	−.315	.000
	$DU(\psi)$	*Coverage*	−.252	.000	−.303	.000	−.226	.000
	$DU(\psi)$	$Cov_{\max}$	−.120	.000	−.225	.000	−.173	.000
	$DU(\psi)$	*Overlap*	−.375	.000	−.342	.000	−.258	.000
	$DU(\psi)$	*Coupling*	0.152	.000	0.409	.000	0.311	.000

all the values could be relevant as there is only moderate correlation between them. Additionally, the absolute values provide a deep insight into the specification (and a possibly refined implementation).

To summarize, when looking for a set of measures that represents unique characteristics of a specification (size, structure, semantics) then the outcome of this part of the study suggests to focus on the values of

- *Conceptual complexity* or *Lines of specification text* when talking about its size,
- *Logical complexity* ($\nu'(l,u)$) or *DU count* for its structural complexity,
- *Coupling* (χ), *O*, and eventually *Cov* for its semantic characteristics.

The measure of *Tightness* is skipped because of its ambiguity of its correlation results (p-scores) and because of its high correlation to the other cohesion measures. *Minimum* and *Maximum Cov* are skipped for the same reasons. On the other hand, the remaining measures (CC, $\nu'(l,u)$, DU, χ, *O*, and *Cov*) will be used in the next two sections of this paper.

4.5. Longitudinal study

The WSDL [46] is one of the rare, big publicly available, formal Z specifications. It specifies the WSDL Version 2.0 (WSDL 2.0), which is an XML language for describing Web services. The specification consists of a detailed natural language description and a Z specification of the core language that is used to specify Web services based on an abstract model of what the service offers. Additionally, it specifies the conformance criteria for documents in this language.

Table X. Pearson, Spearman, and Kendall for the correlation of structure-based measures.

Structure-based measures correlations ($n = 1938$)								
Sig.	*Measure 1*	*Measure 2*	Pearson		Spearman		Kendall	
			R_P	p	R_S	p	R_K	p
Strong (0.8, 1.0)	$\nu'_u(\psi)$	$DU(\psi)$	0.918	.000	0.813	.000	0.660	.000
	$\nu'_l(\psi)$	$\nu'_u(\psi)$	0.871	.000	0.648	.000	0.531	.000
	$\nu'_l(\psi)$	$DU(\psi)$	0.834	.000	0.644	.000	0.518	.000

DOI: 10.1002/smr

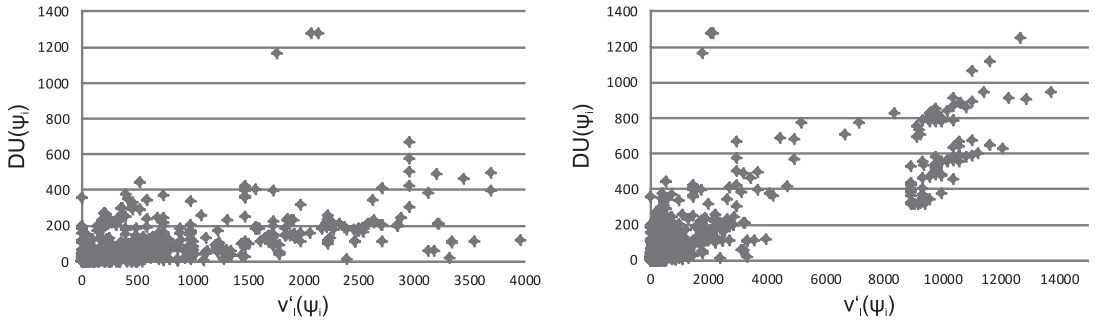

Figure 8. Scatter plots for the Def/Use count value and logical complexity v'_l. On the left, there is the plot for the values of $v'_l \leq 4000$ ($n = 1650$), and to the right, the full plot for the values of $v'_l \leq 14000$ ($n = 1932$).

The specification itself is of medium complexity. There are no predicates that are mathematically very complex. 78% of the expressions make use of the logical AND operator, and 17% make use of existential quantifiers. Only 1.5% make use of logical implications. The final revision contains 814 primes (distributed over 1413 lines of Z code, in 16,641 lines of text altogether). The advantage of this specification is that, starting with 2004 (and during the development of WSDL 2.0) a CVS has been used. WSDL 1.0 is not available in Z, but from November 2004 on 139 versions – due to change requests and other maintenance activities – have been checked in till its final release in 2007 [47]. The first revision is an adoption of WSDL 1.0, and then, successively, functionality has been added, modified, or deleted. Figure 9 presents an overview of the changes in the size of the specification (comments and specification text) for all 139 versions. As can be seen, the first version in the CVS (with revision number 1.1) contained 348 lines of specification text (790 lines of text in total). The specification has steadily been extended, finally yielding a specification (with revision number 1.139) consisting of 1.413 lines of specification text. The last revision in the CVS became the publicly available Z specification of WSDL 2.0. Please note that for reasons of space, the figures' revisions 1.1 to 1.139 are abbreviated to 1 to 139.

Before taking a closer look at the different measures of the WSDL specification, a look at the CVS log is very informative. It provides a first insight into the types of changes (corrective, adaptive, or perfective) that occurred on the way to the final release. Up to revision 1.092, the interventions (called 'last calls') mainly consist of adding and modifying concepts (schemas) to the specification (see Figure 10 for an example, where, at revision 1.038 the last call [LC06a] has been implemented). After revision 1.095, there are solely change requests. Although there have been several changes influencing the final product, the following sections and revisions attract attention and are considered later in more detail:

- *Up to Rev. 1.006*, there is a mapping of WSDL version 1.0 to Z. Only minor changes to the specification take place, but, starting with Rev. 1.007, editorial improvements take place. After the improvements, the model is extended (for example by introducing the two concepts of *Binding* and *Services*).
- Starting with *Rev. 1.020*, a recurring refactoring and extension of the model takes place. Several concepts are added, for example the notion of an InterfaceComponent or that of an *ID*.

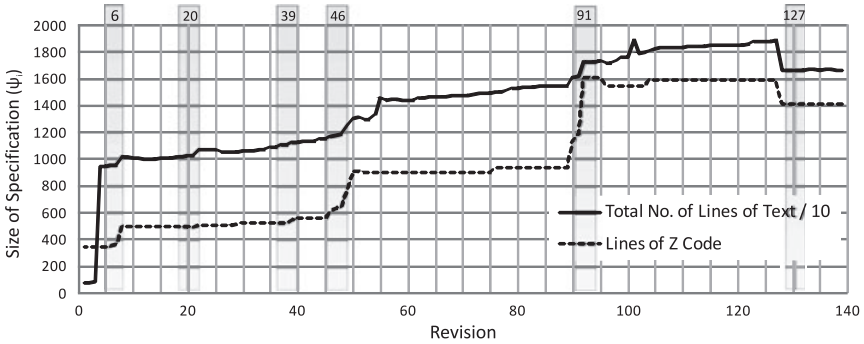

Figure 9. Number of lines of specification text and comment lines for all 139 revisions of *WSDL*2.0. The revisions that are discussed in more detail in the study are highlighted.

