Integrated Computer-Aided Engineering 21 (2014) 235–247
DOI 10.3233/ICA-140465
IOS Press

A communication-channel-based representation system for software

Zekai Demirezen[a,*], Murat M. Tanik[b], Mehmet Aksit[c] and Anthony Skjellum[a]
[a]*Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL, USA*
[b]*Department of Electrical and Computer Engineering, University of Alabama at Birmingham, Birmingham, AL, USA*
[c]*Department of Computer Science, University of Twente, Enschede, The Netherlands*

Abstract. We observed that before initiating software development the objectives are minimally organized and developers introduce comparatively higher organization throughout the design process. To be able to formally capture this observation, a new communication channel representation system for software is developed in three stages a) set-theoretical representation of software design, b) mapping of software design to a communication channel formalism, and c) hierarchical decomposition leading to higher organization. This new representation system provides a better understanding of the software design by introducing a stepwise entropy reduction notion to the design process. Formal representation of hierarchical decomposition of software and entropy-reduction view of software design provides a stronger bridge between established engineering methods and software design, opens up new possibilities in software research, connecting software with information and coding theory.

Keywords: Software design, entropy, communication channel, hierarchical system, organization

1. Introduction

Engineering is the study and practice of developing solutions to technical problems that are timely, cost-effective, and reliable [2,25,36]. Engineers solve technical problems by applying mathematical and scientific knowledge to develop artifacts [9,18]. Software engineering in particular is an engineering discipline whose focus is the production of high quality software systems [35]. It is a challenging task to produce a high quality artifact within the cost and time parameters, especially for complex projects. Systematic software design methodologies reduce the cost of software development and improve the quality of software products [9,14].

A deeper analysis of all these developments reveals that software engineers evidently focus on abstractions such as data, function, and control abstractions in order to master the complexity in software systems. These abstractions, which facilitate systematic decomposition, have been provided in the form of programming language constructs and design tools [14,37,41, 42]. Significant effort has been expended over several decades to find new design techniques, programming languages, and other strategies for the production of software. In the early days, programs were implemented as a single block of instructions. Over time, as problems became more complex and computers became more powerful, the size of the programs has correspondingly increased. Conceptually controlling the large blocks of instructions proved to be difficult for developers. Naturally, to tackle the complexities, language designers started applying hierarchical decompositions techniques [6,16,32]. Large programs were organized into subprograms. Therefore, as a decomposition strategy, numerous approaches were introduced. These approaches, which can generally be grouped under the category of module-based programming, amounted to the development of constructs, such as

*Corresponding author: Zekai Demirezen, Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL, USA. E-mail: zekzek@uab.edu.

function, subroutine, and modules [20,35]. Further developments in hardware technologies and changing requirements led to the need for implementing even more sophisticated programs. Therefore, the shortcomings of module-based abstractions and decomposition eventually became pronounced [6,14,32]. As such, the need for even more advanced decomposition techniques resulted in the development of yet newer programming languages, newer development paradigms, and programming constructs. Recognition of design strategies led to the development of Object-Oriented [6] and Aspect-Oriented techniques [19].

The historical objective in developing these approaches and their associated methodologies and tools has been that the software engineer should maintain conceptual control over the developed design by hierarchical series of abstractions. Although these approaches has been successful in their own right in providing strong support for engineers, the size and complexity of software systems have greatly increased and software engineers' ability to maintain conceptual control has not improved correspondingly. These methodologies are fundamentally linguistic in nature [14,35] and they present difficulty developing an engineering foundation [35].

It is clear that we need to formalize software development to improve our ability to maintain intellectual control and thus to cope with increasing complexity in software systems [30,33,34]. In this paper we present a communication-theory based foundation of software design by combining concepts from the theory of decomposition of complex systems [4,30,33], and communication channel abstraction [10,11].

We observed that before initiating software design process the objectives are minimally organized and designers introduce comparatively higher organization throughout the design process. Every design decision resolves some kind of an unclear situation in design objectives and reduces the number of possible alternatives.

Historically, this process of transforming disorganization to organization is considered to be a concern of complexity analysis [26,30,34]. Therefore, from a complex-system perspective, the software design problem is a form of complexity analysis and system decomposition.

