TOWARD THE MATURITY OF SOFTWARE ENGINEERING: UNIVERSAL, FORMAL, AND MATHEMATICAL DEFINITION FOR VALUE AND OPERATION AS TWO BASIC CONCEPTS OF COMPUTING

1BERNARIDHO I HUTABARAT, 2KETUT E PURNAMA, 3MOCHAMAD HARIADI

1Student, Department of Electrical Engineering, ITS, Surabaya
2Lecturer, Department of Electrical Engineering, ITS, Surabaya
2Assoc. Prof., Department of Electrical Engineering, ITS, Surabaya
E-mail:  1brih@its.ac.id, 2ketut@ee.its.ac.id , 3mochar@ee.its.ac.id

ABSTRACT

The presence of informal and redundant definitions of basic concepts of computing / programming prohibits the advances of software engineering. This problem is not addressed by all literatures of software engineering about formal methods. A paper by the present authors have provided partial solution by establishing Type and Object as two (out of four) disjoint basic concepts of computing and programming. This paper proposes the remaining two of the four basic concepts.

With the substitution test, this paper shows that informality and redundancy of concepts in the widely referenced publications have led to another problem: the circularity of concept. Our proposed concepts have the opposite properties: formal, unique, and non-circular. The definitions are independent toward programming paradigms.

The solution requires the formal definition for expression and operand, a semi-formal definition for statement; and the removal of synonyms like invoke, invocation, parameter and argument. Current standard of software engineering has five synonyms for operation, two synonyms for value, and two synonyms for operand. This paper proposes unique terms, proposes semi-formal and formal definitions for two basic concepts: operation and value. It gives way to advancing the software engineering as a mature discipline.

Keywords: Basic Concept, Value, Operation, Expression, Statement

1. INTRODUCTION

Physics provide engineers of diverse specialties – chemical, electrical, mechanical – the seven unique and formal base quantities/dimensions. Those seven dimensions are length, mass, time, temperature, electric current, substance, and light intensity Engineering students in their first year learn "Concepts Every Engineer should know" [1].


The organization of this paper is as follows. Section 1 overviews the problem and basic concepts of computing. Section 2 elaborates the problems. Section 3 presents formal definition for value and operation. Section 4 applies the theory/hypothesis. Section 5 draws the conclusions. Appendix (sec 6) provides detailed supporting proofs.

2. STATE OF THE ART

Until the present day ref [6] is the only official standard glossary for software engineering. It has been 23 years old. The terminologies (shortened as terms) in numerous research papers have not yet been compiled in the form of [6]. It is thus fitting to review the state of the art mainly from ref [6], with some terms adopted from additional literature. The subsections that follow present the state of the art of the terms.
2.1 Operator, Operation

Reference [6] defines operation as 1

1. In mathematics, the action specified by an operator on one or more operands.
2. In programming, a defined action that can be performed by a computer system.
3. The process of running a computer system in its intended environment to perform its intended functions.

The definition is problematic: it is based on yet another term: action. Moreover, ref [6] does not define action. Instead, it defines operator (written in the 1st definition of operation) as the following:

1. A mathematical or logical symbol that represents an action to be performed in an operation.

Figure 1 shows the quality of a definition through substitution test [7]. We replace the word operator in the first definition of operation. The result is a direct circular definition: the definition of operation is based on the operation itself.

In mathematics, the action specified by a mathematical symbol that represents an action to be performed in an operation on one or more operands.

Fig 1 The First Sentence Produced By Substitution Test

2.2 Instruction

Reference [6] directs the definition of instruction to computer instruction. It defines computer instruction as

A statement in a programming-language, specifying an operation to be performed by computer and the addresses or values of the associated operands.

Fig 2 The Definition Of Instruction From Ref [6]

The definition is problematic. An instruction is not equal to a statement. Section 4 will elaborate statement, expression, and operation-call.


The MOV (or MOVE) is referred to as instruction as well as op code. The two terms are redundant. Notice also that the definition contains the word operation. It is another redundancy.

2.3 Action

The term action is used in HTML (HyperText Markup Language), a very popular programming-language. However, HTML standards [9-10] do not define what action is. W3C school website [11], however, provides a definition as written in fig 3.

The action attribute specifies where to send the form-data when a form is submitted.

Fig 3 The definition of action [11]

The term action is also used in UML (Unified Modeling Language), the most widely used modeling language. Similar to HTML, the UML standards [12-13] do not define what action is.

In absence of the definition from standards, the approximate definition is extracted from a webpage [14] that defines it as in fig 4.