DOI: 10.1002/smr

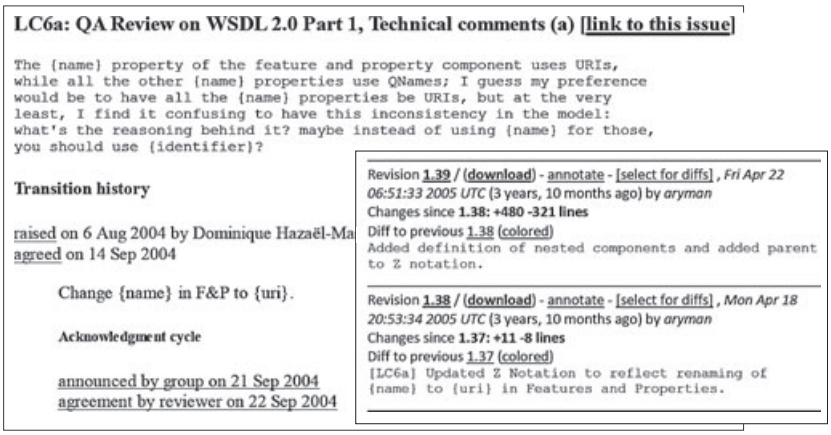

Figure 10. Example of a 'Last Call' description ([*LC6a*]) and two entries (Revs. 1.038 and 1.039) in the CVS log for the *WSDL*2.0 specification.

- Starting with *Rev. 1.038*, another simplification takes place. The concepts of *URIs* and *QNAME*s are merged.
- Beginning with *Rev. 1.045*, a lot of constraints are added to the specification. Several smaller changes are applied to the model (according to *Interface* and *Bindings*). Additionally, *Faults* are introduced as a stand-alone concept.
- At *Rev. 1.077*, editorial improvements to message label rules take place. In addition to that, long definitions are refactored in the *tables* part.
- Beginning with *Rev. 1.090*, the model is extended massively. New concepts, such as that of an *XMLnamespace* or *TypeDesignator*s are introduced.
- At *Rev. 1.096*, according to the CVS log, a simplification of the model takes place. In fact, *Component*s and their *Context* are aggregated into a *TopLevelComponent*; additionally, the concept of *IRI*s is updated.
- Beginning with *Rev. 1.127*, change requests lead to a structural refactoring of the specification. Several concepts, such as that of *Features* and *Properties*, are removed.

Massive changes took place at revisions 1.039ff and 1.090ff. It has to be noted that, after looking at the CVS logs, the changes have been verified by making use of a software tool for file-compare (Beyond Compare V3.2.4), which confirmed the entries in the log. By analyzing the different maintenance activities stored in the CVS log, it is noticeable that a lot of the last calls followed a clear strategy in adding the desired functionality. The steps are as follows:

1. Refactoring the actual version.
2. Adding/removing/modification of a concept.
3. Update of the natural language documentation.

This strategy can be derived from the CVS log directly. The next question addressed is now whether the measures that have been defined in Section 3 capture these changes and whether they can be used to assess the results of the different maintenance activities.

4.5.1. Deterioration. At first, let us look at the measure called deterioration δ. Figure 11 presents the value for δ for all 139 revisions in the CVS. This figure suggests that the specification deteriorates basically around revisions 1.006 and 1.091, and improves around revisions 1.039 and 1.127. In fact, the CVS log also documents the changes, as they mention the extensions to the existing model after revisions 1.006 and 1.090, and also the simplifications at revisions 1.038 and 1.127.

At all revisions, Z code has been added or removed, but the change in the total size of the specification is not the reason for deterioration. Figure 12 shows the values for the mean of the sizes of the Z schemas and the mean of the sizes of the corresponding slice intersections. Deterioration is the difference between the (average) size of a schema and the (average) size of the slice intersection (see Figure 12), thus by the drift between them we can see if and when the specification *semantically* deteriorates. It is noticeable that not

DOI: 10.1002/smr

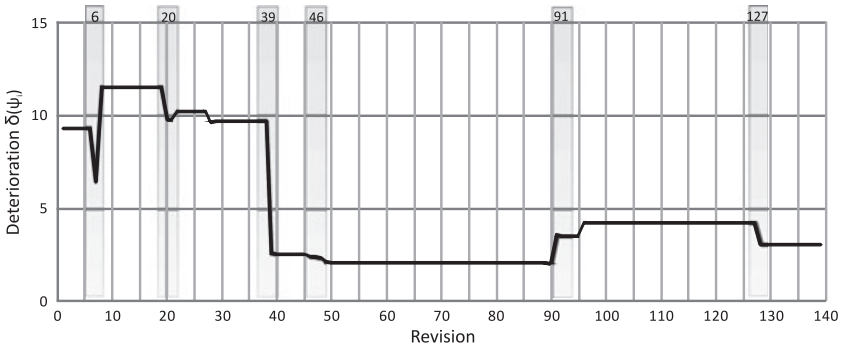

Figure 11. Values of deterioration for all 139 revisions of the WSDL specification.

all revisions with a change in size led to the same amount of change in the size of the slice intersections. For example, at revision 1.039, there is growth in size and a bigger growth in $SP^{b}_{\cap}$. This indicates that currently available concepts have been changed; existing trains-of-thought have been merged. On the contrary, at revision 1.091, there is an increase in size and a lower increase in $SP^{fb}_{\cap}$. This indicates that such primes have been added to the specification schemas that introduce new thoughts. The CVS log indeed confirms this observation: at revision 1.091, the new concepts of *XMLNamespaces* and *TypeDesignator* have been introduced, whereas at revision 1.039, existing schemas have been modified by simplifying the access to the *names* concept.

4.5.2. Relative deterioration. The absolute values in Figure 11 make it difficult to estimate how big the influence of a change really is. The same holds for Figure 12, where it is hard to say whether the difference between the sizes of the schemas and their slice intersections change drastically or not. For this reason, the notion of relative deterioration (ρ) has been introduced in Section 3.4. Figure 13 presents the value of ρ for all 139 revisions. The value is defined in such a way that a positive amplitude indicates an alignment of the schema sizes and their slice intersections (which is assumed to be positive with respect to deterioration), whereas negative values indicate negative effects.

From Figure 13, we can now infer that the biggest negative changes happened between revisions 1.007 and 1.008 and between revisions 1.090 and 1.091. And besides the improvement at revision 1.039, the final refactoring activities around revision 1.127 had also a very positive effect with respect to the value of deterioration.

Another interesting observation can be made: when taking a closer look at Figure 13, the process of implementing last calls gets visible. At revisions 1.006, 1.020, 1.027, and 1.038 it can be deduced from the graph. A change is triggered, first, by a structural improvement (to be seen as a positive amplitude of ρ in the diagram), followed by the introduction of the new thought, which in some of the cases resulted in a negative amplitude of ρ.

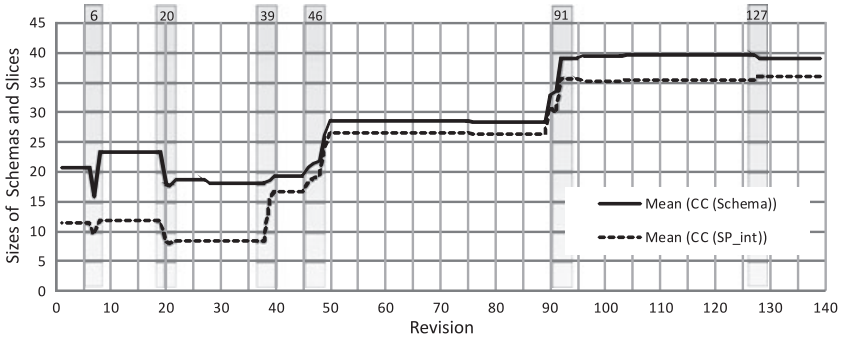

Figure 12. Change in the sizes of schemas and their corresponding intersection slices $SP^{fb}_{\cap}$ for all 139 revisions of the WSDL specification.

