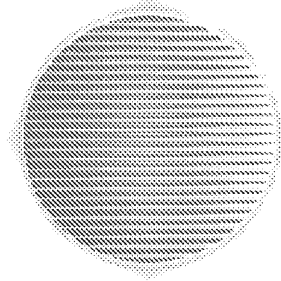
JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE
J. Softw. Maint. Evol.: Res. Pract. 2002; **14**:293–310 (DOI: 10.1002/smr.255)

Research

A semantic entropy metric



Letha H. Etzkorn[1,*,†], Sampson Gholston[2]
and William E. Hughes, Jr[3]

[1] *Computer Science Department, The University of Alabama in Huntsville, Huntsville AL, U.S.A.*
[2] *Industrial and Systems Engineering Department, The University of Alabama in Huntsville, Huntsville AL, U.S.A.*
[3] *U.S. Army Space and Missile Defense Command, Huntsville AL, U.S.A.*

SUMMARY

This paper presents a new semantically-based metric for object-oriented systems, called the Semantic Class Definition Entropy (SCDE) metric, which examines the implementation domain content of a class to measure class complexity. The domain content is determined using a knowledge-based program understanding system. The metric's examination of the domain content of a class provides a more direct mapping between the metric and common human complexity analysis than is possible with traditional complexity measures based on syntactic aspects (software aspects related to the format of the code). Additionally, this metric represents a true design metric that can measure complexity early in the life cycles of software maintenance and software development. The SCDE metric is correlated with analyses from a human expert team, and is also compared to syntactic complexity measures. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: object-oriented metrics; knowledge-based systems; program understanding; domain content; metric validation; syntactic complexity measures

1. INTRODUCTION

Not long ago software engineers wrote most code in a traditional manner, first identifying the problem, then breaking it down into various functions. Recently the object-oriented (OO) paradigm for code development has been shown to reduce faults and improve reusability [1]. OO software is also considered easier to maintain [2]. Over the past 15 years, there have been tremendous increases in the amount of OO code produced. With OO software, what a system does is expressed in terms of interacting 'objects' and classes of objects, each of which provides many services. The collection

*Correspondence to: Dr Letha H. Etzkorn, Computer Science Department, The University of Alabama in Huntsville, Huntsville, AL 35899-0000, U.S.A.
†E-mail: letzkorn@cs.uah.edu

Copyright © 2002 John Wiley & Sons, Ltd.

Received 5 February 2002
Revised 31 May 2002

```
/* Code Segment #1 */
Test[--cnt]->test =
val1[input_count++].counter + val2[tmp_count--]->mycount;


/* Code Segment #2 */
temp = val1[input_count].counter + val2[tmp_count]-
>mycount;
input_count++;
tmp_count--;
--cnt;
Test[cnt]->test =temp;
```

Figure 1. Example of multiple tasks specified in a single line of source code.

of services that a software system provides defines the overall system capabilities. OO software has several characteristics not found in functionally-oriented software. These include:

- objects are encapsulated;
- inheritance allows a subclass of an existing class to supply additional or specialized functionality while reusing the inherited functionality of the ancestor class; and
- parametric polymorphism allows objects to respond to similar commands in different ways.

Due to these OO software characteristics, various metrics suites targeted specifically toward OO software have been developed [3–5]. However, all these OO metrics are defined and calculated using only syntactic aspects of OO software—that is, using only software aspects related to the format of the code or design. Indeed, earlier functionally-oriented metrics were also calculated using only syntactic information. All syntactically-based metrics have the problem that mapping from the metric to the quality the metric purports to measure, such as the software qualities 'complexity' and 'cohesion', is indirect, and often arguable.

2. EXAMPLES OF PROBLEMS WITH SYNTACTIC METRICS

Although size and complexity are truly two different aspects of software, traditionally various size metrics have been used to help indicate complexity. One traditional syntactic metric for measuring the software quality 'complexity' is the lines of code metric. The argument is that software with more lines of code is more complex than software with fewer lines of code. However, this metric has deficiencies. Depending on how the metric is counted, a software component could vary in complexity: some lines of code metrics include comments in the count, whereas other lines of code metrics do not. A more insidious problem is that some programmers will do several tasks in a single line of code, whereas other programmers will do the same tasks on separate lines. A simple example of this (in the C++ language) is shown in Figure 1.

These two code segments do almost exactly the same thing. (The only difference is that, in Code Segment #2, a temp variable is used.) Most lines of code metrics would count Code Segment #1 as one line of code, and would count Code Segment #2 as five lines of code. Another, similar metric (number

Copyright © 2002 John Wiley & Sons, Ltd.

of semicolons metric), would also count the two code segments as quite different in complexity. Note that for readability, Code Segment #2 might in some cases be preferable to Code Segment #1.

Similar examples are available for most existing metrics. In the OO metrics world, consider the Lack of Cohesion in Methods (LCOM) metric [3,5], which measures cohesion—that is, the extent to which the methods of a class perform the same or related tasks. Etzkorn *et al.* [6] discovered several limitations with this metric. The basic idea behind LCOM is methods that both access the same member variable must be performing related tasks. One definition of LCOM is the Li and Henry [5] definition: 'LCOM = number of disjoint sets of local methods; no two sets intersect; any two methods in the same set share at least one local instance variable; ranging from 0 to N; where N is a positive integer.' The Li and Henry version of the LCOM metric was employed for various comparisons in the study reported in this paper, because that version of the LCOM metric performed better than the Chidamber and Kemerer version in earlier studies by Etzkorn *et al.* [6].