Action is a named element which represents a single atomic step within activity, i.e. that is not further decomposed within the activity.

Fig 4 The definition of action from a UML website

The absence of formal definitions in the standards like [9-10, 12-13] causes ad hoc informal definitions with two problematic properties. The first property is the definition is language dependent (e.g., HTML versus UML). The second property is lack of clarity: the definition is defined on yet other terms (named element, activity).

2.4 Method

In object-oriented programming, a method is a subroutine (or procedure) associated with a class. C# standard [15] subsec 8.7.3 defines method as in Fig 5. Notice the problem of the dependency of definition to yet another term: action.

A method is a member that implements a computation or action that can be performed by an object or class.

Fig 5 The Definition Of Method In C# Standard

Specifies the HTTP method to use when sending form-data

Fig 6 The definition of method for HTML from [14]

2.5 Trigger

SQL standard [18] subsec 4.38.1 defines a trigger as in fig 7. The definition is informal and dependent on other terms: action and operation.

A specification for a given action to take place every time a given operation takes place on a given object.

Fig 7 The definition of trigger in SQL standard [20]

SQL standard defines action and operation. But the definition (which differs from ref [6]) has similar problem: the usage of extra terms; see fig 8.

The action, known as a triggered action, is an SQL-procedure statement or a list of such statements. The object is a persistent base table known as the subject table of the trigger. The operation, known as a trigger event, is either deletion, insertion, or replacement of a collection of rows.

Fig 8 Extended definition of trigger that contains yet extra terms

Figure 9 contains an example trigger. The word operation1 denotes the trigger, the triggered action. The word insert denotes operation, or trigger event.

create or replace trigger operation1
before insert on objects1 for each row
begin
null;
end;

Fig 9 An example trigger in Oracle PL/SQL

2.6 Command

Reference [6] defines command as in fig 10. It is dependent on another term: action, and informal. No precise formula to differentiate it from action.

an expression that can be input to a computer system to initiate an action or affect the execution of a computer program; for example, the “log on” command to initiate a computer session.

Fig 10 Reference [6]’s definition of command

2.7 Routine, Subroutine


A subprogram that is called by other programs and subprograms.

Fig 11 Reference [6]’s definition of routine

Reference [6] goes further with the explanation as in fig 12. It is an admission that the standard (ref [6]) is full with redundant and language-dependent terms. A true engineering standard must not have that low degree of quality.

The terms “routine”, “subprogram”, and “subroutine” are used differently in different programming languages; the preceding definition is advanced as a proposed standard. See also: coroutine, subroutine.

Despite the admission of problem as in fig 12, ref [6] defines subroutine as in fig 13. The definition is informal and imprecise.

A routine that returns control to the program or subprogram that called it.

Fig 13 The definition of subroutine from ref [6]

2.8 Call, Invoke

Computing literatures contain terms like RFC (Remote Function Call), RPC (Remote Procedure Call), and RMI (Remote Method Invocation). Reference [19] contains the term 'method invocation'. Invoke and Invocation are synonymous to call.

2.9 Argument

Reference [6] defines argument as in fig 13. The definitions are informal and imprecise. All three definitions are too similar.

1) An independent variable; for example, the variable m in the equation \( E = mc^2 \). (2) A specific value of an independent variable; for example the value m = 24 kg. (3) A constant, variable, or expression used in a call to a software module to specify data or program elements to be passed to that module. See also: argument; formal parameter.

Fig 14 The definition of argument from ref [6]

2.10 Parameter

Reference [6] defines parameter as in fig 15. The definition is informal, and imprecise in terms of differences between parameter and argument.

(1) A variable that is given a constant value for a specified application. See also: adaptation parameter. (2) A constant, variable, or expression that is used to pass values between software modules. See also: argument; formal parameter.

Fig 15 The definition of parameter from ref [6]

2.11 Operand
Reference [6] defines operand as in fig 16. The definition is similar with argument, and too similar with the operand. No formal differences formulated for the three concepts.

A variable, constant or function upon which an operation is to be performed. For example, in the expression A = B + 3, B and 3 are the operands.

**2.12 Value**

Having seen the problematic definitions for synonyms of operation it is time to see the problems associated with value. No research paper known to the author defines value. The absence of the definition for value gave rise to the similar problem with the ones for operation.

**2.13 Literal**

Reference [6] defines literal as in fig 17: explicit representation of the value of an item. Reference [6] is weak due to the absence of the definition of value and the presence of the definition of literal. If ref [6] is to be consistent, it should replace value with literal in defining the argument and parameter.