DOI: 10.1002/smr

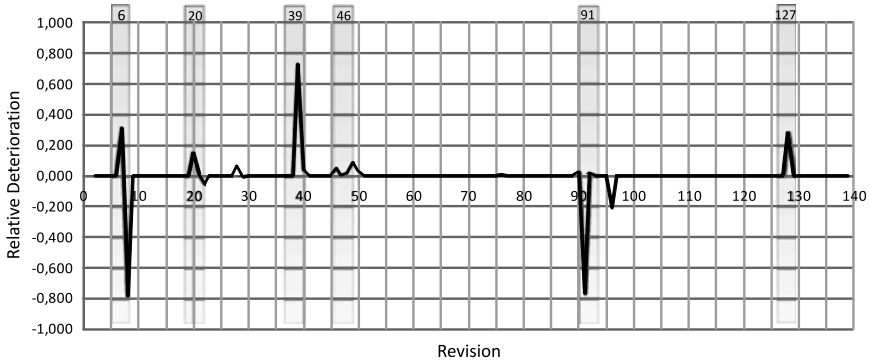

Figure 13. Relative change of the size of deterioration for all 139 revisions of the WSDL specification.

4.5.3. Coupling and cohesion. Up to now, we have only looked at the differences between slice intersections and schema sizes. This section deals with the question of how the changes affect our previously defined quality measures.

Looking at Figure 14, we see that the values for *O* are low at the beginning (around 0.39). This indicates (and can also be seen in the specification) that a lot of concepts or 'thoughts' within the specification schemas are unrelated. Even more, the number of primes common to all other slices became lower by adding new concepts into the specification schemas. The drop in the value of *O* can be seen in Figure 14 between revisions 1.006 and 1.038. Only with the combination of the concepts of *names* at revision 1.038. this trend stopped and *O* increased again (and reaches the value of 0.86 at the end).

The value of *Cov* follows the fluctuation of the other measures – but not at all revisions to the same extent. On the long run, it increases, ending up at a slightly higher value than at the first revision. *Cov* tells us about how specific a schema is (which means how many primes in its body are responsible for a single thought). The developers started at a high level but they managed to improve this property till the final release. It is interesting that the extension at revision 1.091 did not lead to a drastic decline of the measure. The reason for that is the fact that, with the introduction of new concepts. a lot of crisp schemas were added to the existing specification. The changes on the existing schemas were kept marginal, resulting in only minor fluctuations of the value on average.

Coming from intraschema measures to interschema properties, *Coupling* refers to the flow between different schemas in the specification. The lower the values for the flow are, the better it is. In case of WSDL, this flow is quite high. Figure 14 shows that the value fluctuates with the values of *Cov* and *O*, but again not necessarily to the same extent, and not inversely (as would be assumed to be ideal for programs). Beginning with version 1.006, the model has been changed in such a way that schemas started to share concepts, yielding an increase in *Coupling*. Then, at revision 1.020, new, self-contained schemas were added, and the value decreased again. After that, at revision 1.046, a new schema with connections to existing schemas was introduced. The result is an increase in the value of *Coupling*. Interesting is also the situation around revision 1.090. There, massive changes to

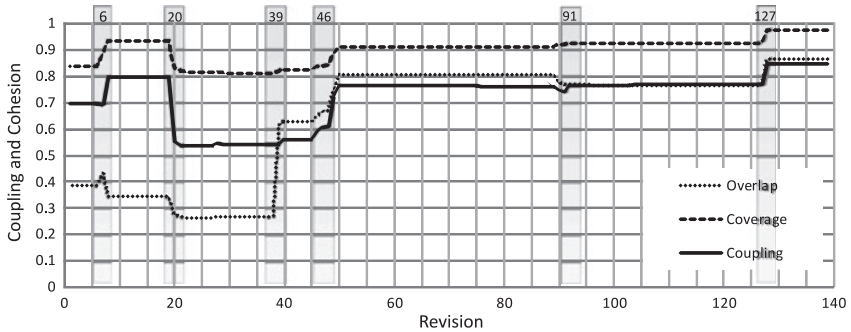

Figure 14. Values of *Coupling*, *Overlap*, and *Coverage* for all 139 revisions of the WSDL specification.

DOI: 10.1002/smr

the specification happened, but they barely led to a change of this quality measure. In fact, the freshly added schemas are as highly interconnected with the other schemas, as the schemas that existed before. Finally, at revision 1.127, the specification has been refactored a little. Some concepts have been removed; others were integrated into single schemas. *Coupling* increases as the number of relations between the concepts has been increased a slightly.

To summarize, the value of *Coupling* and the different cohesion values 'visualize' a lot of changes documented in the CVS log. But they are not necessarily able to evaluate every change. One reason for this is that the measures are not sensitive enough when only minor changes take place (and the change is then only visible at the third position after the decimal point). Another reason is that in some situations the different maintenance activities might cancel each other.

4.5.4. Influence of size of change. As can be derived from the previous deterioration measures δ and ρ, adding or deleting primes influences the quality measures. A remaining question is whether the type and the size of the change correlates with the type and the size of change of the measures. For the following considerations, a relative change in size of the specification is defined and then compared to our quality-based measures.

Definition 11
Relative deviation of CC
Let Ψ_{n-1} and Ψ_n be two consecutive versions of a formal Z specification Ψ. Further, let min(Ψ) and max(Ψ) be the sizes of the smallest resp. largest versions over all versions of Ψ. Then, the relative deviation in size ($\Delta(\Psi_{n-1},\Psi_n)$) with $n > 1$ is calculated as 1 minus the ratio between the change in size of Ψ_{n-1} to Ψ_n and the maximal difference in size. It is defined as follows:

$$\Delta(\Psi_{n-1},,\Psi_n) = \begin{cases} 1 + \dfrac{|\Psi_n| - |\Psi_{n-1}|}{\max(\Psi) - \min(\Psi)} & \text{when } \max(\Psi) > \min(\Psi) \\ 1 & \text{otherwise} \end{cases}$$

The value of the relative deviation of CC is greater than 1 when primes are added to the specification, and it is less than 1 when primes are removed.

As can be seen in Figures 15 and 16, a change in size does not always influence our quality measures to the same extent. From the number of additions/modifications/deletions, it is impossible to predict the effect on the measures. Up to revision 1.127, primes are just modified and added to the specification, but the values for *Cov*, *O*, and *Coupling* sometimes increase *and* sometimes decrease. Also, the amplitude of the changes is no indicator. Around revisions 1.006, 1.046, 1.091, and 1.127, considerable schema modifications took place; however, the biggest changes in *O*, *Cov*, and *Coupling* happened at revisions 1.020, 1.039, and 1.046. On the other hand, at revisions 1.090 and 1.127, the values did not change dramatically.

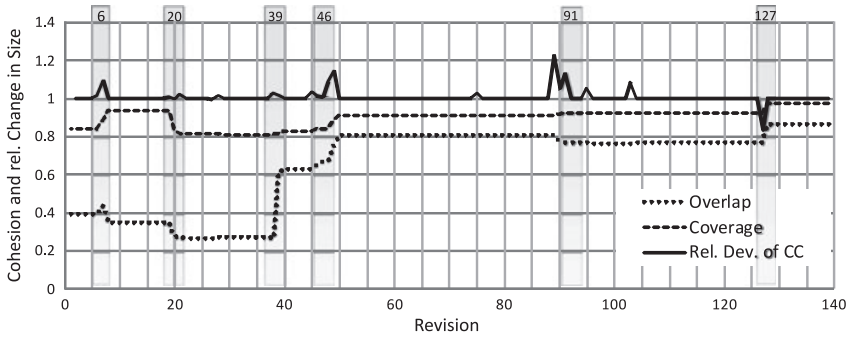

Figure 15. Influence in the change of number of primes on the values of cohesion.