We started with mapping of software systems to set-theoretical representations. Software systems are represented with an arbitrary number of variables. Next we showed the information transfer between variables and demonstrated the correlations among variables as communication channels. Consequently, we developed a communication-channel representation of software systems.

Our research program is to develop a practical yet formal approach to be able to deductively reason (using computer not necessarily mathematicians) about software design. To our knowledge this paper is the first attempt of this kind. From the technical and formal engineering perspective, a computer collectively (hardware, firmware, and software) and essentially can be modeled as a communication channel. The Processor-Memory-Switch model (PMS) combined with Instruction-Set-Processor (ISP) makes up the computer [5]. In our research we pursue to develop a common formalism with which we will be able to model PMS as well as ISP. In this paper we explore only to develop a communication channel representation of software (ISP). This means, we can deductively reason about designing a computer (hardware and software) in the future if we succeed in our attempts.

The next section, Section 2, covers information concepts such as entropy, transmission, correlation formulas, which are useful in the study of software systems in the succeeding sections. Section 3 presents a mathematical concept of organization and of systems used to model software design. Section 4 introduces general design principles from the perspective of design decomposition and related concepts. Section 5 introduces set-theoretical representation of software, communication-channel formalism of software, and hierarchical decomposition of software. The summary, Section 6, concludes the paper.

2. Information-theoretical foundations

The beginnings of the development of information theory can be traced to the initial considerations for the development of the concept of entropy. Theoretical contributions involving entropy functions started in the original investigation of heat phenomena [12]. In the mid 1800s, Carnot explained the limitations in the heat-work transformation using a flowing substance model. He observed that some energy is lost even in the most efficient engine possible [15]. Eventually, Clausius formulated the dissipation of useful energy in terms of a new quantity which he denoted *Entropy* [12].

Following Carnot's observation, Maxwell, Boltzmann, and Gibbs defined heat as disordered motion of atoms and molecules with consideration of the atomic nature of matter [12]. Their foundational investigations eventually initiated a new branch of mechanics, called *Statistical Mechanics* [12]. Ludwig Boltzmann studied *Entropy* as being a measure of degree of orderliness or disorderliness of gas molecules [12].

In the field of communication, Nyquist and Hartley introduced a quantification technique to measure the information in a message. It took about two decades after Hartley's paper for the introduction of a general theory called *Communication Theory*, by Shannon [29]. He demonstrated fundamental theorems for noiseless and noisy channels and established the transmission rate limit for a given channel and a source. Shannon's information measure includes two variables, the sender's and the receiver's state. McGill presented an extension of Shannon's measures to multivariables. He also developed the associated quantitative formulations of transmission, interaction, and correlation concepts for multivariate analysis [23].

2.1. *Techniques used: The quantitative study of information*

Information theory literature defines the *communication channel* as a mathematical object which connects input variables to output variables in a probabilistic manner [29]. A communication channel is represented by an input set $X = \{X_1, \ldots, X_n\}$, an output set $Y = \{Y_1, \ldots, Y_m\}$, and a set of conditional probabilities $P(X_k \mid Y_l)$ for all k, l.

The uncertainty concerning the input set X, denoted by $H(X)$, is called *source entropy*. It is the uncertainty concerning which symbol will be transmitted. The output set Y consists of all the possible symbols that will be received. The amount of uncertainty in the receiver part, denoted by $H(Y)$, is called *receiver entropy*. Therefore, $H(Y)$ may include uncertainty which the sender should not account for. The conditional entropy $H(Y \mid X)$ is the measure of this uncertainty and it is equivalent to *noise*. In other words, part of the source entropy may not be received by the receiver because of noise. The quantity $H(X \mid Y)$ is the average amount lost, and it is called *equivocation* [29]. The *amount of information transmitted*, $T(X : Y)$, is the uncertainty shared by both input and output sets.

Shannon's Entropy Formula is a measure of the entropy of a variable X that is by definition the sum[1]

[1] Unless otherwise specified, we shall use logarithms of base 2. The unit of H is in *bits*.

$$\begin{aligned} H(X) &= -p_1 \log p_1 - \cdots - p_N \log p_N \\ &= \sum_{i=1}^{N} \varphi(p_i) \end{aligned} \tag{1}$$

where $p_i = P(X_i)$ and $\varphi(p) = -p \log p$.

