1992

Applying metrics to rule-based systems

Paul Doyle
Technological University Dublin, paul.doyle@tudublin.ie

Renaat Verbruggen
Dublin City University

Recommended Citation

P. Doyle and R. Verbruggen, "Applying metrics to rule-based systems," Proceedings Fourth International Conference on Software Engineering and Knowledge Engineering, 1992, pp. 123-130, doi: 10.1109/SEKE.1992.227938.

Applying metrics to rule-based systems

Paul Doyle & Renaat Verbruggen
School of Computer Applications
Dublin City University
Dublin 9, Ireland.
u0700406@dcu.ie VerbruggenR@dcu.ie

Abstract

Since the introduction of software measurement theory in the early seventies it has been accepted that in order to control software it must first be measured. Unambiguous and reproducible measurements are considered to be the most useful in controlling software productivity, costs and quality, and diverse sets of measurements are required to cover all aspects of software. This paper focuses on measures for rule-based language systems and also describes a process for developing measures for other non-standard 3GL development tools. This paper uses "KEL" as an example and the method allows the re-use of existing measures and indicates if and where new measures are required. As software engineering continues to generate more diverse methods of system development, it is important to continually update our methods of measurement and control.

1. Introduction

Control within software development has been acknowledged as one of the key requirements for establishing predictive procedures and processes. DeMarco summed up this fundamental which underlies the importance of software measurement when he wrote

"*You cannot control what you cannot measure*". [1]

This however leaves us with the difficult task of specifying methods of measurement which are unambiguous, precise, and reproducible. Fenton recently detailed a proposal for implementing such a system after providing the following definition of measurement, which attempts to clarify its limitations. "*measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules*". [2]

Now our goal is to measure attributes of entities which we have identified as 'interesting'. Software metrics/measures have attempted to perform these measures in a scientific and reliable way. However measurements for attributes are not always easily defined for entities (eg. Useability of a system). Indeed there tends to be a substantial amount of confusion regarding 'software metrics' within the software industry. DeMarco attempted to clarify the objectives of measurement within the field of Software Metrics by identifying a common objective: Metrics attempt to measure attributes of software products in order to control its productivity, costs, and quality. This control is achieved by producing models of system behaviour based on historical data with which to compare our current system.

For all aspects of software to be controlled a diverse range of measurements are required. With an ever expanding set of development paradigms it has become necessary to develop measures which incorporate the properties of these new paradigms. Our approach focuses on measures that can be applied to rule-based languages systems and also on a process through which measures for other 4GL development tools my be defined. Using this process we have defined a set of measures for a rule based language which we will refer to as KEL.

2. Quality modelling

Within software development there are two identifiable entities which may be measured. Processes (eg. methods of development etc.) and products (deliverables such as source code or documentation of a product). We can subdivide these characteristics even further by distinguishing between *internal* and *external* attributes. The former refers to those attributes which can be measured purely in terms of the product and process (Lines of Code, modularity, coupling, structuredness etc.), while the latter refers to measurement with respect to how the product or process relates to its environment. ISO 9126 [3] is a proposed European standard which provides a list of nine external attributes which are of

0-8186-2830-8/92 $03.00 © 1992 IEEE

interest to the majority of software developers and customers (Useability, Maintainability, Security etc.).

Managers and users of systems are more interested in these external attributes of entities (managers are concerned with the maintainability of a product whereas users are interested in useability). These external attributes are not directly measurable as definitions tend to be ambiguous. They can however, be measured by defining them in terms of measurable internal attributes. A simple example would be to define reliability as the number of bugs per 100 lines of code. It can generally be said that we use internal attributes to support external ones because we cannot measure external attributes directly.The remaining sections of this paper will focus on the development of measures for product attributes in rule based languages. Quality modelling [4], which involves relating metrics, internal attributes and external attributes to some theoretical framework, will be used to associate external product attributes (sometimes referred to as the factor) to internal attributes (known as the criteria) which in turn are evaluated by using proposed sets of measures.