In source program, an explicit representation of the value of an item; for example the word FAIL in the instruction: If x = 0 then print “FAIL”.

Reference [6] is not the only literature having the redundant concept. Reference [19] contains similar problem. While value (not literal) is one of ref [19] four core concepts, ref [19] does not define value. Instead, it defines literal as in fig 18.

A literal is a symbol that denotes a value that is fixed and determined by the particular symbol in question.

**3. PROPOSED THEORY AND APPROACHES**

In this section we propose the theory of value and operation as two basic concepts of computing and of programming. After stating the hypotheses we are presenting semi-formal and formal definitions for operation and value. A semi-formal definition is like an informal definition, but with limited vocabulary to help making formal definitions. The proposed theory is adopted from four basic concepts in [22] and similar to four core concepts in [19]. Two of the four basic concepts (i.e., Type and Object) have been formalized in [23]. The rest of this section hypothesizes and defines the two remaining basic concepts: value and operation.

**3.1 Hypotheses**

We hypothesize that

1. Operator, command, routine, subroutine, function, procedure, operator, method, and action are synonyms of operation; any substantial deviation is subjective to the human interpreter of the term.
2. Literal and state are just synonyms of value.
3. Function, procedure, and trigger are specific categorizations for operations.

In the next two sections we present semi-formal definition for operation and value. A semi-formal definition can be formalized in straightforward way
using predicate calculus. By contrast, there is no straightforward way to produce formal definition from informal definition.

### 3.2 Value

Figure 20 shows two semi-formal definitions of value as a basic concept of computing.

\[ A \text{ value is of some type(s).} \quad (1.1) \]
\[ A \text{ value has no identity.} \quad (1.2) \]

*Fig 20 Semi-formal definition for value*

Fig 21 formalizes the definition of value. The **Values** represent the universe of values. The **Types** represent the universe of types. Both Values and Types are subscripted. The ~ represents negation operation, while **is of some type** is assumed as an operation returning boolean value.

\[ \forall \text{Values}_i \in \text{Values} \exists \text{Types}_j \in \text{Types} \]
\[ \text{is of some type (Values}_i, \text{Types}_j) \quad (1.3) \]

\[ \forall \text{Values}_i \in \text{Values} \sim \text{has id} (\text{Values}_i) \quad (1.4) \]

*Fig 21 Formal definitions for value*

### 3.3 Operation

We choose the term *operation* over *operator* for two reasons:

1. Creator of programming languages and tools often equate operator to a subset of system-defined operation (e.g., assignment operator, but not Writeline operator).
2. The term operation is more often found in programming manuals.

Figure 22 contains semi-formal definitions for operation. The formal definition follows fig 22.

1. An operation **has identity**. \quad (2.1.1)
2. a. An operation is a function (special operation) or a procedure (general operation). \quad (2.2.1)
2. b. A general operation returns **value of some type**. \quad (2.3.1)
2. c. A special operation does not return value of any type. \quad (2.4.1)
3. An operation operates operands that can be values, objects, types, or operation(s). \quad (2.5.1)

*Fig 22 The semi-formal definitions for operator*

In order to formally define the concept of operation, we need to define some universes.

- **Values** refer to the universe of values,
- **Oprts** refer to the universe of operations,
- **Objects** refer to the universe of objects,
- **Types** refer to the universe of types.

We can now formalize the concept of operation. Formula 2.1.2 formalizes the semi-formal definition (2.1.1) that an operation has identity.

\[ \forall \text{Oprts}_i \in \text{Oprts} (\text{has_id(}\text{Oprts}_i)) \quad (2.1.2) \]

Formula 2.2.2 formalizes the semi-formal definition 2.2.1. It categorizes any operation into special and general operation. The categorization follows the categorization of types in [23].

\[ \forall \text{General-oprts} \in \text{General-oprts} \]
\[ \forall \text{Special-oprts} \in \text{Special-oprts} \]
\[ \text{General-oprts} \neq \text{Special-oprts} \land \text{General-oprts} \cap \text{Special-oprts} = \emptyset \land \text{Special-oprts} \cup \text{General-oprts} = \text{Oprts} \quad (2.2.2) \]

Formula 2.2.2 can be written as formula 2.2.3. Replace General-oprt by Procedure and Special-oprt by Function. For sake of brevity formula 2.2.3 is the one that mainly used.

\[ \forall \text{Funcs} \in \text{Funcs} \forall \text{Procs}_j \in \text{Procs} \]
\[ \text{Funcs} \neq \text{Procs}_k \land \text{Funcs} \cap \text{Procs} = \emptyset \land \text{Funcs} \cup \text{Procs} = \text{Oprts} \quad (2.2.3) \]

The next step is formalizing the semi-formal definitions 2.3.1 and 2.4.1. For this we need two things listed below:

1. Function \( \tau \) symbolizing the TORV (Type Of Returned Value) of an operation.
2. Type void as special type with empty set of values is used to simulate the presence of type of returned value for procedures.