The observed transmission between two variables, X, and Y, is defined as follows:

$$T(X : Y) = H(X) + H(Y) - H(X \cdot Y). \tag{2}$$

A transmission function can be generalized to an arbitrary number of variables. *Correlation* [23,26] among variables $U_1, U_2, \ldots, U_n$ is the total information transmission and by definition as follows:

$$\begin{aligned} C(U_1, U_2, \ldots, U_n) &= \sum_{i=1}^{n} H(U_i) - H \\ &\quad (U_1 \cdot U_2 \cdot \ldots \cdot U_n). \end{aligned} \tag{3}$$

The quantitative study of information given in this section is used to model information transfer among the software elements in Section 5. The total information transfer is utilized for the formal investigation of software design as a decomposition of software systems into subsystems.

3. Necessity of systems approach

The basic idea that underlies statistical mechanics is that an organized system has a lower entropy than a disorganized one [26,38]. The difference between these two systems can be defined as a reduction in the entropy, and the difference can be calculated by the methods of statistical mechanics. In statistical mechanics, an organized system is composed of ordered molecules [12]. The states of particles and correlation among these particles were demonstrated with *Entropy* term. Watanabe demonstrated the information calculation as the measure of organization [38]. Rothstein used the redundancy calculation to demonstrate the organization [26]. In communication theory, correlation is defined with the *Redundancy* term [29].

Ashby made major contributions to the information theoretical analysis of complex systems [4]. He defined a complex system as a set of variables with constraints. He mentioned that the presence of organization arises from communication between variables. In his seminal paper, he examined the constraints as multivariate relationships within a system and denoted them as *Internal Informational Exchange*. He showed that when the variables are related, constraints exist and they can be quantified with information theory.

Conant [8] applied information theory to system decomposition. He studied pairwise interaction of variables in a dynamic system and provided a technique to decompose a system into weakly connected subsystems. His technique detects subsystems of a complex system while quantifying the interactions among the variables.

3.1. Techniques used: The system perspective

Simon, in his development of a *science of design*, investigated the nature of systems in general [30]. He defined *complex system* informally as the composition of a large number of components interacting in a complex way. Typically, in systems, the whole exhibits emergent behavior and becomes more than a linear sum of the parts.

Watanabe and others introduced a formal treatment technique for the analysis of complex systems following Simon's definition [8,26,38]. Their preferred starting point was the mathematical notion of the structure of organization, which was conceptually identical to the systems view of Simon. In this perspective, a complex system is composed of correlated subsystems or elements [4,38]. The formal analysis of a complex system is therefore, related to the degree of correlation among its subsystems or elements [4,38]. Naturally, correlation can indicate the level of depth and breadth of interactions among subsystems or elements. Therefore, this correlation can be used to show the degree of interaction of subsystems. An organized structure includes redundancy and the amount of redundancy reduces the information required to reveal that structure [26,38]. Therefore, structure provides information. If we recall Shannon's results, we observe that in communication theory, the structure of a system implies the structure of a message and redundancy within the message, which corresponds to the amount of uncertainty [4,26,38].

The degree of organization increases if the degree of uncertainty of the system decreases despite a large degree of uncertainty on individual components. Thus the strength of organization is measured by the balance between the uncertainty of the components with respect to the uncertainty of the whole. Since entropy is a measure of uncertainty, then the degree of organization can be defined as

$$\begin{aligned} \mathit{Organization} &= (\mathit{sum\ of\ entropies\ of\ parts}) \\ &\quad -(\mathit{entropy\ of\ whole}). \end{aligned} \tag{4}$$

In a sense, decomposition of a complex system is a matter of identification of its components and their interactions. A many component system interacting in a complex way is naturally not conducive to the observation of its component interactions. The difficulty rests in the identification of all the system components, a requirement for decomposing the system into loosely coupled subsystems or elements [30].

One can define the system as a set of variables and observe the correlation between them [26,38]. It is assumed that the information flow within the system is representative of the relations between the variables [4, 38]. As mentioned above, the answer lies in the hierarchical decomposition of the total correlation [4,38]. There are multiple ways of producing such a decomposition scheme. It is a matter of the "parameter of interest" [35] to decide, which depends on the purpose of the analysis. To reduce complexity, strongly connected elements are grouped into subsystems [8,30]. This highlights that correlations among subsystems are weaker than correlation within subsystems.