One of the problems that Etzkorn *et al.* [6] found with this metric is that the metric can fail to correctly reflect the cohesion of the class when the constructor or destructor functions are used in the method count, depending on how the constructor or destructor is written. If the constructor is used to initialize all the member variables (attributes) in the class, then this metric fails, since each other method in the class will be considered to be cohesive with respect to the constructor function (since they use some of the same member variables). Since the constructor is then cohesive with respect to each method in the class separately, all methods are cohesive with each other, and the class has a maximal cohesion rating, even though the methods themselves may have no tasks in common. Other, similar problems have also been found with the LCOM metric [7–9]. Other syntactic cohesion metrics can be shown to have similar problems [10].

Since different structural aspects of code can result in different metrics values, even when the code is performing the same task, syntactic metrics are not always accurate descriptors of quality. Metrics that provide a better mapping between the software and its associated quality factors thus have the potential to be used in improving software quality, including the quality of newly developed software as well as currently maintained software. Additionally, such metrics could aid in the identification of good potentially reusable software components.

3. PREVIOUS USE OF ENTROPY-BASED SOFTWARE METRICS

In the past, various metrics based on the concept of entropy, which is extensively used in the area of information theory (communications) to estimate the content of messages, have also been shown to be useful in evaluating procedural software's code complexity [11–16]. Chapin [17] showed that entropy (as a measure of message flow) can be used as a software maintenance metric. More recently, entropy has also been applied to OO design complexity [18]. Early versions of software entropy metrics were very syntactically-related. For example, Davis and Leblanc, and, separately, Harrison used the empirical distribution of operators to compute code entropy in procedural programs, where 'operators' were defined to be either a special symbol (mathematical operators), a reserved word, or a function call [11,12]. Berlinger [13] and Kim *et al.* [15] also used entropy as a syntactic complexity measure. More recently, Bansiya *et al.* [18] replaced the 'operators' with 'name-strings' as the basic source of information for the entropy design metric, where name-strings were user-defined names of classes, data declarations, methods, and parameters in class definitions. This was more semantically-related in

Copyright © 2002 John Wiley & Sons, Ltd.

that user-defined name-strings were used instead of program operators. However, the entropy metric of Bansiya *et al.* [18] made no distinction between name-strings that were related to the implementation domain (high-level work area for which the software is solving a problem), and name-strings that were related to the code alone. For example, a name-string such as 'linked_list,' if within the list of class attributes, would have been included in the entropy calculation, even though it was related strictly to the code and not to the implementation domain.

4. CODE-RELATED AND DOMAIN-RELATED INTERNAL DOCUMENTATION

There is a conceptual leap between code-related information (looking only at the computer code itself) and internal-documentation-related information (looking at comments and identifiers within the software). By looking at the code alone, without looking at the related comments and identifiers (names of variables, functions, and classes, etc.), the programmer-analyst can tell that the code provides a linked list. By looking at the comments and identifiers, the programmer-analyst can tell that this code handles employee records. This has been discussed by Biggerstaff *et al.* [19] as part of a discussion relating the examination of code alone to the examination of both the code and the code internal documentation, consisting of comments and identifiers. However, this conceptual leap also occurs within internal documentation itself, considered separately from the code. Using the same example as before, code-related name strings and/or comments will inform the reader that the code provides a linked list. However, name-strings and/or comments related to the implementation domain will inform the reader that the code provides handling of employee records.

Thus, even when considering only information drawn from internal software documentation (comments and identifiers), without looking at the imperative computer code, the complexity of software can be considered to exist in two different forms.

- **Programming information.** The complexity of the code itself which includes the complexity of the algorithm, and the complexity of the data structures involved. When considering only internal software documentation, this code complexity is reflected by comments and identifiers describing the operation of the code.
- **Domain information.** The complexity of the task that the code performs within a domain. This can be re-stated as the human-level complexity of the task within the context of the overall problem area. This information is also described by internal software documentation.

In the best situation for measurement of complexity with syntactic metrics, there would be a one-to-one correspondence between code complexity and domain-task complexity, in that the most efficient known algorithm and data structures would always be selected for a given task. In this case, as is true in many simple cases, measuring the syntactic code complexity would give a good indication of domain-task complexity. However, even in very simple situations, examples can be found that violate this one-to-one correspondence [10]. Consider the case of a SORT problem. Many different SORT routines are available, with order of magnitude of execution differences ranging from $\log_2 n$ to n^2. From this it can be seen that it is possible to change the code complexity while retaining the same domain-task complexity. The domain-, or human-level-, task complexity basically would require the data to be placed in, for example, ascending order, without considering how this was achieved.

Copyright © 2002 John Wiley & Sons, Ltd.

In general, most complexity analyses performed by human beings would include aspects of both code complexity and implementation-domain complexity. Brooks [20] and Duncan [21] contend that human program comprehension includes programming knowledge, domain knowledge, and comprehension strategies. Therefore a complexity metric that provides a code complexity analysis of a software component by the use of a syntactic analysis alone will never provide a complete view of complexity. Thus, a metric that includes an examination of domain-related complexity provides a more human-oriented view of the complexity of a particular task than does a simple syntactical analysis of complexity. If syntactic and semantic complexity analyses are performed together, then a complete view of the complexity of a program can be obtained.

This paper introduces an automated entropy metric, called the Semantic Class Definition Entropy (SCDE) metric, that measures the complexity of classes in OO software, using domain-related information from class documentation (comments and identifiers). In addition to providing a more direct mapping between a complexity measure and the human-oriented view of complexity than is provided by current syntactically-oriented OO metrics, the version of the SCDE metric introduced here can also be considered as an OO design metric, since it analyses the class definition alone. This means that the SCDE metric could be calculated early in the design phase of the development or maintenance life cycle, prior to the implementation-phase development of method code. This would give developers an opportunity to eliminate unwanted complexity early in the life cycle, which should significantly help in reducing rework during and after implementation. The earlier that metrics are applied in a project's life cycle, the more control maintainers and developers can exercise over product quality, cost, and schedule [18]. The SCDE metric can be employed during every software maintenance and development life cycle.