DOI: 10.1002/smr

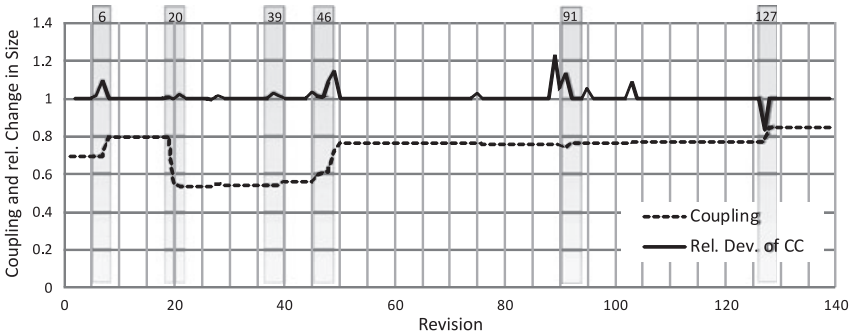

Figure 16. Influence in the change of number of primes on the value of coupling.

4.6. Baseline

After the head-to-head comparison of the measures and the longitudinal analysis, we may take a closer look at baseline metrics. The idea of a baseline is to support separating those schemas of a specification with divergent measures from those that are assumed to be 'normal'. Table XI summarizes the statistics for all 1938 schemas from our sample collection of specifications. The mean (average) of the values represents the values that can be expected for a 'normal' Z specification. The dispersion of the data is illustrated by the values of standard deviation. For example, *overlap* has a standard deviation of 0.396, which means that most of the values fall within a range of 0.792 around the mean value. It also indicates that the measure is quite sensitive as its range is quite broad. On the other hand, *Cov* has a standard deviation of 0.245, indicating a slightly less sensitive measure (as the range is smaller).

Table XI also provides the values for the 95% confidence interval for the measures. When we can assume that the schemas studied so far represent a sufficiently random sample of the population, then the values demonstrate that the means are quite good representatives. For example, it can be stated that, with a 95% confidence, the average value of *Cov* falls between 0.759 and 0.781. The other metrics show quite similarly small confidence intervals.

For the sake of completeness, the values for size and structural complexity are also provided in Table XI. It shows an overview of the distribution of the values, but the mean has to be taken with care as the values for standard deviation are very high. In fact, the specifications differ a lot concerning the number of control and data dependencies.

The baseline (and the measures) can now be used to assess the further development of Z schemas. The approach itself is scalable and can easily be integrated into the development process. All the environment requires as input is a syntactically correct specification and then it calculates and exports the measures into a CVS file for further examination. The time needed for the calculation depends on the number of dependencies in the ASRN that are to be identified and the number of slices that are to be considered. In our case, the sample specifications have been assessed on a standard laptop (2.67 GHz i7 CPU, 4GBRAM), and the time varied from less than a second (BBook specification with eight slices, $v'_l = 4$, $v'_u = 10$, $DU = 4$), up to 5 min (*FlowxSystem* specification with 2.148 slices, $v'_l = 214$, $v'_u = 45.303$, $DU = 3.476$).

Let us now clarify how the approach can be used. As a first example, let us take a look at the WSDL specification. At revision 1.005, the schema *Components* ($CC = 37$) has an overlap value of 0.443, which is below the mean, and a value of *Coupling* of 0.944. The schema handles all components

Table XI. Metrics distribution and 95% confidence interval for all 1938 schemas ψ_i.

Measure	$CC(\psi_i)$	$v'_l(\psi_i)$	$v'_u(\psi_i)$	$DU(\psi_i)$	$Cov(\psi_i)$	$O(\psi_i)$	$\chi(\psi_i)$
Mean	52.271	8.623	1036.341	93.760	0.770	0.640	0.249
Std. Deviation	80.080	14.642	3014.316	228.961	0.245	0.396	0.262
95% Conf. Interval	3.565	0.652	134.202	10.194	0.011	0.018	0.012
Range from	48.706	7.971	902.139	83.566	0.759	0.623	0.238
Range to	55.837	9.275	1170.543	103.954	0.781	0.658	0.261

DOI: 10.1002/smr

such as *bindingComps* or *serviceComps*. The *Components* schema has a lot of different things to deal with; thus, the intersection slice contains only a small number of predicates. As the schema is used everywhere, the relationship to other schemas is high. This schema (and with it the handling of the components) has been refactored in the next revision: it has been split into three schemas (*BindingComponents*, *ServiceComponents*, and *OtherComponents*). Whereas *Coupling* decreases to 0.7, the value of *Cov* increases to 0.549 (for the three schemas).

As another example, we can take a look at is the *Tokeneer* specification. There, the state schema *IDStation* has a *Coupling* value of 0.431, which is high (also compared with the average value of 0.224 for this specification). The reason is that this schema includes 14 other states. In fact, whenever this schema is mapped to code, it could be assumed that the resulting section in the program code will also be quite complex to handle.

Although it is not part of this study, it can be assumed that the baseline values, together with the actual values, can be used for a first orientation as argued previously. And yet, another property is revealed when looking at the distribution of the means of different specifications: simple (textbook) specifications are typically very compact, do consist of smaller schemas. The values for cohesion are quite high and *Coupling* is usually low. On the other hand, larger specifications are not so dense and they tend to result in lower values for cohesion (and slightly higher values for *Coupling*).

Figure 17 makes these tendencies explicit. It visualizes the sizes of the schemas and the baseline. The diagram on the left side shows that about 90% of all schemas have a CC between 3 and 130. This is also why the diagram to the right (in the absence of enough specification schemas) is limited to an upper bound of $CC = 130$. Within this range, it can be observed that the values for cohesion decrease with increasing sizes of the schemas, whereas the value of *Coupling* increases. *Coupling* will be around an average level of 0.30, the values of cohesion level off at around 0.60. The mean values in Table XI are still of interest, but the variation suggests to adjust them a little depending on the sizes of the specifications at hand.

5. VALIDITY

With the results of the empirical study, the question of validity arises. The study investigates the relation between (and within) size-based, structure-based, and semantics-based measures as defined in Sections 3.2 and 3.3. It does not claim to consider the notion of coupling and cohesion as experienced by developers. In that case, human beings would have to be involved, and the notion of validity would have been an issue to be addressed from a different vantage point. As there are no human factors to be regarded, only the following issues have to be considered:

- The selection validity, which refers to the degree to which the findings can be generalized,
- The internal validity, which is the degree of causal inferences in the empirical study.

Concerning the selection validity, the 35 specifications have been chosen with care as they are from different fields and written by differently experienced authors. The 20.193 lines of specification text of all

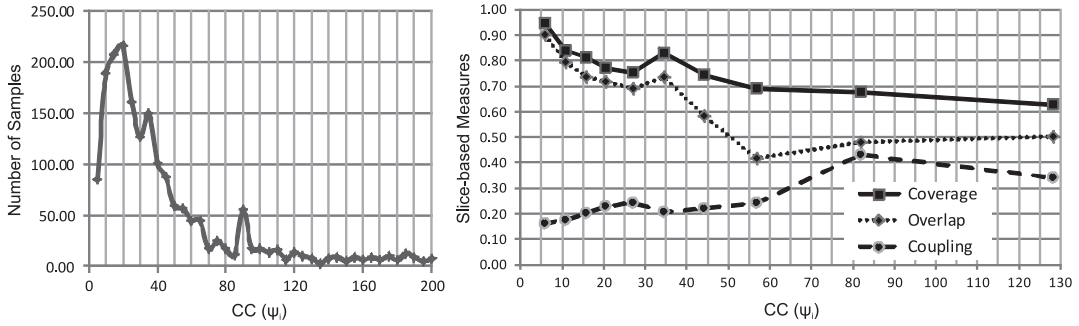

Figure 17. (Left side) About 90% of all schemas (1764 of 1938) have a conceptual complexity between 3 and 130. (Right side) Values of slice-based measures for specifications of raising sizes, up to specifications of size CC = 130.