4. General design principles and software design

In traditional engineering disciplines, design is considered to be a fundamental activity [1,17,25,27]. The act of design starts with recognition of a design problem [9,25,33]. A designer determines the problem according to his or her parameter(s) of interest. A parameter of interest corresponds to a designer's judgment and includes the criteria that will drive the design. After analyzing a problem, the designer conceives of a solution or family of solutions that will correct or improve the current situation.

Following Smith and Browne [31], design problems consist of five elements: goals, constraints, alternatives, representations, and solutions. While goals comprise the specification of needs, solutions provide satisfaction of those goals. Designers normally generate various alternative approaches in order to solve the given problem. During evaluation, designers narrow the space of alternative designs [7,25]. The designer is required to make decisions based on many parameters and to choose among possible alternatives, while evaluating the feasibility of each choice.

Every effective design decision resolves some part of an unclear situation and reduces the number of possible alternatives. In the face of uncertainty, a designer is obliged to evolve a design so that if an artifact were to be produced according to that design, it would meet the requirements and satisfy the stated constraints.

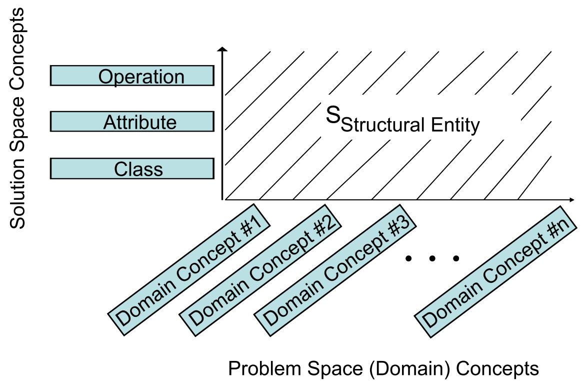


Fig. 1. Structural entity space.

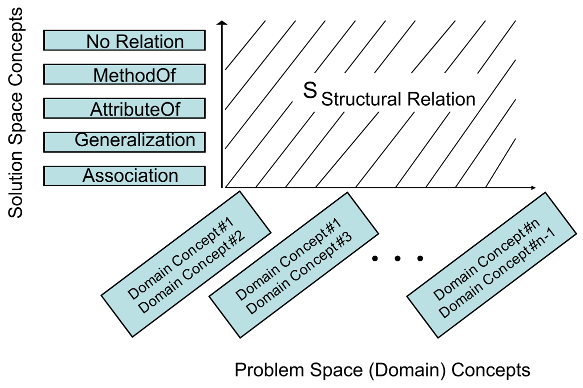

Solution Space Concepts
No Relation
MethodOf
AttributeOf
Generalization
Association
S Structural Relation
Domain Concept#1 Domain Concept#2
Domain Concept#1 Domain Concept#3
. . .
Domain Concept#n Domain Concept#n-1
Problem Space (Domain) Concepts

Fig. 2. Structural relation space.

4.1. Techniques used: Design spaces

Designers have to consider multiple alternatives during design and reach a decision based on experience and the methodology that they employ. In applications, requirements are usually imprecise and uncertain [13, 40]. Designers may eliminate some design alternatives in early stages with fuzzy methods [39], that may result in loss of information [21]. Therefore, designers need a consistent technique to represent, compare, and select among design alternatives. Design Space notion is defined as a function that maps "fundamental concepts" to design properties. Design properties include quality factors and implementation details that cover functional and non-functional requirements [3].

During the design process, designers work on two different spaces: Problem Space and Solution Space. Problem Space includes only the details from business/customer domain. On the other hand, Solution Space includes technical terms and incorporates solution details, while each space has its own representation [9,31]. One responsibility of a software designer is to transform problem space concepts into solution space concepts. Problem space concepts are terms, definitions, and rules from business/customer domain which are independent of technical details. On the other hand, solution space concepts are technical terms that incorporate solution details.

All possible design alternatives for these specifications form a design space for the software. To identify software abstractions, and corresponding decomposition activities, two different design spaces are defined. These are

– Structural Entity Space and
– Structural Relation Space.

Design space decomposition starts with the specification of design spaces which define entities, attributes, and relationships between the concepts. While entities within solution space are specified in an entity epace, relationships are given in a relation space.