5. THE SEMANTIC CLASS DEFINITION ENTROPY METRIC

5.1. Domain-related concepts and keywords

The basic unit of measurement for the OO SCDE metric is the class. Classes are used as the units for entropy measurement since classes represent the most important fundamental building blocks of an OO system, and are an identifiable abstraction that is present both in designs and implementations. Classes representing real-world entities, ideas, or concepts are keys to a good OO solution. The complexity of a design is significantly influenced by the complexity of its classes and the structural relationship between classes [18].

Bansiya *et al.* used 'name-strings' as the basic source of information for the entropy design metric, where name-strings were user-defined names of classes, data declarations, methods, and parameters in class definitions [18]. The SCDE metric replaces the 'name-strings' of Bansiya *et al.* with 'the domain-related concepts and keywords that are identified as belonging to the class'. The domain-related concepts and keywords for each class are identified, prior to the calculation of the SCDE metric, by the use of an internal-documentation-related program understanding tool. Two examples of such tools include the DESIRE (DESign Information Recovery Environment) system [19], and the PATRicia (Program Analysis Tool for Reuse) system [22,23]. The SCDE metric, as described in this paper, uses the domain-related concepts and keywords provided, for each class, by the PATRicia system; however, a similar program understanding tool could be used instead.

Copyright © 2002 John Wiley & Sons, Ltd.

5.2. PATRicia system background information

The PATRicia system identifies reusable components in OO software [23]. Part of identifying reusable components requires understanding the functionality provided by a program within a domain of interest. To understand a program, the PATRicia system uses a unique heuristic approach, deriving information from the linguistic aspects of comments and identifiers, and from other non-linguistic aspects of OO software, such as a class hierarchy. The module of the PATRicia system that handles program understanding and information extraction is called CHRiS (Conceptual Hierarchy for Reuse including Semantics). When analysing comments, CHRiS first completely parses a sentence using a simple natural language parser, then uses its inference engine to semantically process the various parses. In the case of identifiers, CHRiS uses empirical information on common formats for variable and function identifiers to syntactically tag subkeywords.

CHRiS employs a weighted, hierarchical semantic network (a structure that represents knowledge as a pattern of interconnected nodes and arcs) in which lower-level concepts—in this case conceptual graphs—can infer higher-level concepts. Inference in the semantic network occurs through a form of spreading activation, where active nodes spread to, or infer, surrounding nodes. Natural language tokens from comments and identifiers are applied to an interface layer of the semantic net, which consists of syntactically tagged keywords, and from this layer inference proceeds to the rest of the semantic net.

The semantic network is itself OO, being implemented as objects and messages in the expert system shell of CLIPS version 6.0. In CHRiS, a concept—anything appropriate to the domain that can be represented in a conceptual graph—is implemented as one or more CLIPS objects. When a concept is asserted, weighted messages are sent from that concept to surrounding concepts, and a comparison of the weights provides an inferencing mechanism. The CHRiS module of the PATRicia system has been satisfactorily validated [23].

CHRiS produces (among other reports) a concept report that identifies the concepts and keywords that have been found in each examined class. These concepts and keywords from this report are used in the calculation of the SCDE metric. Figure 2 shows a CHRiS concept report for GnCommand (Class #15), drawn from the GINA GUI package [24]. This report shows the keywords from the knowledge base that have been identified as belonging to the class, along with the part of speech of the keyword, and the number of times that keyword was detected in the class. It shows the concepts (from within interior conceptual graphs) that have been inferred by the CHRiS inferencing engine as being associated with the class. There are two different versions of CHRiS, one that includes inherited concepts in the class report, and one that does not include inherited concepts. The report shown in Figure 2 does not include inherited concepts.

5.3. Definition of the SCDE metric

The basic unit of information for the SCDE metric is the domain (human-level) related concepts or keywords occurring within the class definition that are identified by the PATRicia system (or, as noted earlier, other internal-documentation-based program understanding systems) as belonging to the implementation domain. Through the rest of this discussion, this basic unit of information just described will be referred to as *domain-related concepts or keywords*. A domain-related keyword is a word extracted from the comments and identifiers of a class that has been shown to appear in the

Copyright © 2002 John Wiley & Sons, Ltd.

```
              Class GnCommand

command (adjective,usage adjective) found
1 time(s)
command (adjective,usage adjective) found
1 time(s)


             Class GnCommand

object, with weight 0.000000,
  inferenced 1 time(s),
at location(s) GnObject  found
Facts are:
  "GUI item that can be placed, hid,
  redrawn, scaled, etc."
```

Figure 2. CHRiS report for GnCommand.

implementation-domain knowledge base. A domain-related concept is a higher-level concept, found in the implementation-domain knowledge base, that has been inferred from the presence of related lower-level keywords or concepts, during the inferencing portion of the program understanding tool operation (in this case, during operation of the PATRicia system). Figure 2 provides an example description of keywords and concepts: 'command (adjective, usage adjective)' is a keyword whereas 'object, with weight. . . ' is a concept.

The amount of information conveyed by each of the domain-related concepts or keywords (a sequence of alphabetic characters), S_i, is inversely related to its probability (P) of occurring $P(S_i) = P_i$. Harrison [12] has formalized that the amount of information I_i, in bits conveyed by a single alphabetic string (in this case a domain-related concept or keyword) S_i, with probability of occurrence P_i, is

$$I_i = -\log_2 P_i \tag{1}$$

and that information is additive, i.e. the information conveyed by two domain-related concepts or keywords is the sum of their individual information content.

The probability, P_i, of the ith most frequently occurring domain-related concept or keyword is equal to the percentage of total domain-related concept or keyword occurrences it contributes, that is

$$P_i = \frac{f_i}{N1} \tag{2}$$

where $N1$ is the total number of (non-unique) domain-related concepts or keywords identified as belonging to a class based on information from the class definition and class hierarchy, and f_i is the number of occurrences of the ith most frequently occurring domain-related concept or keyword identified as belonging to the class based on information from the class definition and class hierarchy.