The use of software engineering methods leads to construction of products with certain structural properties. These properties are characterised by internal attributes such as modularity, re-useability, coupling, cohesiveness, redundancy, hierarchy, and structuredness etc. Some may even state that the verification of the correct implementation of these methods will ensure 'satisfactory' levels of external attributes thus the assumption that 'good' internal structure leads to good' external quality is part of most software quality models.

Although there is an 'intuitive feel' regarding the connection between the internal structure of software products and external product attributes, there is very little scientific evidence to establish specific relationships. This is perhaps the result of difficulties in setting up relevant experiments and a lack of understanding of how to measure important internal product attributes properly.

Defining models of quality aids in the development of a structured process through which attributes of software may be measured, recorded, and re-used in future projects. By providing reliable data, based on historical and measured values, prediction and assessment techniques may be used to control productivity, cost and quality. Measurable targets may be set within software projects which will increase confidence in the producers claims to specified external attributes. Without these attributes being made quantifiable little weight can be associated with claims of a products level of quality.

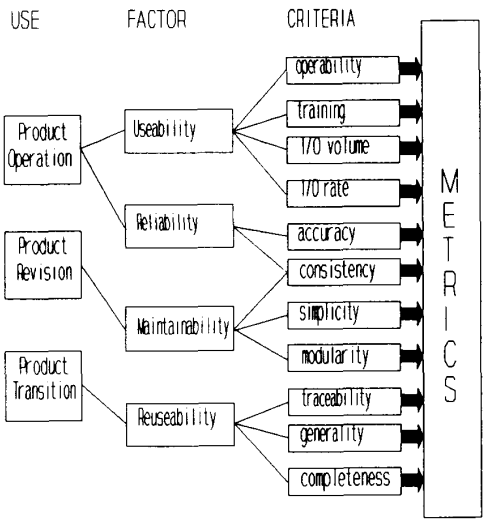

Figure 1 A typical Quality Model

3. Defining measures for rule based languages

The development of measures for languages has been based around third generation languages. The COCOMO model [8] which is perhaps one of the better known cost estimation models provides data for its three models, however this only includes machine languages and 3GLs. Little additional data regarding object oriented, knowledge engineering or relational database languages has been published, although Verner & Tate proposed a process involving Function Point analysis [5], which would provide collection data for COCOMO applicable to 4GLs [6], and a proposal for a suite of metrics for object oriented languages has recently been published by Chidamber and Kemerer [7]. Rule based and relational languages however, have been slower to attract the interest of researchers in the field of metrics. This paper proposes an approach for the development of metrics for a rule based language which will be referred to as KEL.

Below are the steps from which a list of measures for KEL have been developed. This processes is based on approaches commonly used within the existing 3GL framework for measure selection. Necessary alterations were required however to take the presence of rules into account. This was accomplished by producing a generic process that accommodates for systems which are outside the areas of traditional measurement techniques. Each of these steps will be expanded upon as they are related to our

KEL.

3.1 Measurement definition process for software development languages

Step 1. Analyse the language: Identify its components.

Step 2. Decide what components can be measured by existing techniques.

Step 3. Investigate how (if possible) all other components can be measured.

Step 4. Define a model under which these measures make sense.

Step 5. Measure these attributes. Collect data and correlate results obtained with software performance in an attempt to validate the theoretical model.

Step 6. Re-iterate/ modify.

3.2 Implementation of measurement definition process

Step 1. Analyse the language: Identify its components.

The object of this step is to decompose the language into its various development tools and determine what strategy has been employed for each component. If we take a commercial 4GL language, typical components would be a Forms Management System, Report generator, Query language, Database etc. Each of these components employ diverse implementation strategies. For example a query language would be procedural containing textual elements and relations, whereas a forms manager would have a non-procedural nature.