4.1.1. Structural entity space

Mappings between structural problem domain concepts (C_{Domain}) into structural solution concepts are shown in this space. Figure 1 demonstrates the space as a two dimensional space. Following Aksit and Tekinerdogan [3], definitions of Structural Entity Space and corresponding solution space concepts are given as follows:

– The predefined property P_{Entity} (represented as the y-axis in Fig. 1) is a set of solution space alternatives for problem space concepts. $P_{Entity} = \{Class, Operation, Attribute\}$, and
– $S_{StructuralEntity}$ defines the design space that maps the concepts of C_{Domain} to the elements of P_{Entity} and as such represents the total set of alternatives of domain models.

4.1.2. Structural relation space

This space shows the relations between problem domain concepts in relational terms. Figure 2 demonstrates the two-dimensional space. Definitions of Structural Relation Space and corresponding solution space concepts are given as follows:

– The predefined property $P_{Relation}$ (represented as the y-axis in Fig. 2) is a set of alternatives for the relationships between concepts. $P_{Relation} = \{Association, AttributeOf, Generalization, MethodOf, NoRelation\}$, and

– $S_{StructuralRelation}$ defines a design space that maps the 2-tuple concepts of C_{Domain} to the elements of $P_{Relation}$ and as such represents the total set of relationship alternatives of domain models.

4.1.3. Design space example

Figure 3 presents the application of the design-space decomposition to a library example [3]. The example is the design of a set of collection classes, such as *LinkedList, OrderedCollection,* and *Array* to be a part of an object-oriented library. These classes should provide the needed operations to read and write the elements stored in collection objects. Furthermore, the sorting operation is needed to sort items within collection objects.

For the identification of the software abstractions, and the corresponding decomposition activities for the library example, two design spaces, *Entity Space, Relation Space,* are demonstrated as follow.

The structural model of the library example is composed of the concepts of the domain (C_{Domain}) and the relationships in the domain (R_{Domain}). They are listed below:

– $C_{Library} = \{Library, Collection, LinkedList, Array, OrderedCollection, collectionItems, sort, read, write\}$,
– $R_{Library} = \{(Library, Collection), (Library, LinkedList), (Library, OrderedCollection), (Library, Array), (Collection, LinkedList), (Collection, OrderedCollection), (Collection, Array), (sort, Collection), (sort, LinkedList), (sort, OrderedCollection), (sort, Array)\}$.

The two types of structural decompositions, entity design space and relation design space, are shown in Fig. 3. Designer decisions in entity and relation design spaces (represented in two dimensions) are marked in Fig. 3.

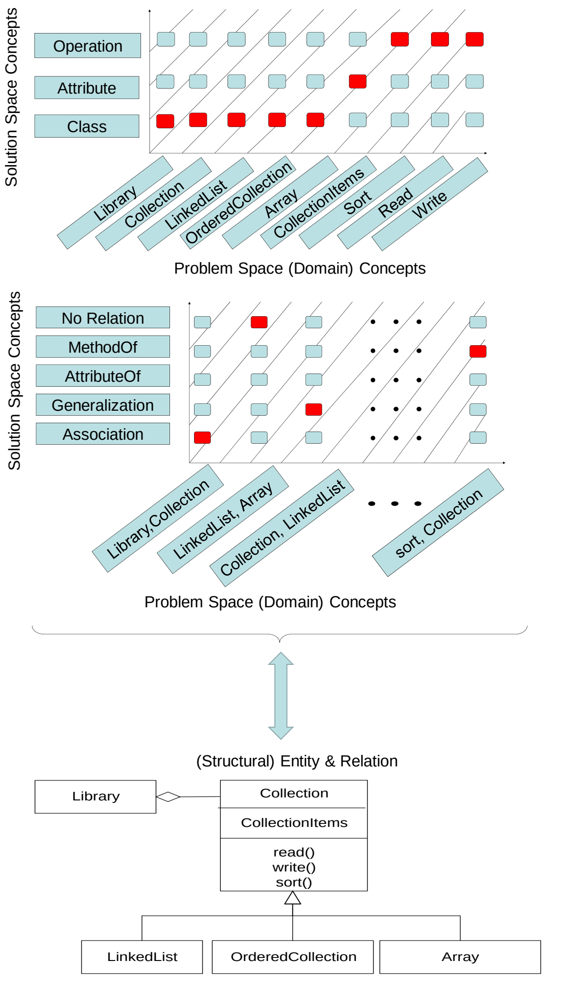


Fig. 3. An overall diagram of successive design decisions leading to the final product.