Thus, the average amount of information contributed by each domain-related concept or keyword in a class definition is given by

$$H = -\sum_{i=1}^{N1}(P_i \log_2 P_i) \tag{3}$$

Copyright © 2002 John Wiley & Sons, Ltd.

```
class GnMouseDownCommand : public GnCommand {
    friend class KvView;
    friend class GnView;
    friend class GnApplication;
    friend class GnDocument;
protected:
    int start_x;
    int start_y;
    int last_x;
    int last_y;
    GnTimer *timer;
    Boolean mouse_already_moved;
  private:
    META_DEF_1(GnMouseDownCommand,GnCommand);
  public:
    /* Constructors: */
    GnMouseDownCommand( GnDocument *, GnView *, int
start_x, int start_y );
    ~GnMouseDownCommand();

    // Submit mouseDownCommand to framework
    virtual void submit();

    virtual char *name() {
return("MouseDownCommand"); };

  protected:
    virtual void constrain_mouse (int &x, int &y);
    virtual void draw_feedback (int x, int y,
Boolean clear);
    virtual void track_mouse (int x, int y,
                  Boolean started = FALSE,
                  Boolean finished =TRUE);
    virtual void mouse_idle ( int x, int y );
    virtual void not_submitted ();
    virtual void scroll_before_redo ();

    virtual int hysteresis() { return(2); };
    virtual int auto_scrolling() { return(True); };
    virtual int idle_timeout() { return(250); };
    virtual Boolean call_doit() { return(True); };

    void handle_timeout(caddr_t client_data, GnTimer
*timer);
    void idle_action ();
  public:
    void motion_notify (int x, int y, Boolean
finished =FALSE);
    void button_release ( int x, int y );
  protected:
    get_last_x() { return last_x; }
    get_last_y() { return last_y; }

};
```

Figure 3. Statement of the class GnMouseDownCommand.

Copyright © 2002 John Wiley & Sons, Ltd.

Keywords Recognized by Knowledge-Base
command (adjective) fi = 1
command (noun) fi = 1
down (adjective) fi = 1
mouse fi = 1
down (adverb) fi = 1
Concepts Inferenced by Knowledge-Base
mouse, fi = 1
move_mouse_up, fi = 2
mouse_event, fi = 3
move_mouse_down, fi = 3
track, fi = 1
down fi = 1

$N1$ = total # of non-unique occurrences
$N1$ (keywords) =5 , $N1$ (concepts) =11, $N1$ overall = 16
$n1$ = total # of unique occurrences
$n1$ (keywords) = 5, $n1$ (concepts) = 6, $n1$ overall = 11

SCDE overall	SCDE keywords only	SCDE concepts only
1/16	1/5	1/11
1/16	1/5	2/11
1/16	1/5	3/11
1/16	1/5	3/11
1/16	1/5	1/11
1/16		1/11
2/16		
3/16		
3/16		
1/16		
1/16		
SCDE is 3.28	SCDEa is 2.32	SCDEb is 2.41

Figure 4. Calculation of SCDE, SCDEa (keywords only), and SCDEb (concepts only).

where $n1$ is the total number of (unique) domain-related concepts or keywords identified as being associated with a class definition.

This shall be referred to as the semantic empirical object-class definition entropy. The entropy is computed on a per-class basis with $n1$ being the number of unique (vocabulary) domain-related concepts or keywords identified as belonging to a class based on information from the class definition. This frees the computation of the SCDE metric from dependence on the overall size of the project.

Copyright © 2002 John Wiley & Sons, Ltd.

A larger value of $n1$ implies that a larger amount of domain knowledge is required to understand the class. This directly translates to more complexity. Since we are interested in human-level complexity within an implementation domain, we define complexity as a measure of the effort required to learn, understand, and implement a class within the implementation domain. Accordingly, a class with a larger SCDE metric is going to be more complex than a class with a smaller SCDE metric. The SCDE metric is defined as

$$\mathrm{SCDE} = -\sum_{i=1}^{N1}\left[\left(\frac{f_i}{N1}\right)\log_2\left(\frac{f_i}{N1}\right)\right] \tag{4}$$

where: $n1$ is the number of unique domain-related concepts or keywords; $N1$ is the total number of (non-unique) domain-related concepts or keywords; f_i, $1 \leq i \leq n1$, is the frequency of occurrence of the ith domain-related concept or keyword.

In addition to the general form of the SCDE metric, there are two other forms of this metric, called SCDEa and SCDEb. Both are calculated in the same way as the SCDE metric, except that for SCDEa only the domain-related keywords are used in the metric calculation, while for SCDEb only the domain-related concepts (derived from the knowledge-based inference engine from the keywords) are used in the metric calculation. Class GnMouseDownCommand is shown in Figure 3 [24]. Figure 4 shows an example calculation of SCDE, SCDEa, and SCDEb for class GnMouseDownCommand

6. VALIDATION OF THE SEMANTIC CLASS DEFINITION ENTROPY METRIC

6.1. First experiment

The first experiment performed compared the three SCDE metrics to five other syntactic OO software metrics. All of the metrics were calculated over a set of 68 classes, including several different class hierarchies, drawn from two C++ Mathematical packages, NEWMAT08 [25] and MFLOAT [26], and from three GUI packages, GINA [24], wxWindows [27], and Watson [28]. The five comparison syntactic metrics included the Weighted Methods per Class (WMC) metric [3], which we calculated in the manner proposed by Li and Henry as the sum of the McCabe's cyclomatic complexity values for the methods of a class [5]. It also included the number of attributes in a class, the number of methods in the class (this is the WMC metric with the complexity of each method set to 1), the number of external variable accesses, and the number of message sends.