With the two above assumptions we can define

\[ \forall \text{Funcs} \in \text{Funcs} \]
\[ \tau(\text{Funcs}) \neq \text{void} \land \text{Sov}(\tau(\text{Funcs})) \neq \emptyset \quad (2.3.2) \]

\[ \forall \text{Procs} \in \text{Procs} \]
\[ \tau(\text{Procs}) = \text{void} \land \text{Sov}(\tau(\text{Procs})) = \emptyset \quad (2.4.2) \]

At this point we formalize the semi-formal definition ‘an operation operates the operands’ by specifying two cases: the absence and the presence of operands through formulas 2.5.2 and 2.5.3. Sec
3.4 translate Operands into Values, Operations, Types, and Objects in several definitions.

\[ \exists Oprts_i \in Oprts N (Operands) = 0 \quad (2.5.2) \]

\[ \exists Oprts_i \in Oprts N (Operands) \neq 0 \quad (2.5.3) \]

### 3.4 Expression Versus Statement

A common mistake in several literatures is equating operation (or its synonyms) to statement, as exemplified in fig 2. ISO SQL standard commits similar mistake by using the phrases like ‘DELETE statement’, ‘SELECT statement’ [18].

References [24] states that expression is different from statement but does not define both. Fortunately ref [25] defines expression as in fig 22.

An expression is a construct that will be **evaluated** to yield a value. (2.6.0)

The first author in [22] proposes a more general definition of expression, formulated as 2.6.1. Based upon that definition and the observation of various programming-languages, semi-formal definition of statement is formulated as formula 2.6.2.

Evaluation of expression can return value. (2.6.1)

Evaluation of statement cannot return value. (2.6.2)

**Fig 23 Expression versus statement**

The following sample code illustrates the difference. The former is statement, and the latter is expression.

```c
b; // evaluate (b;) returns void, no value
b // evaluate (b) returns a value, the value of b
```

We assume the presence of functions as follows:

1. **TypeOf**, that takes expression or statement as its only operand. It returns void if the operand is a statement, and non-void (basic value, record value, collection value) otherwise.
2. **SoV**, short of Set of Values. This function takes a type-expression as its only operand. If the operand is void, SoV returns empty set; else it returns set of value from a given type.

\[ \forall Stmts_i \in Stmts SoV (TypeOf(Stmts_i)) = \emptyset \quad (2.6.3) \]

\[ \forall Exprs_j \in Exprs \]

\[ TypeOf(Exprs_j) \in Special-types \]

\[ \| TypeOf(Exprs_j) \in General-types \quad (2.6.4) \]

Reference [22] wrote two additional semi-formal definitions for expression.

An expression can be formed by value only, type only, operation only, object only, or combined occurrences of things denoting the basic concepts. (2.6.5)

An expression is formed by operation(-call) and operand. (2.6.6)

We can formalize the expression using formula 2.6.5. V denotes value, Ob denotes Object, T denotes Type, and Op denotes Operation. The superscript is superscript that denotes the number of occurrences. Thus, \( V^a \) means \( a \) occurrences of value. Similar rule applies for other basic concepts. Formula 2.6.7 then formally defines expression.

\[ V^a Ob^b T^c Op^d \ ; \ a, b, c, d \geq 0 \ ; \ a + b + c + d \geq 1 \quad (2.6.7) \]

### 3.5 Formalizing The Actual Operand In The Expression

Operation is often defined as something that has operands. In previous section we say that operation operates the operands. In practice, operation operates the expressions. However, any expression will be evaluated and can in turn be perceived as a single operand. Thus, we still must answer the question: What is an operand?

The best answer comes from the four basic concepts. There are only four possible form of operand, and that four possible forms are no other than the four basic concepts. Hence, an operand can take form of one of these:

- Value
- Operation
- Type
- Object

Since all the above four basic concepts have been formalized, the concept of operand has been formalized. With that completion, we are now in a position to formalize the concept of expression from the second semi-formal definition.