4.2. Design activity as an uncertainty-reduction process

Software developers generate various alternative approaches to decompose the given problem. Usually several decomposition alternatives exist and the designer must make decisions based on many parameters and to make choices among possible decomposition alternatives, while evaluating the feasibility of each choice. This situation reflects the *uncertainty* that designers encounter in finding a specific decomposition. Uncertainty exists in every step of software design, such as the clarification of requirements, mapping problem space concepts into solution space concepts, and transformation of solution space concepts into executable concepts.

Every design decision resolves some part of an unclear situation and reduces the number of possible alternatives. Thus, each design activity is an *uncertainty-reduction* process.

In the beginning there is minimal organization therefore high uncertainty exists (high entropy). The decisions carrying out design activities reduce uncertainty and introduce comparatively higher organization (low entropy). In Fig. 4, the design process is represented from the perspective of the developer to capture the uncertainty reduction process. All possibilities within the two spaces, in Fig. 4, demonstrate the uncertainty. On

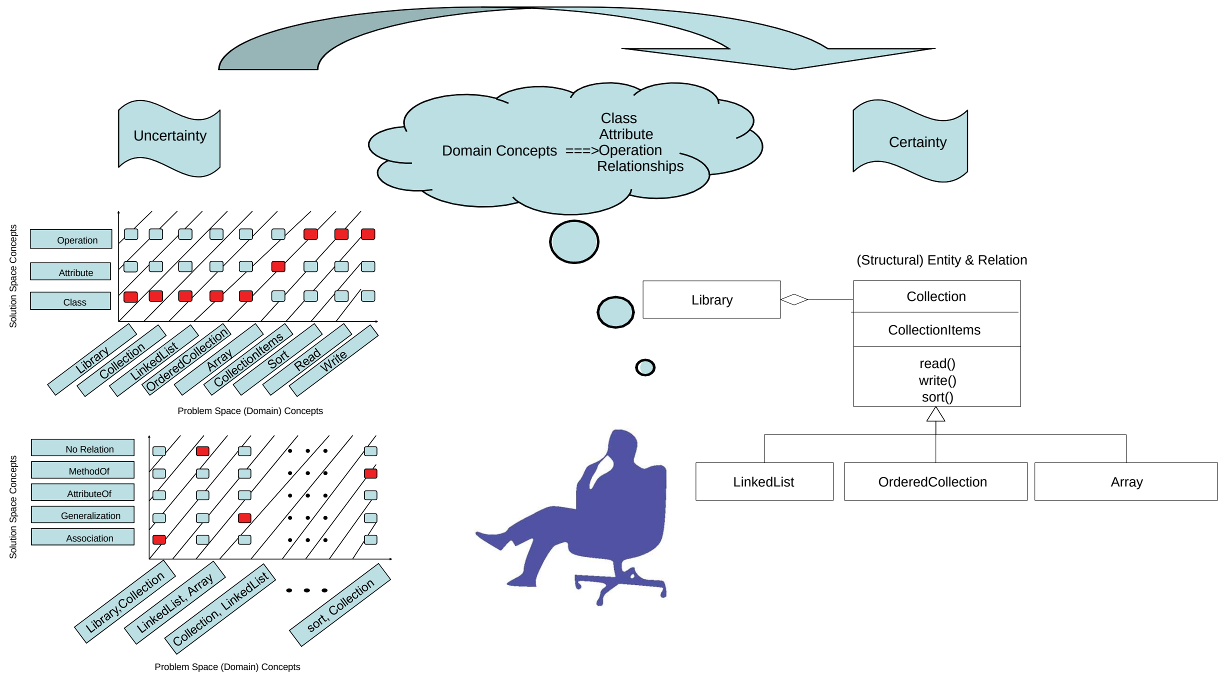


Fig. 4. Software design activities transforms uncertainty to certainty.

the other hand, the artifact displays organization (low entropy). The initial state of the library design spaces represent all possible alternatives with minimal organization (implying high entropy). The successive design decisions, as marked within design spaces of the library example, introduce comparatively higher organization (implying low entropy). Representing software design as an uncertainty reduction process is one of the novel contributions of this work.