The null and alternate hypotheses for the first experiment were:

- H_{10}: $\rho = 0$ (null hypothesis)—there is no significant correlation between the specified SCDE metric and the specified syntactic OO metric;
- H_{1A}: $\rho \neq 0$ (alternative hypothesis)—there is significant correlation between the specified SCDE metric and the specified syntactic OO metric.

The Pearson correlation coefficients between the pairs of metrics are shown in Table I. Although the desired strength of correlation required to make decisions based on the data can vary in different research areas, in addition to using statistical significance, some statistics researchers have attempted to make a general definition or rating of which correlation values can be considered good correlation values versus which correlation values can be considered poorer correlation values. Assuming a

Copyright © 2002 John Wiley & Sons, Ltd.

Table I. Correlations of SCDE metrics with five syntactic OO metrics.

	WMC	No. of message sends	No. of external variable accesses	No. of methods in the class	No. of attributes in the class
SCDE	0.235	0.169	0.381	0.230	0.305
	p-value = 0.393	p-value = 0.502	p-value = 0.118	p-value = 0.360	p-value = 0.218
SCDEa	0.774	0.887	0.660	0.354	0.459
	p-value = 0.009	p-value = 0.000	p-value = 0.003	p-value = 0.150	p-value = 0.055
SCDEb	0.148	0.062	0.156	0.156	0.232
	p-value = 0.627	p-value = 0.808	p-value = 0.536	p-value = 0.536	p-value = 0.354

reasonably sized data set, Cohen [29] holds that a correlation of 0.5 is large, 0.3 is moderate, and 0.1 is small. Hopkins [30] calls <0.1 trivial, 0.1 to 0.3 minor, 0.3 to 0.5 moderate, 0.5 to 0.7 large, 0.7 to 0.9 very large, and 0.9 to 1 almost perfect.

Using Hopkin's correlation ratings, the correlation between SCDEa and WMC, SCDEa and the number of message sends, and SCDEa and number of external variable accesses was large to very large, although SCDE and SCDEb had lower correlations. On this basis, the null hypothesis H_{10}, of no correlation between SCDEa and WMC, is rejected. Thus, it can be reasonably assumed, according to the alternative hypothesis H_{1A}, that the SCDEa metric is a predictor of complexity as measured by WMC.

Similarly, the null hypothesis H_{10}, of no correlation between SCDEa and the number of message sends, is rejected, and the SCDEa metric is a predictor of complexity as measured by the number of message sends.

Finally, the null hypothesis H_{10}, of no correlation between SCDEa and the number of variable accesses, is rejected. Thus according to the alternative hypothesis, the SCDEa metric is a predictor of complexity as measured by the number of external variable accesses.

The p-values in Table I are generally good when the associated correlation is a higher value, and poor when the associated correlation is a lower value. Thus the higher correlations, SCDEa and WMC, SCDEa and the number of message sends, SCDEa and the number of external variable accesses, and SCDEa and the number of attributes in a class, show a statistically significant correlation at the 95% confidence level—all had p-values less than 0.05.

6.2. Second experiment

Another experiment was performed that compared the three SCDE metrics and five syntactic OO software metrics separately with a human-level evaluation of complexity. The evaluation team for this experiment (Evaluation Team #1) consisted of 15 students in a graduate software engineering class. They were asked to assess the complexity (as well as other attributes) of a set of classes and class hierarchies drawn from two C++ Mathematical packages: NEWMAT08 [25] and MFLOAT [26]. This included 25 classes drawn from 14 different hierarchies. They also analysed classes drawn from

Copyright © 2002 John Wiley & Sons, Ltd.

three GUI packages: GINA [24], wxWindows [27], and Watson [28]. This included 17 different classes drawn from eight class hierarchies. None of the team members were initially familiar with the systems examined.

All members of Evaluation Team #1 had at least a bachelor degree in computer science or computer engineering, with an average of over four years of relevant industry experience. Using the group inter-rater reliability coefficient specified by Law and Sherman [31], which assesses inter-rater agreement, group inter-rater agreement was checked for this evaluation team, showing some level of inter-rater agreement for their complexity analyses. Evaluations of complexity by this team showed a sample variance (S_x^2) of 0.061, and a random response pattern variance (Σ^2) of 0.125. When r_{wg} represents the within-group inter-rater reliability coefficient, which assesses inter-rater agreement, then $S_x^2 \Sigma^2$ was 0.619, and $r_{\mathrm{wg}} = 1 - (S_x^2 \Sigma^2)$ was 0.381.

The reviewers gave a rating for complexity for each class using the following scale: Excellent, Good, Fair, Poor, and Unacceptable. When considering numeric ratings, the reviewers were told to assume that: Excellent = 1.0, Good = 0.75, Fair = 0.50, Poor = 0.25, Unacceptable = 0.0.

The null and alternate hypotheses for the second experiment were:

- H_{20}: $\rho = 0$ (null hypothesis)—there is no significant correlation between the specified metrics and the Evaluation Team #1 ratings;
- H_{2A}: $\rho \neq 0$ (alternative hypothesis)—there is significant correlation between the specified metrics and the Evaluation Team #1 ratings.

The results from this experiment are shown in Table II. As this experiment shows, when compared to a human-oriented analysis of complexity, SCDEa performed equivalently to the number of message sends and number of external variable accesses metrics, and better than the Li and Henry WMC metric, over the data examined. This is a useful result, since SCDEa was calculated using information from the class definition alone, which is available early in the software development cycle, while the number of message sends metric, the number of external variable accesses metric, and the Li and Henry version of the WMC metric all require the code for each method to be implemented prior to the calculation of the metric. SCDEa performed much better than the number of methods in the class metric (this is the WMC metric with the complexities of all methods set equal to 1) and to the number of attributes of the class metric. This is important, since those two metrics can also be calculated from the class definition alone. This result tends to indicate that the SCDEa metric, which can be calculated early in the software development cycle, is a better predictor of complexity than syntactic metrics which can also be calculated early in the software development cycle.