Formalization of informal description 2.5.1 takes into consideration the valid combinations of operands. Formulas 2.5.4 through 2.5.9 formally define the operation as something that operates values, objects, types, and operations in valid combinations. \( N \) denotes a function accepting an
operation and returning the number of virtual operands.

\[ \exists \text{Oprts}_i \in \text{Oprts} \ \exists \text{Objects}_j \in \text{Objects} \ \exists \text{Values}_k \in \text{Values} \ N(\text{Oprts}_i) > 0 \ \&\& \ \text{operate} (\text{Oprts}_i, \text{Objects}_j) \ (2.5.4) \]

\[ \exists \text{Oprts}_i \in \text{Oprts} \ \exists \text{Values}_k \in \text{Values} \ N(\text{Oprts}_i) > 0 \ \&\& \ \text{operate} (\text{Oprts}_i, \text{Values}_k) \ (2.5.5) \]

\[ \exists \text{Oprts}_i \in \text{Oprts} \ \exists \text{Objects}_j \in \text{Objects} \ \exists \text{Values}_k \in \text{Values} \ N(\text{Oprts}_i) > 0 \ \&\& \ \text{operate} (\text{Oprts}_i, \text{Values}_k) \ (2.5.6) \]

\[ \exists \text{Oprts}_i \in \text{Oprts} \ \exists \text{Values}_k \in \text{Values} \ N(\text{Oprts}_i) > 1 \ \&\& \ \text{operate} (\text{Oprts}_i, \text{Values}_k) \ (2.5.7) \]

\[ \exists \text{Oprts}_i \in \text{Oprts} \ \exists \text{Objects}_j \in \text{Objects} \ \exists \text{Oprts}_t \in \text{Oprts} \ N(\text{Oprts}_t) > 1 \ \&\& \ \text{operate} (\text{Oprts}_t, \text{Objects}_j, \text{Oprts}_t) \ (2.5.8) \]

\[ \exists \text{Oprts}_i \in \text{Oprts} \ \exists \text{Values}_k \in \text{Values} \ N(\text{Oprts}_i) > 1 \ \&\& \ \text{operate} (\text{Oprts}_i, \text{Values}_k, \text{Oprts}_i) \ (2.5.9) \]

The formal formulas 2.5.4 through 2.5.9 represent non-type-expressions that can be summarized by formal formula 2.6.8 (subset of formula 2.6.7).

\[ V^a \ O_b^b \ O_p^d; \ a, b, c, d \geq 0; \ a + b + d \geq 1 \ (2.6.8) \]

Fig 24 There are no types in non-type-expressions

The formal formulas 2.5.10 through 2.5.12 represent type-expression involving operations. References [28-29] digress on type-expression (including the one without operation).

\[ \exists \text{Types}_m \in \text{Types} \ \exists \text{Oprts}_i \in \text{Oprts} \ \exists \text{Values}_k \in \text{Values} \ N(\text{Oprts}_i) > 1 \ \&\& \ \text{operate} (\text{Oprts}_i, \text{Types}_m, \text{Values}_k) \ (2.5.10) \]

\[ \exists \text{Values}_k \in \text{Values} \ N(\text{Oprts}_i) > 2 \ \&\& \ \text{operate} (\text{Oprts}_i, \text{Values}_k, \text{Oprts}_i) \ (2.5.11) \]

\[ \exists \text{Types}_m \in \text{Types} \ \exists \text{Objects}_j \in \text{Objects} \ \exists \text{Oprts}_i \in \text{Oprts} \ N(\text{Oprts}_i) > 2 \ \&\& \ \text{operate} (\text{Oprts}_i, \text{Types}_m, \text{Objects}_j, \text{Values}_k) \ (2.5.12) \]

For the sake of completeness, formula 2.6.9 formalizes general definition for type-expression.

\[ V^a \ O_b^b \ T^c \ O_p^d; \ c > 0; \ a, b, d \geq 0 \ (2.6.9) \]

Fig 25 There must be at least one type in type-expressions

The term is generic and it is useful to use specific terms actual-operand and virtual-operand. The difference between the two is described in the following fragment of C source-code.

```c
void print_it (int virtual_operand)
{ printf ("%d", virtual_operand); }
void main()
{ print_it (5); // 5 is the actual-operand }
```

Fig 26 Virtual-operand versus actual-operand

The difference between virtual-operand and actual-operand is worth exploring to attach the precise semantics to the polymorphic (or polytypic, see [25]) operation. Figure 23 shows the operation-declaration [6] of printf [30].

```c
int printf(const char *restrict format, ...
```

Fig 27 Operation-declaration of printf in [16]

The call to printf can involve only one operation. However, printf is not a unary operation because it can be called with more operands. Referring to the printf as n-ary operation (by possible actual-operands) is more precise than as unary operation – by mandatory virtual-operand. This is our precedent in proposing that the (maximum) number of actual-operand be the cardinality of operation.