DOI: 10.1002/smr

1938 Z schemas contain student solutions (55 schemas, 2.8%), textbook exercises (197 schemas, 10.2%) as well as 'real world' specifications (1686 schemas, 87.0%). The 'real world' examples are from practitioners as well as from academic authors, and the diversity of the specifications makes it more likely that the results are valid for a wider class of formal specifications. In addition to that, nearly all of the specifications used in this study are publicly available and the benchmark specifications are available for download at the webpage of the ViZ project. This makes it easier, if necessary, to refer to them and to verify the results later on.

It is important to note that some of the older specifications used in this study had to be modified a bit to be accepted by the CZT parser. The modifications were performed by hand and only dealt with syntactical improvements (hard spaces) so that the parser was able to read the text without syntax errors. Eventually, the '' symbol had to be replaced by the '==' sign to satisfy the Z grammar of the CZT parser. These modifications were carried out with care. To rule out the possibility of coincidental changes of line breaks or identifier names, all of the files were compared again afterwards, using a professional file-compare software.

Considering internal validity, single group and multiple group threads, as well as social threads cannot arise. The only thread that might influence the outcome of the study is the software system used to generate and calculate the measures. The software components involved are the CZT parser, the slicing environment ViZ, *Matlab R2007b*, and *SPSS* 14.0. *Matlab* and *SPSS* are numerical computer environments used for the statistical analysis. Both tools have been used alternately to verify the results of the analysis. It is very unlikely that the data from both environments are erroneous. It is more likely that problems arise because of the parser or the slicer.

The CZT parser is being developed as a *sourceforge* project since the year 2003 and is the basis of a lot of extensions[¶] (for example the Z animations tool *JaZa* or the *Eclipse* plug-in). Irrespective of these various enhancements, the parser is stable. What remains is the slicing environment that might not have worked as expected. ViZ has also been developed in the year 2003 and is also a fundamental part of a couple of extensions. During summer 2011, it has been upgraded to make use of the *CZT* 1.5.0 version, and it is now also able to deal with the dot notation correctly. With this new version, the whole syntax of Z can be parsed. The results of the slicer have been validated systematically during development, on the one hand by comparing the outcome of the slicing environment to documented results to be found in literature, and on the other hand by structured paper-and-pencil tests. Even if this does not totally impede the threat, it reduces at least the chance that implementation faults influence the results.

To qualify the findings of this contribution, it is important to note the study's limitations and also the arguments to mitigate some of them. The limitations are:

- The restricted set of experimental subjects. The study is based on a collection of 35 formal specifications only, providing 1686 specification schemas in total.
- Specific focus. The study does not define complexity and quality-related measures for all state-based specification languages. It also only looks at a limited set of measures.
- Scalability of the approach. It is not clear whether the measures applied (and the conclusions drawn) are applicable to specifications of different/growing sizes.
- A possible experimenter's bias. The author of the contribution is also interested in prediction models and thus looking for special sets of measures suitable for the prediction.

However, the following arguments counterbalance (at least parts) of the possible criticism:

- As mentioned in Section 4.1, about 80% of the schemas stem from real world specifications. In that sense, the 1686 schemas are really representative and dominate by far the set of experimental subjects. Compared with the 63 C programs (summing up to 1.2 MLOC) used in the study of Meyers and Binkley [12], the 20,139 lines of specification text (57,541 lines when including the different revisions of WSDL) do not sound so much. However, the set of experimental subjects is an order of magnitude larger than the sets used in the few studies comparable to this contribution. Samson *et al*. used [48] eight experimental subjects, and Wu and Yi [21] took a look at 12 Z schemas only.
- This contribution focuses on Z on purpose. Z is state-based, and other languages borrow from its simplicity and ideas. On the first sight, is seems to be the case that the measures defined are

[¶]For more details on the CZT *sourceforge* project see http://czt.sourceforge.net. Page last visited: Nov. 2010.

DOI: 10.1002/smr

applicable to other state-bases specification languages, too. This might be true, but to achieve this, several steps are necessary, each of them with their own pitfalls. Firstly, the definitions of the measures heavily rely on the notion of (hidden) dependencies in a specification. For another specification language, one has to think carefully about the identification of these concepts. Secondly, the technique of specification slicing is not new, but it also has to be defined separately (and again carefully) for the new specification language. Finally, as the meaning and the use of predicates in specification languages are different [49] (and as developers thus have to express their thoughts differently), the correlations between the measures might be different, too. As there is now at least sufficient understanding of how these techniques work for Z, this contribution only looks at this specification language and does (yet) not try to map the findings to other languages.

- The collection of experimental subjects comprises specifications of all different types and sizes. In fact, during the past year, it turned out that adding new (and larger) specifications to the set did not influence the overall results any more. The size of a specification does not seem to matter a lot. The reason is that most of the measures are going back to the notion of slices, and in previous papers, it has been shown that slicing works very well with specifications of different sizes** [50, 51]. Concerning the types of the specifications, it has to be admitted that there might be specifications 'out there in the world' changing the situation again. But, the repository used for this study is freely available and can be extended whenever needed.
- The results of this work evolved over a period of 6 years. All intermediate steps have been presented, published, and discussed with colleagues at several conferences and workshops. Although the author has an interest in coming up with a stable set of measures for Z, the ViZ environment and the measures have also be used by others. Recently, under the supervision of Miroslav Staron (University of Gotenburg), Tabarah evaluated the measures in his thesis again [52], coming up with comparable results.

6. RELATED WORK

The basis for the approach presented is the reconstruction of control and data dependencies within Z specifications. The papers influencing this study have already been mentioned in Section 2. Basically, the general idea goes back to the work of Weiser [14, 15], who showed how to compute the set of program statements that affect program values at some point of interest (called the slicing criterion), and to the work of Meyers and Binkley [12], who did the same analysis as in this work, but for a large collection of 'traditional' programs.

Slicing has become an established technique for dealing with the complexity of systems, supporting program comprehension, testing, debugging, program restructuring, downsizing, measurement, and also parallelization. Several surveys introduce into the different notions and variations of slicing [16, 53, 54], as the underlying technique (first defined for imperative programming languages) has soon been extended to other programming paradigms, such as functional programming languages [55–58], object-oriented languages [59–62], concurrent programs [63], and even rule-based systems [64]. Even model checking is supported by slicing. Hatcliff *et al*. [65] apply slicing to software model checking including formal notions of correctness, and Odenbrett *et al*. [66] sketch a slicing algorithm for system specifications written in the Architecture Analysis and Design Language.

Concerning formal specifications, the first approach of 'specification slicing' goes back to Oda and Araki [17]. Their idea has been extended and redefined by Chang and Richardson, Bollin, and Wu [18, 13, 21]. An extension of the slicing approach to Object-Z specifications has been introduced by Brückner and Wehrheim [67]. They also make use of a 'Program' dependence graph that is built from control and data dependencies. Building upon this work, Brückner then used slicing to reduce the space explosion problem during property verification [68].

Although slicing is used for the generation of measures in this contribution, this work focuses on *comprehensibility support* for formal specifications. The reason is that, apart from the intended use of the specification (refinement, test case generation, etc.), specifications constantly change and evolve during

**Experiments show that the bigger the specifications are, the better slicing works. Thus, the sizes of the resulting slices stay approximately the same.

DOI: 10.1002/smr

their creation [69]. And with that, it becomes necessary for developers to have the best support for reading, understanding, and guidance. However, papers that deal with the assessment or measurement of specifications are rare.

One of the first studies that took a closer look at specification measures was conducted by Samson, Nevill and Dugard in 1987. They used examples to show that there is a correlation between specification metrics and metrics of the deferred implementation [48]. The authors demonstrated how, by counting the number of equations, an estimate of effort is possible at a much earlier stage in the development process.