KEL runs on the VAX under VMS. It use the VAX/VMS Record Management System (RMS), and its own Dictionary Management System (DMS). The DMS is the core of the system where data is described. Data types, fields and files are described in the DMS. KEL also consists of On-line, menu, and report program generators, as well as a forms management system.

Component	**Details**
RMS	Part of VMS so its not within the scope of our proposed measures.
DMS	Relationships exist between the files, field and data types (see figure 4). These relationships are similar to those expected to be foundwithin a database system. Knowledge in the form of rules which directly relate to these entities can also be stored. These rules are textual.
Modules	Modules are a combination of procedural textual elements in the form of rules and relationship definitions between other modules and files which are non-procedural ie. form driven.

Step 2. Decide what components can be measured by existing techniques.

This would usually involve an extensive search of publications for articles etc. relating to measurable attributes of software. An attempt has been made to summarise this information to give some indication of the range of measures currently available.

a) De Marco's "Bang" Metric

- Based on the specification documentation
- Functional measure. Implementation independent indicating system size.

Decompose each part of the specification model down to its primitive level. Data elements, objects and relationships are produced from the data dictionary and object diagrams. [1]
This is the earliest predictor of effort which drives the cost model and hence it is only a very rough estimator.

b) Design

Design weight

- Based on design documentation
- Measure of effort implied in the design ie cost.

Projections of effort are more accurate than those obtained from Bang. Primitive measures obtained and weighted.

$$DW = \Sigma \text{ modules weight/ no. modules [1]}$$

COCOMO & Function Point analysis

- Based on design documentation
- Measure the size, effort, and schedule estimation

FP analysis [5] is a measure of system functionality. It measures the size of data processing systems by

using a weighted sum of the number of inputs, outputs, master files, and enquiries. FPs are converted to lines of code (LOC) using an expansion factor for language. eg. COBOL FP = 110 LOC.

COnstructive COst MOdel (COCOMO) [8] uses 3 algorithmic cost estimation models (basic, intermediate, and detailed). Data for these models is available applications written in COBOL, Fortran, PL/1 and assembly.

$$Effort = a\ (\ size\ )^{b} \times product\ of\ cost\ drivers$$

Values for "a" and "b" are based on the mode of development; cost drivers are provided for each mode.

c) Textual Complexity Measures

- Based on lexical elements of source code
- Measure of program size (eg: LOC)

There have been various contributors to this area of measurement and although it is often difficult to find a comprehensive source, frequently referenced books are by Halstead [9], Conte [10], and a recently publish book by Fenton [2].

d) Structural Complexity Measures

- Based on the control structure of procedural source code.
- Measures complexity of modules/programs.

A set of possible control structures are defined and from these flowgraphs and decomposition trees can be automatically generated. Measures are then applied to these graphs [2].

e) Architectural Measures of complexity

- Based on the calling relationship between modules, extracted from source code.
- Measures program complexity

Call graphs, which graphically represent the calling relationship between modules may be generated automatically from the source code of a language. Measures associated with the nodes and arcs have been defined by many researchers [2].

f) Dynamic Measures

- Based on the test coverage of source code
- Measures the Percentage of code executed after a number of test runs.

Dynamic measures are tool based with a selection of metrics proposed. Descriptions of such metrics are to be found in the documentation for such tools [11]

g) Rules / Integrity Measures
Little documented research available.

h) Data/Database complexity Measures
Little documented research available.

Figure 2 indicates which of the measures defined above are applicable to KEL, and which need to be developed. The first set of measures identified were for the earliest period in the software life cycle. Specification documents are implementation independent and thus may be measured regardless of the application development methodology, hence they exist for KEL. Design documentation measures are more dependent on the implementation strategy and thus DeMarco identified two models for measuring design documents (synchronous and asynchronous) , this however is sufficient to cover traditional design methods which can be used for KEL. Textual elements were identified and all measures for traditional software should apply. Modules have a calling structure which is enforced by the menu program generator so architectural measures may be applied, and similarly procedural sections of modules contain common structural components which may be characterised under existing measures.
Other components which are not part of traditional 3GL methodologies have also been identified. However we have indicated that few existing measures for these properties exist. They are: data/database measures of relationships, dynamic measures for non-procedural elements and measures for knowledge stored as rules. These measures needed to be defined.