Since SCDEa versus the evaluation team has a very large correlation (again using Hopkin's correlation ratings) value of 0.774, the null hypothesis H_{20}, of no correlation between SCDEa and Evaluation Team #1, is rejected. Thus it can be reasonably assumed that the SCDEa metric is a predictor of complexity as would be measured by the evaluation team. The three higher correlations in Table II, those of SCDEa, number of message sends, and number of external variable accesses, were all significant at a greater than 95% confidence level—all had p-values less than 0.05.

The SCDE metric presumably has lower values due to its partial input from concepts, since the SCDEb value, which is calculated totally from concepts, also had lower correlation values than was achieved using the SCDEa metric. Although it is possible that this is due to a problem with the metric, this could also be due to a deficiency within the PATRicia system, with respect to this particular data set of software. Further study is required on this matter. To elaborate on the failure possibilities with

Copyright © 2002 John Wiley & Sons, Ltd.

Table II. Correlations of the Evaluation Team #1 ratings and the specified metrics.

	SCDE	SCDEa	SCDEb	No. of Message sends	No. of external variable accesses	No. of methods in class	No. of attributes in class	Li and Henry WMC
Evaluation Team Complexity	0.235 p-value = 0.349	0.774 p-value = 0.000	0.148 p-value = 0.559	0.703 p-value = 0.001	0.712 p-value = 0.001	0.114 p-value = 0.652	0.392 p-value = 0.108	0.445 p-value = 0.158

the PATRicia system, however, assume that the PATRicia system's knowledge base was inadequate in that it did not have sufficient knowledge or the correct knowledge required to analyze these particular classes. This is equivalent to having a software engineer or programmer analyze the classes, when that particular software engineer or programmer is lacking training in the specific area covered by these classes. In both cases, the PATRicia system with an inadequate knowledge base or a programmer with insufficient knowledge, the analysis would be less than complete. This is a potential problem with all knowledge-based systems: they are only as good as their knowledge bases. The PATRicia system was shown in Etzkorn and Davis [23] to have overall good coverage of the graphical user interface (GUI) area, but it has not yet been similarly analyzed for other knowledge areas, such as that of mathematical software packages.

6.3. Third experiment

Finally, a study was performed comparing the SCDE metric to the Class Definition Entropy (CDE) metric previously defined by Bansiya *et al.* [18]. This comparison was performed using 17 classes (drawn from eight class hierarchies) chosen from three GUI packages [24,27,28]. The classes chosen from each GUI package provide a minimal windowing set within the package. The SCDE values and the CDE values for these classes, calculated using the PATRicia system, are shown in Table III. This is a different set of classes than was used for the second experiment.

In Table III, the fact that classes 3, 4, and 17 have zero values for all the SCDE metrics is an example of the limitations of the knowledge-based analysis: for those classes the PATRicia system did not recognize any concepts in its knowledge base. A zero SCDEa can also occur when only one keyword is present (similarly for SCDEb). From the calculation of the metric, the results are $n1 = 1$, $N1 = 1$, $\log_2(1/1) = 0$, so SCDEa $= 0$, and similarly for the SCDEb calculation.

However, when one keyword is present (so that SCDEa $= 0$), and one concept is also present (so that SCDEb $= 0$), SCDE is calculated as $n1 = 1$, $N1 = 2$ (this is because a keyword is currently counted as a different entity from a concept in the PATRicia system knowledge base, even when both contain the same text). This is the reason that classes 1 and 8 have SCDE $= 1$, while SCDEa $=$ SCDEb $= 0$. Hence, SCDEa $= -(-0.5\log_2(0.5)) + (-0.5\log_2(0.5)) = -(-0.5 + -0.5) = 1$.

Copyright © 2002 John Wiley & Sons, Ltd.

Table III. SCDE, SCDEa, SCDEb, CDE, and evaluation team ratings for GUI packages.

Class number	Class name	SCDE	SCDEa	SCDEb	Evaluation Team #1 ratings	Evaluation Team #2 ratings	CDE
1	`wxObject`	1.00	0.00	0.00	0.95	0.95	1.49
2	`wxTimer`	0.00	0.00	0.00	0.94	0.94	1.85
3	`wxbTimer`	0.00	0.00	0.00	0.99	0.99	2.36
4	`wxEvent`	0.00	0.00	0.00	0.98	0.98	2.59
5	`GnContracts`	0.00	0.00	0.00	0.97	0.97	2.88
6	`wxMenu`	1.00	1.00	0.00	0.50	0.50	2.92
7	`GnMouseDownCommand`	3.28	2.32	2.41	0.71	0.71	3.05
8	`GnObject`	1.00	0.00	0.00	0.96	0.96	3.07
9	`wxbButton`	1.82	1.00	1.24	0.90	0.90	3.16
10	`wxButton`	1.88	1.00	1.38	0.79	0.79	3.17
11	`wxWindow`	3.25	1.00	3.05	0.43	0.43	3.24
12	`wxbMenu`	2.52	1.00	2.27	0.86	0.86	3.34
13	`wxbWindow`	3.17	1.00	2.88	0.89	0.89	3.38
14	`wxItem`	0.00	0.00	0.00	0.64	0.64	3.40
15	`GnCommand`	1.00	1.00	0.00	0.92	0.92	3.67
16	`wxMouseEvent`	2.24	0.00	1.92	0.93	0.93	2.35
17	`wxbItem`	0.00	0.00	0.00	1.00	1.00	3.48

The SCDE metric and the CDE metric were separately correlated against the ratings provided for each class by the same evaluation team (Evaluation Team #1) described earlier, as well as an additional evaluation team (Evaluation Team #2). The evaluators' ratings of the classes are shown in Table III.

Evaluation Team #2 consisted of seven knowledgeable C++ developers. All members of this evaluation team had degrees in computer science or computer engineering, and had from 5 to 17 years of experience in software design and development, including 3+ years with C++ and GUI development. None was initially familiar with the systems examined. The rating system employed was the same used by Evaluation Team #1 (described in the second experiment above).