In 1996, Clarke and Wing [70] presented a survey of the use of formal specifications and existing tools. More than 120 references are provided, giving the reader a sense of acceptability of formal methods. Unfortunately, the measures they address are only related to the quality of the final software systems, the development costs and time, and the productivity rate in the projects.

The often cited CDIS (Central Control Function Display Information System) project [71] demonstrated that formal design yields a highly reliable code. The authors showed that formal methods are very effective in acting as a catalyst for testing. However, the main measures in the Coordinated Direct Investment Survey project were the lines of specification text and the time needed to write the specifications.

In 1997, Finney, Rennolls and Fedorce first addressed the issue of comprehensibility of specifications. In an empirical study, they looked at the influence of comments, naming, and structure of Z specifications [72] and found incidences for an influence of comments on the scores obtained by the test groups. Already in 1997, they wrote that 'it would be desirable to have available indices of comprehensibility' for formal specifications ([72], p.11).

In 1998, Karasch *et al*. reported an investigation into the application of slicing an algebraic specification notation and applied the slicing measures defined by Ott and Thuss [24] to OBJ specifications [73]. Their study is based on 73 modules (1340 lines) only and they concluded that slicing techniques can successfully be applied to algebraic specifications, but also admitted that their slice-based measures might not be entirely appropriate for algebraic specifications.

In 2002, an empirical study conducted by Sobel and Clarkson [74] showed that the application of formal analysis provides great benefit to implementation with respect to completeness. One outcome of the study was that the group using formal methods passed nearly 100% of the standard set of test cases in comparison with 45.5% passed by the control teams. Again, the measurement focused on the final products and not on the specifications.

In 2004, Wu and Yi [21] defined simple Z specification quality measures. In contrary to the approach presented in this paper, their measures are based on the use and inclusions of other schemas. They applied them to a small case study and related the measures also to the corresponding object-oriented measures of a deferred Java implementation. They found positive correlations between some of the measures, but their results are based on only 12 schemas and 6 Java classes – an order of magnitude less than the sample size collected for this study.

Other specification comprehension techniques that make use of intraschema relations are those of the transformations to other types of notations [75, 32, 76] or the application of diverse concept location techniques [20]. The list of formal specification projects can be continued (a collection can be found at the Formal-Methods-Wiki page of Bowen [77]), but quality and complexity measures are, apart from size-based measures, not considered in the available documentations.

7. SUMMARY

A lack of suitable measures can diminish the benefits of an idea or approach. This article, therefore, takes a closer look at measures of formal Z specifications to broaden the field of application and acceptability. The three key contributions of the empirical study are:

- The study provides a head-to-head comparison of size-based/structure-based and semantics-based measures. It demonstrates which of these measures offer unique views of the specification, and one result is that slice-based measures are especially good and sensitive descriptors for a specification at hand.
- The study takes a closer look at the evolution of a specification. In a longitudinal study of the WSDL, it shows that slice-based measures can also be used for assessing deterioration effects of formal specifications.

DOI: 10.1002/smr

- Last but not least, the study takes a closer look at the different measures and gives a summary of their means and standard deviations. As the values vary for specifications of different sizes and types, it also provides a baseline for the measures.

The objective of this contribution was not to invent new measures, but to examine whether or not the idea of using dependencies and slices to measure interconnectivity and intraconnectivity also holds for state-based formal specifications. To some extent, this question can be answered positively as parts of the measures describe unique properties. The consolidation of the trust into the different measures has a lot of benefits. With the experiences gained in this study, a prediction model has been developed, relating complexity and quality indicators between Z specifications and ADA code [78]. The longitudinal study of the WSDL specification indicates the usefulness of the measures, but it is to be seen to which extent these results will be supported by other cases having an extended lifetime of evolving specifications. Likewise, adaptations of the quality measures are still issues for further research, and a more comprehensive study on this issue is certainly warranted.

APPENDIX A: SYMBOLS AND ABBREVIATIONS.

TABLE A.1. SUMMARY OF SETS, SYMBOLS, AND OPERATIONS USED FOR MEASUREMENT.

Notation	*Description*	Defined/Used
$SSlice_f(\Psi,\psi,V)$	*Transitive forward slice in a schema ψ of a specification Ψ according to an abstraction criterion V*	Sec. 2.2, Def. 3
$SSlice_b(\Psi,\psi,V)$	*Transitive backward slice in a schema ψ of a specification Ψ according to an abstraction criterion V*	Sec. 2.2, Def. 4
$SSlice_{fb}(\Psi,\psi,V)$	*Transitive forward/backward slice in a schema ψ of a specification Ψ according to an abstraction criterion V*	Sec. 2.2, Def. 5
$SP^x(\Psi,\psi)$	*Slice profile of type x (that is a collection of all possible slices of a schema ψ, where 'x' stands either for 'fb' forward/backward or 'b' backward)*	Sec. 2.2, Def. 6
$SP^x_{\min}(\Psi,\psi)$	*Smallest slice in the slice profile of type x*	Sec. 2.2, Def. 7
$SP^x_{\max}(\Psi,\psi)$	*Largest slice in the slice profile of type x*	Sec. 2.2, Def. 7
$SP^x_{\cap}(\Psi,\psi)$	*Slice intersection (set of primes that are element of every slice in the slice profile of type x)*	Sec. 2.2, Def. 7
$SP^x_{\cup}(\Psi,\psi)$	*Slice union (set of primes that are element of at least one slice in the slice profile of type x)*	Sec. 2.2, Def. 7
$CC(\Psi)$	*Conceptual complexity (number of primes in the specification Ψ)*	Sec. 3.1, Tab. 2
$v'(\Psi)$	*Logical complexity (number of control dependencies in the specification Ψ)*	Sec. 3.1, Tab. 2
$DU(\Psi)$	*Definition-use count (number of data dependencies in the specification Ψ)*	Sec. 3.1, Tab. 2
$\tau(\Psi,\psi)$	*Tightness (ratio of the size of the slice intersection to the size of the schema)*	Sec. 3.2, Tab. 3
$Cov(\Psi,\psi)$	*Coverage (relates the sizes of all possible specification slices to the size of the schema)*	Sec. 3.2, Tab. 3
$Cov_{\min}(\Psi,\psi)$	*Minimum coverage (ratio between the smallest slice in the slice profile and the size of the schema)*	Sec. 3.2, Tab. 3
$Cov_{\max}(\Psi,\psi)$	*Maximum coverage (ratio between the largest slice in the slice profile and the size of the schema)*	Sec. 3.2, Tab. 3
$O(\Psi,\psi)$	*Overlap (measures how many primes are common to all possible specification slices)*	Sec. 3.2, Tab. 3
$F(\Psi,\psi_s,\psi_d)$	*Interschema flow (measures the number of primes of the slices in ψ_d that are in ψ_s)*	Sec. 3.3, Tab. 4
$C(\Psi,\psi_s,\psi_d)$	*Interschema coupling (the normalized ration of the Interschema flow between schemas ψ_s and ψ_d)*	Sec. 3.3, Tab. 4
$\chi(\Psi,\psi_i)$	*Schema coupling (weighted measure of Interschema coupling of schema ψ_i to all other schemas in Ψ)*	Sec. 3.3, Tab. 4
$\delta(\Psi)$	*Deterioration (difference between the average schema size and the average size of the slice intersections)*	Sec. 3.4, Def. 9
$\rho(\Psi_n)$	*Relative deterioration (difference in deterioration values of consecutive versions of a specification)*	Sec. 3.4, Def. 10

DOI: 10.1002/smr

ACKNOWLEDGEMENT

I am very grateful for the comments and hints of the anonymous reviewers, Roland Mittermeir and John Brown, whose constructive remarks motivated me to elaborate further on this topic.