Step 3. Investigate how (if possible) all other components can be measured.

KEL is a rule based language for developing software for business applications. Projects are developed by first using standard structured analysis and design (eg. SSADM), then identified file structures are implemented within its database. Program development consists of using a forms management system and procedural rule based code. The principle modules within KEL are:

- Data Dictionary System
- On-Line program generator
- Report program generator
- Menu program generator
- Chain program generator

- Batch program generator
- Automatic program documentation
- Utilities
- Run-time environment

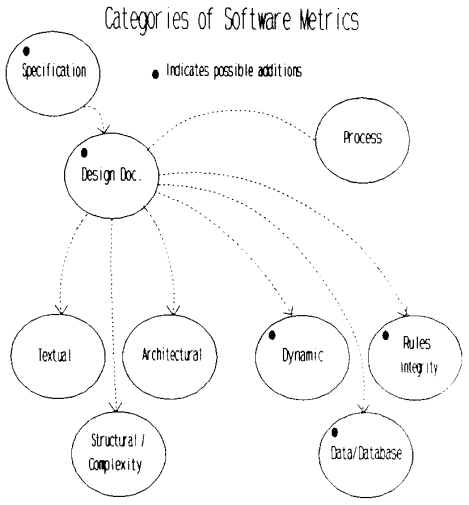

Figure 2 Areas were metrics are applicable to KEL

KEL can be considered to contain four layers of rules. All operations identified can be applied to the three entities identified within the DMS which we can view as three layers and the rules associated with modules.

- Type operations
- Field operations
- File/record operations
- Module/program operations

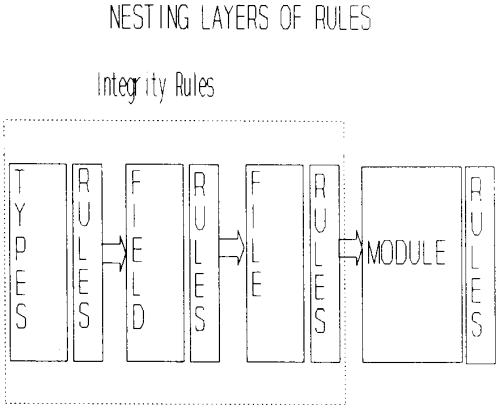

Figure 3 Relationship of rules within specified layers

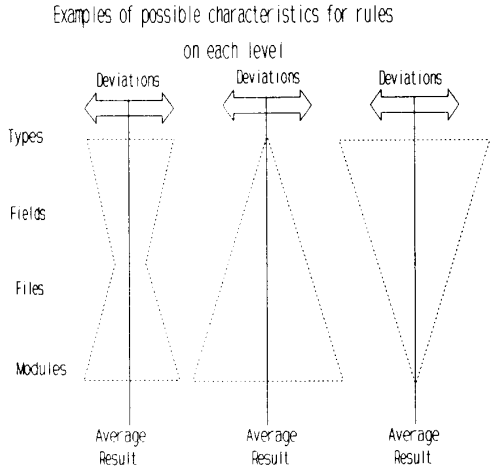

Figure 4 Possible graphically represented characterisation of measures on each level

The first three layers are the dictionary integrity rules contained within the DMS. The differences between all four layers is mainly associated with the complexity of the rules allowed. Most measures defined for rules can be applied to each level. Figure 3 illustrates the relationship between each layer of rules.

The results from metrics applied to each of the four layers may be plotted in an attempt to characterise the relationship between these layers. Figure 6 graphically represents the possible tolerance values for our proposed measures. An average result of the measure is obtained over the four levels. Deviations from this average are plotted to identify differences in these measures for each layer.