The Spearman rank correlation coefficient (calculated using the Minitab statistical software [32]), r_s, was used to test the significance of the correlation between the SCDE metric and the experts, and between CDE and the experts. Spearman correlation was used here instead of the Pearson correlation (that was used earlier in this paper) due to the smaller size of the data set. Table III shows the relative rankings of each of the 17 classes, for the experts, and for each metric analysed.

The null and alternate hypotheses for the third experiment were:

- $H_{30}: \rho = 0$ (null hypothesis)—there is no significant correlation between the specified metric and the specified evaluation team;
- $H_{3A}: \rho \neq 0$ (alternative hypothesis)—there is significant correlation between the specified metric and the specified evaluation team.

Copyright © 2002 John Wiley & Sons, Ltd.

Table IV. Spearman correlations between SCDE, SCDEa, SCDEb, CDE, and evaluation team ratings.

	Evaluation Team #1	Evaluation Team #2
SCDE	$r_s = -0.571$	$r_s = -0.640$
SCDEa	$r_s = -0.642$	$r_s = -0.720$
SCDEb	$r_s = -0.509$	$r_s = -0.576$
CDE	$r_s = -0.240$	$r_s = -0.328$

The results from this correlation are provided in Table IV. All variations of the SCDE metrics values compared better to each of the evaluation teams than did the CDE values, which tends to indicate that SCDE is a better calculator of complexity than CDE. Using the Hopkin's correlation ratings, all versions of SCDE (SCDE, SCDEa, and SCDEb) have a large to very large correlation with both evaluation teams, while CDE has a small to moderate correlation with the evaluation teams.

The negative correlations shown in Table IV arise from a scale inversion. For the evaluators a higher number was a better (i.e. less complex) complexity rating, and a lower number was a lower (i.e. more complex) complexity rating. For the SCDE metrics, the lower numbers are the better complexity ratings (i.e. less complex), while the higher numbers are the worse complexity ratings (i.e. more complex).

For a sample size of 17 and a level of significance of 5% ($\alpha = 0.05$), the Spearman cutoff for failing to reject H_{30} is 0.49 [33]. Since the computed r_s in each of the SCDE correlations is well above the cutoff, the null hypothesis (H_{30}), that there is no correlation between SCDE and the evaluation team, between SCDEa and the evaluation team, and between SCDEb and the evaluation team, is rejected in each case. Thus it appears that SCDE, SCDEa, and SCDEb are each a predictor of complexity, as would be measured by either of the evaluation teams. In Table IV, the SCDE, SCDEa, and SCDEb correlations with the evaluation teams' ratings were significant at the 95% confidence level—all had p-values less than 0.05.

7. DISCUSSION AND FUTURE RESEARCH TOPICS

Additional validation of these metrics in other domains would be useful, and would provide additional information about some observations made in this study. As another validation procedure, the three SCDE metrics should be compared to the ratings provided by human experts who are asked to separately identify the complexity of the tasks provided by the class in terms of the overall problem domain, as opposed to the complexity of the code in terms of algorithmic and data complexity. The current study compared the SCDE metric to ratings provided by human experts who were simply asked to identify the complexity of the class. In this study, the human experts presumably included both programming knowledge and domain knowledge in their analysis [20], and thus did not differentiate between the code complexity and the domain-task complexity of a class.

Copyright © 2002 John Wiley & Sons, Ltd.

The validation of the SCDE metrics as domain-task-complexity metrics alone would allow the separate analysis of domain-task complexity (by use of the SCDE metric) and code complexity (by use of current syntactic complexity metrics). Comparisons of domain-task complexity to code complexity could be used in reusability analysis. The domain-task-complexity values could also potentially be used to determine the level of cohesion of a class.

8. CONCLUSIONS

Over the data sets examined, the SCDEa metric performed as well as or better than syntactic OO software metrics in comparisons against human-oriented complexity. The SCDEa metric performed as well as syntactic metrics that require the code to be implemented before calculation, even though the SCDEa metric was calculated from the class definition alone. The SCDEa metric performed much better than syntactic metrics that are also calculated from the class definitions alone. The SCDEa metric performed better than the older CDE metric, which can also be calculated from the class definition alone.

Since SCDEa can be collected from the class definition alone, it is a true design metric that can be calculated during the design phase of the software maintenance or development life cycles, before any code has been implemented. Because, in these statistical studies, the SCDEa metric performed much better than other design metrics, it can be concluded that the SCDEa metric could provide a better early indication (in the design phase rather than in the implementation phase) of the design complexity than has been available before.

REFERENCES

1. Hooper JW, Chester RO. *Software Reuse Guidelines and Methods*. Plenum: New York NY, 1991; 254 pp.
2. Pressman RS. *Software Engineering: A Practitioner's Approach* (5th edn). McGraw-Hill: New York NY, 2001; 542.
3. Chidamber SR, Kemerer CF. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering* 1994; **20**(6):476–493.
4. Etzkorn LH, Bansiya J, Davis CG. Design and complexity metrics for OO classes. *Journal of Object-Oriented Programming* 1999; **12**(1):35–40.
5. Li W, Henry A. Object-oriented metrics that predict maintainability. *The Journal of Systems and Software* 1993; **23**(2):111–122.
6. Etzkorn L, Davis C, Li W. A practical look at the lack of cohesion in methods metric. *Journal of Object-Oriented Programming* 1998; **11**(5):27–34.
7. Basili VR, Briand LC, Melo WL. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering* 1996; **22**(10):751–761.
8. Henderson-Sellers B, Constantine LL. Coupling and cohesion (toward a valid metrics suite for object-oriented analysis and design). *Object-Oriented Systems* 1996; **3**(3):143–158.
9. Hitz M, Montazeri B. Chidamber and Kemerer's metrics suite: A measurement theory perspective. *IEEE Transactions on Software Engineering* 1996; **22**(4):267–271.
10. Henderson-Sellers B. *Object-Oriented Metrics: Measures of Complexity*. Prentice-Hall: Upper Saddle River NJ, 1996; 252 pp.
11. Davis J, LeBlanc R. A study of the applicability of complexity measures. *IEEE Transactions on Software Engineering* 1988; **14**(9):1366–1372.
12. Harrison W. An entropy-based measure of software complexity. *IEEE Transactions on Software Engineering* 1992; **18**(11):1025–1029.
13. Berlinger E. An information theory based complexity measure. *Proceedings of the National Computer Conference—1980*. AFIPS Press: Reston VA, 1980; 773–779.