REFERENCES

1. Jones CB. Systematic Software Development Using VDM (2nd edn) Prentice Hall International: Upper Saddle River, NJ 07458, USA, 1990.
2. Hall A Seven myths of formal methods. *IEEE Software* 1990; **7**(5):11–19.
3. Bowen JP, Hinchey MG. Seven more myths of formal methods. *IEEE Software* 1995; **12**(4):34–41.
4. Woodcock J, Davis J. Using Z: Specification, Refinement, and Proof (Prentice-Hall International Series in Computer Science) Prentice Hall: Hemel Hempstead, Hertfordshire, UK, 1996.
5. Ross PE. The exterminators. *IEEE Spectrum* 2005; **42**(9):36–41.
6. Hinchey M, Jackson M, Cousot P, Cook B, Bowen JP, Margaria T. Software engineering and formal methods. *Communications of the ACM* 2008; **51**(9):54–59, doi:10.1145/1378727.1378742.
7. Hierons RM, Bogdanov K, Bowen JP, Cleaveland R, Derrick J, Dick J, Gheorghe M, Harman M, Kapoor K, Krause P, *et al*.. Using formal specifications to support testing. *ACM Computing Surveys* 2009; **41**(2):1–76, doi:10.1145/1459352.1459354.
8. Fenton N, Kaposi A. An engineering theory of structure and measurement. *Software metrics*. Measurement for Software Control and Assurance, Kitchenham B, Littlewood B (eds.). Elsevier: London, UK, 1989; 27–62.
9. Hall A, Dill DL, Rushby J, Holloway CM, Butler RW, Zave P. Industrial practice - impediments to industrial use of formal methods. *IEEE Computer* 1996; **29**(4):22–27.
10. Mittermeir R, Bollin A. Demand-driven specification partitioning. *Modular Programming Languages*, *Lecture Notes in Comp. Science*, vol. 2789, Böszörmenyi L, Schojer P (eds.). Springer Berlin: Heidelberg, 2003; 241–253.
11. Bollin A. Slice-based formal specifiation measures – mapping coupling and cohesion measures to formal Z. *Proceedings of the Second NASA Formal Methods Symposium*, Muñoz C (ed.), NASA/CP-2010-216215, NASA, Langley Research Center, 2010; 24–34.
12. Meyers TM, Binkley D. An empirical study of slice-based cohesion and coupling metrics. *ACM Transactions on Software Engineering and Methodology* 2007; **17**(1):2:1–2:27.
13. Bollin A. Specification comprehension – reducing the complexity of specifications. PhD Thesis, Universität Klagenfurt April 2004.
14. Weiser M. Program slices: formal, psychological, and practical investigations of an automatic program abstraction method. PhD Thesis, University of Michigan 1979.
15. Weiser M. Program Slicing. *Proceedings of the* 5th *International Conference on Software Engineering*, IEEE Press: Piscataway, NJ, USA, 1982; 439–449.
16. Tip F A survey of program slicing techniques. *Journal of Programming Languages* 1995; **3**(3):121–189.
17. Oda T, Araki K. Specification slicing in a formal methods software development. 17th *Annual International Computer Software and Applications Conference*, IEEE Computer Society Press, 1993; 313–319.
18. Chang J, Richardson DJ. Static and dynamic specification slicing. *Technical Report*, Department of Information and Computer Science, University of California 1994.
19. Spivey J. The Z Notation: A Reference Manual (2nd edn) Prentice Hall International: Upper Saddle River, NJ 07458, USA, 1992.
20. Bollin A. Concept location in formal specifications. *Journal of Software Maintenance and Evolution: Research and Practice* 2008; **20**(2):77–105.
21. Wu F, Yi T. Slicing Z specifications. *ACM SIGPLAN Notices* 2004; **39**(8):39–48.
22. Diller A. Z – an introduction to formal methods. John Wiley and Sons, 1999.
23. Malik P. A retrospective on czt. *Software–Practice and Experience* 2011; **41**(2):179–188.
24. Ott LM, Thuss JJ. Slice based metrics for estimating cohesion. *In Proceedings of the IEEE-CS International Metrics Symposium*, IEEE Computer Society: Los Alamitos, CA, USA, 1993; 71–81.
25. Harman M, Okulawon M, Sivagurunathan B, Danicic S. Slice-based measurement of coupling. *Proceedings of the IEEE/ACM ICSE workshop on Process Modelling and Empirical Studies of Software Evolution. Boston, Massachusetts*, IEEE Computer Society: Los Alamitos, CA, USA, 1997; 28–32.
26. Longworth HD. Slice based program metrics. Master's Thesis, Michigan Technological University 1985.
27. McCabe TJ. A complexity measure. *IEEE Transactions on Software Engineering* 1976; **2**(4):308–320.
28. Tai KC. A program complexity metric based on data flow information in control graphs. *Proceedings of the 7th International Conference on Software Engineering* 1984; 239–248.
29. Bollin A. Maintaining formal specifications. *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM* 2005*), Budapest, Hungary*, IEEE Computer Society: Los Alamitos, CA, USA, 2005; 442–453.
30. Ott LM, Thus JJ. The relationship between slices and module cohesion. 11th International Conference on Software Engineering, IEEE Computer Society: Los Alamitos, CA, USA, 1989; 198–204.
31. Wassyng A. Who are we, and what are we doing here? *Proceedings of the 18th International Symposium on Formal Methods FM* 2012, Giannakopoulou D, Méry D (eds.), 2012; 7–9.
32. Idani A, Ledru Y. Object oriented concepts identification from formal B specifications. *Electronic Notes in Theoretical Computer Science* 2005; **133**:159–174.