(i) Metrics associated with rules

From discussions with language designers and application programmers it was felt that rules were the core KEL resource. Rules consist of one or more statements which differ on each layer by the number of primary and subordinate statements possible. The following are simplified source rules from KEL.

Example 1: Type rule:
Used to define customised data types.

Assert cust_no eq 0 or cust_no in all customer.Cust_no
Else Print "Invalid Customer Number";
End Assert;

Example 2: Field rule:

Used to control values of record fields.

```
Derive totval = valm1 + val2 + val3;
    assert totval < 1500
    Else
        Print "Total value exceeds 1500";
    End assert;
```

Example 3: Record/file rule:

Integrity rules which are not logically associated with just one field can be specified at the record (ie. file) level in the Dictionary rather than the field level. Field rules are active when data is input. Record rules are only activated when the record is committed.

```
If deleting order
Assert no_shipments eq 0
Else
    Print "Cannot delete order, shipments exist";
    Reject;
End assert;
```

Example 4: Module rule:

```
Derive exch_rate_to = currto.exch_rate_pl;
Derive value_ir_to using
if currcode_to = 1
    value_ir_to = value_curr_to;
else
    value_ir_to = value_curr_to/exch_rate_to;
end if;
end derive;
```

(ii) Defining measures

a) Rule / Data relationship

Data items can be identified along with the rules which manipulate them. A data item is a variable named in the module/ record/ field/ type, which appears only once on the RD (rule/data) relationship diagram. Rules are conditional statements within the module/ record/ field/ type. In example 4, the second compound rule (Derive..Using) provides the following relationships (figure 7). The data item *value_ir_to* is connected to Rule 2, and also to the two rules connected to this (IF and ELSE). The data item within the IF statement *currcode_to* is usually implied as the false condition for the corresponding ELSE statement hence its connection to both rules. We derive a graph of the relationship of rules to data items and the following measures may then be defined associated with these relationships.

Average number of rules per data item.
Max number of rules per data item.
Average number of data items per rule.
Max number of data items per rule.

These metrics can be applied to all four layers of the software.

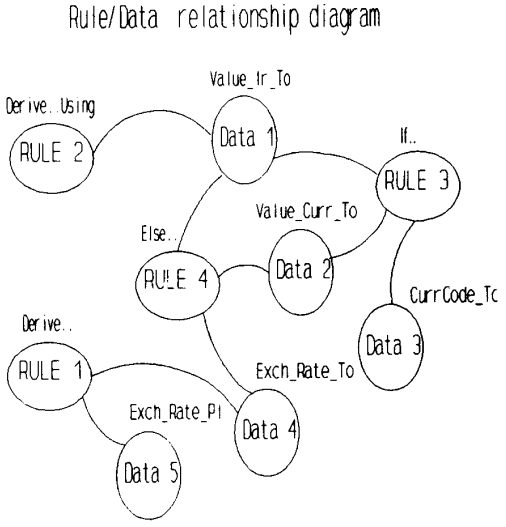

Figure 5 This is a RD relationship diagram for a module rule

We can use the example in figure 5 to calculate the number of connections from each data item to rules and the number of connections from each rule to data items. A measure of coupling can also be defined (ie. the interconnectivity of the graph). Using graph theory we can define an adjacency matrix for the first level of connections as shown in figure 8. Further levels of coupling are obtained by multiplying the matrix by itself.

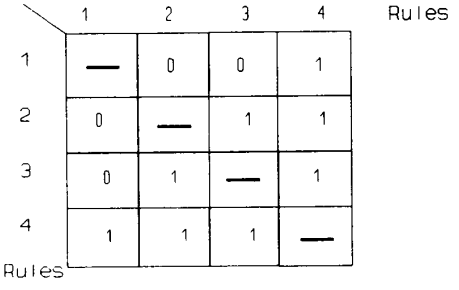

Figure 6 Adjacency matrix for first level connectivity of rules