Copyright © 2002 John Wiley & Sons, Ltd.

14. Gonzalez RR. A unified metric of software complexity: Measuring productivity, quality, and value. *Journal of Systems and Software* 1995; **29**(1):17–37.
15. Kim K, Shin Y, Wu C. Complexity measures for object-oriented program based on the entropy. *Proceedings Asia Pacific Conference on Software Engineering, 1995*. IEEE Computer Society Press: Los Alamitos CA, 1995; 127–136.
16. Roca JL. An entropy-based method for computing software structural complexity. *Microelectronics and Reliability* 1996; **36**(5):609–620.
17. Chapin N. An entropy metric for software maintainability. *Proceedings Twenty-Second Annual Hawaii International Conference on System Sciences, Vol. II: Software Track*. IEEE Computer Society Press: Los Alamitos CA, 1989; 522–523.
18. Bansiya J, Davis C, Etzkorn L. An entropy-based complexity measure for object-oriented designs. *Theory and Practice of Object-Systems* 1999; **5**(2):1–9.
19. Biggerstaff TJ, Mitbander BG, Webster DE. Program understanding and the concept assignment problem. *Communications of the ACM* 1994; **37**(5):72–82.
20. Brooks R. Towards a theory of comprehension of computer programs. *International Journal of Man and Machine Studies* 1983; **18**(6):543–554.
21. Duncan RP. A presentation environment for improving software understanding. *Master's Thesis*, Computer Science Department, The University of Alabama in Huntsville, Huntsville AL, 1991; 200–219.
22. Etzkorn LH, Davis CG. Automated object-oriented reusable component identification. *Knowledge-based Systems* 1996; **9**(8):517–524.
23. Etzkorn LH, Davis CG. Automatically identifying reusable components in object-oriented legacy code. *IEEE Computer* 1997; **30**(10):66–71.
24. Baecker A, Genau A, Sohlenkamp M. The GINA Library, version 2.0. Human-Computer Interaction Research Division, Institute for Applied Information Technology, German National Research Center for Computer Science, Berlin, Germany, 1991. http://www.first.gmd.de/connect/gina.html [21 May 2002].
25. Davies R. Newmat—short introduction. R. Davies, Wellington, New Zealand, 1995. http://webnz.com/robert/ol_doc.htm [21 May 2002].
26. Kaufmann F, Mueller W. MFLOAT Version 2.0. Technical University of Graz, Graz, Austria, 1995. http://www.math.tuwien.ac.at/~sleska/html/doshtml/mfloat20/mfloat20.htm [21 May 2002].
27. Smart J. wxWindows Users Manual Version 1.60. Artificial Intelligence Applications Institute, University of Edinburgh, Edinburgh, UK. http://www.wxwindows.org [21 May 2002].
28. Watson M. *Portable GUI Development with C++*. McGraw-Hill: New York NY, 1993; 221 pp.
29. Cohen J. *Statistical Power Analysis for the Behavioral Sciences* (2nd edn). Lawrence Erlbaum Publishing Co.: Mahwah NJ, 1988; 567 pp.
30. Hopkins WG. *A New View of Statistics*. SportScience: Dunedin, New Zealand. http://www.sportsci.org/resource/stats/ [21 May 2002].
31. Law JR, Sherman PJ. Do raters agree? Assessing inter-rater agreement in the evaluation of air crew resource management skills. *Proceedings of the Eighth International Symposium on Aviation Psychology*. Ohio State University: Columbus OH, 1997; 608–612.
32. Minitab. Minitab Release 12 for Windows. Minitab, Inc., State College PA, 1997. http://www.minitab.com [21 May 2002].
33. Bluman AG. *Elementary Statistics*. McGraw-Hill: Boston MA, 1994; 687 pp.

AUTHORS' BIOGRAPHIES

Letha H. Etzkorn is an assistant professor at the University of Alabama in Huntsville. Her research interests include software engineering (especially metrics and program understanding), and mobile and intelligent agents. She has a BEE and an MSEE from the Georgia Institute of Technology, and a PhD from the University of Alabama in Huntsville.

Sampson Gholston is an assistant professor at The University of Alabama in Huntsville. His research interests include quality engineering, design for quality, supplier management, quality management, and applied statistics. He has a PhD from The University of Alabama in Huntsville in Industrial and Systems Engineering and a Masters from The University of Alabama in Industrial Engineering.

Copyright © 2002 John Wiley & Sons, Ltd.

William E. Hughes, Jr is a senior engineer and branch chief of the Operations Research and Economic Analysis group at the U.S. Army Space and Missile Defense Command in Huntsville, Alabama. His work encompasses estimation of requirements and costs, with a primary focus on regression modeling and statistical analyses. He has a BS in Industrial Engineering from Auburn University, an MS in Operations Research from Florida Technology University, and is a Doctoral Candidate in Industrial and Systems Engineering and Engineering Management at the University of Alabama in Huntsville. His dissertation effort involves discovery and development of a series of regression models to predict the magnitude of development phase cost growth in missile systems.

Copyright © 2002 John Wiley & Sons, Ltd.