DOI: 10.1002/smr

33. Cristià M, Alvez M. A collection of sample specifications for the test template framework fastest. http://www.fceia.unr.edu.ar/ mcristia, CIFASIS, Universidad Nacional de Rosario, Flowgate Security Consulting, Argentina. Page last visited: August 2012.
34. Bowen J. Formal Specification and Documentation Using Z: A Case Study Approach. International Thomson Computer Press (ITCP): London, UK, 1996.
35. Jacky J, Patrick M, Unger J. Formal specification of control software for a radiation therapy machine. *Technical Report*, Technical Report 95-12-01, Radiation Oncology Department, University of Washington, Seattle, WA 1997.
36. Woodcock J, Freitas L. An electronic purse (in Z/eves) – specification, refinement, and proof. Oxford Univ. Computing Laboratory, Department of Computer Science (HISE Group): Heslington YO10 5DD, UK, 2006.
37. Hayes I, Flinn B, Gimson R, King S, Morgan C, Sørensen IH, Sulfrin B. Specification Case Studies. Prentice Hall International: Hertfordshire, UK, 1992.
38. Basin D, Kuruma H, Takaragi K, Wolff B. Specifying and verifying the hysteresis signature system with HOL-Z. *Technical Report 471*, ETH Zrich 2004.
39. Abrial JR, Börger E, Langmaack H (eds.). *Formal methods for industrial applications, specifying and programming the steam boiler control*, *lecture notes in computer science*, vol. 1165, Springer, 1996.
40. Brucker AD, Rittinger F, Wolff B. HOL-Z 2.0: a proof environment for Z-specifications. *Journal of Universal Computer Science* 2003; **9**(2):152–172.
41. Freitas L, Utting M, Malik P, Miller T. SourceForge project – community Z tools. http://sourceforge.net/projects/czt. Page last visited: Oct. 2010.
42. Cooper D, Barnes J. Tokeneer ID station – EAL5 demonstrator: summary report. *S.p1229.81.1*, Praxis High Integrity Systems 2008.
43. Miller T, Freitas L, Malik P, Utting M. CZT support for Z extensions. *Proc. 5th International Conference on Integrated Formal Methods (IFM* 2005*)*, Springer, 2005; 227 – 245.
44. Rees DG. Essential Statistics (4th edn) Chapman & Hall/CRC: Boca Raton, Florida 33431, 2003.
45. Fenton NE, Pfleeger SL. Software metrics (2nd edn) PWS Publishing Company: 20 Park Plaza, Boston, MA 02116, 1989.
46. Chinnici R, Moreau JJ, Ryman A, Weerawarana S. Web services description language (WSDL) Version 2.0. http://www.w3.org/TR/wsdl20 2007.
47. Ryman A. Concurrent versioning system log for WSDL 2.0. http://dev.w3.org/cvsweb/2002/ws/desc/wsdl20/ Attic/wsdl20.tex 2007.
48. Samson W, Nevill D, Dugard P. Predictive software metrics based on a formal specification. Information and Software Technology, *5*, vol. 29, Butterworth-Heinemann: Newton, MA, USA, 1987; 242–248.
49. Garlan D. Preconditions for understanding. *Proceedings of the 6th international workshop on Software specification and design*, IWSSD '91, IEEE Computer Society Press: Los Alamitos, CA, USA, 1991; 242–245.
50. Woodward MR, Allen SP. Slicing algebraic specifications. *Information and Software Techn.* 1998; **40**:105–118.
51. Bollin A. The efficiency of specification fragments. Proceedings of the 11th IEEE Working Conference on Reverse Engineering, IEEE Computer Society: Los Alamitos, CA, USA, 2004; 266–275.
52. Tabareh A. Predictive software measures based on formal Z specifications. Master's Thesis, University of Gothenburg - Department of Computer Science and Engineering September 2011.
53. Lucia AD. Program slicing: methods and applications. *Proceedings of 1st Workshop on Source Code Analysis and Manipulation, Florence, Italy, IEEE Comp. Soc. Press, USA*, 2001; 142–149.
54. Silva J. A vocabulary of program slicing-based techniques. *ACM Computing Surveys* 2012; **44**(3):12:1–12:41, doi:10.1145/2187671.2187674.
55. Reps TW, Turnidge T. Program specialization via program slicing. Selected Papers from the International Seminar on Partial Evaluation, Springer-Verlag: London, UK, 1996; 409–429.
56. Biswas SK. Dynamic slicing in higher-order programming languages. PhD Thesis, University of Pennsylvania, Philadelphia, PA, USA 1997. AAI9814822.
57. Ochoa C, Silva J, Vidal G. Dynamic slicing based on redex trails. *Proceedings of the* 2004 *ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, PEPM '04, ACM: New York, NY, USA, 2004; 123–134, doi:10.1145/1014007.1014020.
58. Rodrigues NF, Barbosa LS. Higher-order lazy functional slicing. *j-jucs* 2007; **13**(6):854–873.
59. Larsen L, Harrold MJ. Slicing object-oriented software. *Proceedings of the 18th international conference on Software engineering, ICSE'96*, 1996; 495—505.
60. Liang D, Harrold MJ. Slicing objects using system dependence graphs. *Proceedings of the International Conference on Software Maintenance, ICSM'98*, 1998; 358–367.
61. Nanda MG, Ramesh S. Interprocedural slicing of multithreaded programs with applications to java. *ACM Transactions on Programming Languages and Systems* 2006; **28**(6):1088–1144, doi:10.1145/1186632.1186636.
62. Wang T, Roychoudhury A. Dynamic slicing on java bytecode traces. *ACM Transactions on Programming Languages and Systems* 2008; **30**(2):10:1–10:49, doi:10.1145/1330017.1330021.
63. Cheng J. Slicing concurrent programs: a graph-theoretical approach. *Lecture Notes in Computer Science, Automated and Algorithmic Debugging*, Springer-Verlag, 1993; 223–240.
64. Wakounig D, Bouchachia A. Reverse engineering of rule-based systems. Proceedings of the Eighteenth International Conference on Software Engineering & Knowledge Engineering SEKE 2006, Zhang K, Spanoudakis G, Visaggio G (eds.). Knowledge Systems Institute Graduate School: San Francisco, California, USA, 2006; 45–50.

DOI: 10.1002/smr

65. Hatcliff J, Dwyer MB, Zheng H. Slicing software for model construction. *Higher-Order and Symbolic Computation* 2000; **13**:315–353, doi:10.1023/A:1026599015809.
66. Odenbrett M, Nguyen VY, Noll T. Slicing AADL specifications for model checking. *NASA Formal Methods* 2010, **2010**; 217–221.
67. Brückner I, Wehrheim H. Slicing object-Z specifications for verification. *In ZB* 2005, *Volume 3455 of LNCS*, Springer-Verlag, 2005; 414–433.
68. Brückner I. Slicing concurrent real-time system specifications for verification. *Integrated FM*, 2007; 54–74.
69. Bollin A. Is there evolution before birth? Deterioration effects of formal Z specifications. Proceedings of the 13th International Conference on Formal Methods and Software Engineering, ICFEM'11, Springer-Verlag: Berlin, Heidelberg, 2011; 66–81.
70. Clarke EM, Wing JM. Formal methods: state of the Art and future directions, CMU computer science technical report CMU-CS-96-178. *Technical Report*, Carnegie Mellon University August 1996.
71. Pfleeger SL, Hatton L. Investigating the influence of formal methods. *IEEE Computer* 1997; **30**(2):33–43.
72. Finney K, Rennolls K, Fedorec A. Measuring the comprehensibility of z specifications. *Journal of Systems and Software* Jul 1998; **42**(1):3–15, doi:10.1016/S0164-1212(98)00003-X.
73. Karasch T, J JR, MR SMRW, Allen S. Slicing algebraic specifications. *Information and Software Technology* 1998; **40**(2):105–118, doi:10.1016/S0950-5849(98)00029-9.
74. Sobel AEK, Clarkson MR. Formal methods application: an empirical tale of software development. *IEEE Transaction on Software Engineering* 2002; **28**(3):308–320.
75. Kim SK, Carrington D. A formal mapping between UML models and object-Z specifications. *Lecture Notes in Computer Science* **2000**; 1878:2–21.
76. Fekih H, Ayed LJB, Merz S. Transformation of B specifications into UML class diagrams and state machines. SAC '06: Proceedings of the 2006 ACM Symposium on Applied Computing, ACM: New York, NY, USA, 2006; 1840–1844, doi:10.1145/1141277.1141709.
77. Bowen JP. Formal methods wiki. http://formalmethods.wikia.com/wiki/Formal_Methods_Wiki. Page last visited Oct 9th, 2010 February 2010.
78. Bollin A, Tabareh A. Predictive software measures based on Z specifications – a case study. *2nd Workshop on Formal Methods in the Development of Software 2012 (WS-FMDS 2012)*, Andrés C, Llana L (eds.), no. 86 in EPTCS 86, 2012; 33–40, doi:10.4204/EPTCS.86.5

AUTHORS' BIOGRAPHY:

Andreas Bollin is Associate Professor at the Software Engineering Research Group of the Alpen-Adria Universität Klagenfurt, Austria. He completed his Master degree in Telematics at Graz University of Technology, where he also was a graduate assistant at the Institute for Information Systems and Computer Media. He worked for a Software consulting company (ISCN) in Dublin and Graz, before receiving his PhD in Applied Informatics at the University of Klagenfurt. His research interests include Software Comprehension and Reverse Engineering, Formal Methods, but also Didactics in Informatics and Multimedia Systems. He is member of the IEEE Computer Society and the ACM. Contact him at Andreas.Bollin@uni-klu.ac.at.

DOI: 10.1002/smr