3.6 Approaches

The approach used in this paper is linguistic (substitution test) and mathematic. The linguistic approach involves substituting the words and paraphrasing of sentences. The mathematical approach uses predicate calculus.

3.6.1. Id (identity)

We propose the term identity (shortened as id) as a term that is more generic toward name. Names have non-numerical connotation. The operation name (or operation code) MOV in a processor may have corresponding operation id of 111. While both MOV and 111 can both be referred to as identity, the 111 can hardly be referred to as name.

Table 1 Mapping of synonyms for operation
3.6.2. Mappings

We propose the removal of redundant terms. Tables that follow list the mappings to remove redundant terms.

**Table 2 Mapping of synonyms for id**

<table>
<thead>
<tr>
<th>Synonyms</th>
<th>Non redundant term</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>Identity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Synonyms</th>
<th>Non redundant term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op code, mnemonic</td>
<td>Operation id</td>
</tr>
</tbody>
</table>

**Table 3 Mapping of synonyms for call and related phrase**

<table>
<thead>
<tr>
<th>Synonyms</th>
<th>Non redundant term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke</td>
<td>Call</td>
</tr>
<tr>
<td>Method call, method invocation,</td>
<td>Operation call</td>
</tr>
<tr>
<td>Operator call, operator invocation</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4 Mapping of synonyms for operand and related phrases**

<table>
<thead>
<tr>
<th>Synonyms</th>
<th>Non redundant term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument, Parameter</td>
<td>Operand</td>
</tr>
<tr>
<td>Argument, Parameter, Actual parameter</td>
<td>Actual operand</td>
</tr>
<tr>
<td>Formal argument, formal parameter,</td>
<td>Virtual operand</td>
</tr>
</tbody>
</table>

We recognize that some terms deserve to be retained because they convey specific meanings. Table 5 lists three terms and their specific and precise definitions.

**Table 5 Some specific terms that are retained**

<table>
<thead>
<tr>
<th>Term</th>
<th>Dimension</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>V Op T</td>
<td>Operation that returns value of some type.</td>
</tr>
<tr>
<td>Procedure</td>
<td>V Op T</td>
<td>Operation that returns no value of any type.</td>
</tr>
<tr>
<td>Trigger</td>
<td>V Op T</td>
<td>Operation that must be called implicitly, cannot be called explicitly.</td>
</tr>
</tbody>
</table>

The first and second definition of operator in [6] can be rewritten as:

1. An operation operates zero-or-more operands.
2. An operation can be performed by a computer system.
3. A process.

The informal definitions of module presented here is sufficient to provide solution for explaining the action in HTML form. It will be detailed in sec 4.2.

4. RESULT

This section elaborates the result of applying the solutions presented in section 3. Six subsections in this section contain rewritten definitions out from ref [6]. Example definitions are rewritten mainly by removing the redundant terms.

4.1 Removing The Term Operator

The term operator is redundant. Referring to the proposed theory in section 3 we can rewrite the ref [6] definition of operation into something similar but more succinct. The definition of operation – rewritten from ref [6] – is as follows:

1. An operation operates zero-or-more operands.
2. An operation can be performed by a computer system.
3. A process.

The first and second definition of operator in [6] can be rewritten as:

4. An operation is symbolized by an identity.
5. Human operator.

In the above rewritten definition, number (1) is served by formal formula 2.5.3 through 2.5.9; while number (4) is served by formal formula 2.1.2. The rewritten definitions number (2) and (5) need not be formalized. Formalization of the definition number 3 deserves a separate paragraph, the next one.
A process is an operation. Formulas for operation in sec 3 may as well be rewritten by replacing the universe of operations ($Oprts$) with the universe of processes ($Processes$) and essentially nothing changes. We stick with operation because it is practically more general. As example that process is less general than operation, sense that the notion of 'process writeline' is less appropriate compared to 'operation writeline'. Specifically for process, we put it as a synonym in the rewritten semi-formal definition for operation, written in sec 4.5.

### 4.2 Removing The Term Action

Action often means operation. In rare cases, however, the action does not mean operation. In HTML, action means module (a logical translation, see sec 3.12). Assuming the presence of HTML-like language, fig 2 can be rewritten as fig 29.

![Fig 29 Rewritten definition of action in HTML](image)

Fig 29 provides extra aid for understanding. Module1.php is a source-code module whose operation inside it will be used as a post operation.

### 4.3 Removing The Term Method For HTML

While the term OO method deserves its specific term, the term method in HTML does not. The definition in fig 31 (rewritten version of fig 3) is more precise and succinct.

![Operation in the protocol (HTTP) to send form-data](image)

Figure 32 provides extra aid for understanding, through a code written in a hypothetical language similar to HTML.

### 4.4 Trigger As Implicitly Called Operation

Based on the formal and informal definition of operation, we can paraphrase the definition for trigger as in fig 33.

![A trigger is an operation that is automatically called in response to certain events on a particular table or view in a database.](image)

All bulleted sentences are correct. Hence, the semi-formal definitions of operation are universal (and thus worthy as a solid theory). We replace the term operation with command, trigger, subroutine, routine, and action from the semi-formal definition of operation.

### 4.5 Summary For Operations

Let us see whether semi-formal definitions of operation are really universal (and thus worthy as a solid theory). We replace the term operation with command, trigger, subroutine, routine, and action from the semi-formal definition of operation. The following list conveys the result of testing:

- A command, a trigger, a subroutine, a routine, an action, a process has identity.
- A command, a trigger, a subroutine, a routine, an action and a process may or may not return value.
- A command, a trigger, a subroutine, a routine, an action, and a process operates the operands.

All bulleted sentences are correct. Hence, the semi-formal definitions of operation are universal. The terms command, trigger, subroutine, routine, and action are truly redundant.

### 4.6 Removing The Term Literal

Based on the definition of value, the term literal can be removed from software engineering glossary book like [6]. Instead, the definition for value can be used, and example like in fig 34 can be used.

![In source-code, a value (like the string value "FAIL") in the statement: If x = 0 then print "FAIL".](image)
4.7 Removing The Term State

The state transition diagram is really a value transition model; each is a model how the value (of an object) transition from one to another.

Fig 33 is a redrawn example, with terms rewritten according to the proposed theory. In fig 15 the 1 and 2 are referred to as states. In fig 33 the 1 and 2 are referred to as values. Note the term operation in fig 33 replaces the term action in fig 18.

Fig 35 Value-transition model

5. CONCLUSIONS

Current software engineering is not (a mature) engineering. The de jure standard glossary for software engineering contains redundant, informal (imprecise), and circularly defined terms. The use of synonyms for operation and value prohibits the formalization of the two concepts.

The removal of synonyms is the first step to formalizing a concept. This paper proves the redundancy of synonyms for values and operations. Substitution test is used as a linguistic approach to show the problems in current theories.

After synonyms are removed, the concept of value and operation are formulated in semi-formal way. The semi-formal definition is written with limited vocabulary in human language – not mathematical one – to avoid redundant terms. The substitution test as a linguistic approach is again used, this time to show that the definition solves the problem of redundancy. The formulated semi-formal definitions serve equally well if the term operation is changed into command, trigger, action, routine, subroutine, or any other synonyms.

Current standard of software engineering has five synonyms for operation, two synonyms for value, and two synonyms for operand. The proposed theory – if adopted – will make a standard that contains no such redundancy. Uniqueness of concepts reflects a desired property for a software engineering standard.

Finally, this paper proves the concepts can be precisely defined, something that has never been done previously. A standard glossary of software engineering containing semi-formal and formal definition for value, operation, type, and object will be better than the current one. The clear boundaries among concepts mark one step forward toward establishing software engineering as a true and mature engineering discipline.

Three additional terms – operand, statement, expression – are also defined. They are reserved for future researches. However, the definitions are worth considering to be put in a standard.

APPENDIX: TABULATED RESULT

Five tables (6 through 10) compare the current theories versus proposed theory. Some concepts that are reserved for future researches (statement, expression, actual-operand, virtual-operand, operation-call, type-expression, trigger, method, module) are exempted from the tables. Table 6 shows that seven references contributed to eight synonyms for operation, two redundant synonyms for value, and two synonyms for operand.

Table 7 shows that semi-formal definition of operation is shorter than informal definitions of redundant concepts. Table 8 shows that our theory has no excuse for unnecessary redundancy, and contains formal definition. Table 9 shows mappings of definition from two language-centric literatures, applied for two concepts: action and method. Finally, Table 10 compares current theories for value (that contain redundancy and informal) versus proposed theory (unique and formal).
### Table 7 Reference [6] versus proposed theory

<table>
<thead>
<tr>
<th>Current theory [6, 9-14]</th>
<th>Proposed theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation, operator, subroutine, routine, command, method, instruction, action, activity</td>
<td>operation</td>
</tr>
<tr>
<td>value state literal</td>
<td>value</td>
</tr>
<tr>
<td>operand parameter argument</td>
<td>operand</td>
</tr>
</tbody>
</table>

#### Current theory [6]: redundant and wrong terms

**[Operation]**
1. In mathematics, the *action* specified by an operator on one or more operands.
2. In programming, a defined action that can be performed by a computer system.
3. The process of running a computer system in its intended environment to perform its intended functions.

**[Operator]**
1. A mathematical or logical symbol that represents an *action* to be performed in an *operation*.

**[Computer instruction]** A *statement* in a programming-language, specifying an *operation* to be performed by computer and the addresses or values of the associated operands.

**[Command]** an *expression* that can be input to a computer system to initiate an *action* or affect the execution of a computer program; for example, the "log on" *command* to initiate a computer session.

**[Routine]** A *subprogram* that is called by other programs and subprograms.

**[Subroutine]** The terms “routine”, “subprogram”, and “subroutine” are used differently in different programming languages; the preceding definition

#### Proposed theory: unique term

**[Operation]**
1. An *operation* has identity. (2.1.1)
2. a. An *operation* is a function (special operation) or a procedure (general operation). (2.2.1)
2.b A general *operation* returns value of some type. (2.3.1)
2.c. A special *operation* does not return value of any type. (2.4.1)
3. An *operation* operates operands that can be values, objects, types, or operations. (2.5.1)
is advanced as a proposed standard. See also: coroutine, subroutine.

**Table 8** On duplicity and formality: Reference [6] versus proposed theory

<table>
<thead>
<tr>
<th>Current theory [6]</th>
<th>Proposed theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>The terms &quot;routine&quot;, &quot;subprogram&quot;, and &quot;subroutine&quot; are used differently in different programming languages; the preceding definition is advanced as a proposed standard. See also: coroutine, subroutine.</td>
<td>No duplicity, no excuses.</td>
</tr>
<tr>
<td>All definitions are informal.</td>
<td>The concepts of value and operation are formalized. The following two formulas are formalization of the concept value.</td>
</tr>
</tbody>
</table>

\[
\forall \text{Values}_i \in \text{Values} \; \exists \text{Types}_j \in \text{Types} \; \text{is_of_some_type} (\text{Values}_i, \text{Types}_j) \quad (1.3)
\]

\[
\forall \text{Values}_i \in \text{Values} \; \neg \text{has_id} (\text{Values}_i) \quad (1.4)
\]

The formalization of the concept operation uses many formulas: 2.1.2, 2.2.2, 2.2.3, 2.3.2, 2.4.2, 2.5.2, 2.5.3. Not repeated in this table for brevity.

**Table 9** References [14, 17, 18] versus proposed theory

<table>
<thead>
<tr>
<th>Current theory [14]</th>
<th>Proposed theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference [14] The action attribute specifies where to send the form-data when a form is submitted.</td>
<td>The module attribute specifies where to send the form-data when a form is submitted.</td>
</tr>
<tr>
<td>Reference [17] Action is a named element which represents a single atomic step within activity, i.e. that is not further decomposed within the activity.</td>
<td>No further definition needed. See the definition of operation.</td>
</tr>
<tr>
<td>Reference [18] A method is a member that implements a computation or action that can be performed by an object or class.</td>
<td>A method is an operation that has either an implicit local-object in the operation-body, or implicit-operand; but not both.</td>
</tr>
<tr>
<td>Reference [14] Specifies the HTTP method to use when sending form-data</td>
<td>Operation in the protocol (HTTP) to send form-data</td>
</tr>
</tbody>
</table>

**Table 10** References [7, 9] on literal versus proposed theory on value

<table>
<thead>
<tr>
<th>Current theory [7, 9]. Redundant and informal terms: literal and value</th>
<th>Proposed theory. Unique and formal concept: value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference [7] A literal is a symbol that denotes a value that is fixed and determined by the particular symbol in question. Reference [6] In source program, an explicit representation of the value of an item; for example the word FAIL in the instruction: if x = 0 then print &quot;FAIL.&quot;</td>
<td>A value is of some type(s). A value has no identity.</td>
</tr>
</tbody>
</table>

References [6, 7] do not formalize the concept. Formalizes the concept of value. See Table 8.

**REFERENCES:**