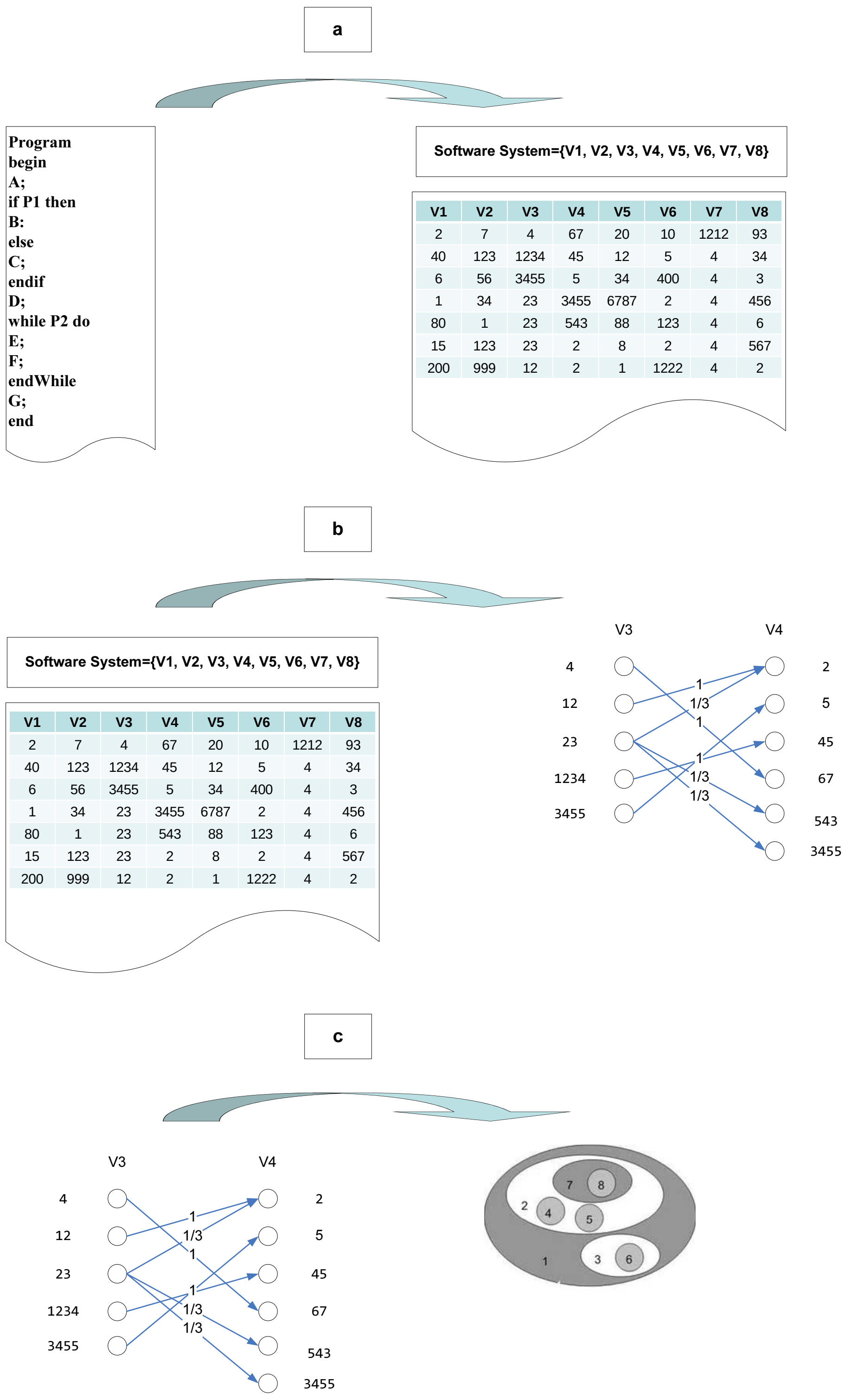
5. Information theoretical representation of software design

In our modeling approach, multivariate correlations among variables are modeled using communication channel formalism [10,23]. Total correlation over the complex system is the sum of the total correlation within the subsystems plus the correlations among the subsystems. Furthermore, each subsystem can be broken down into further subsystems and the fundamental rule holds in turn for the subsubsystems and their correlations [8,10]. One of the basic criterion for evaluation of the decomposition is that the correlation among the subsystems be insignificant compared to the total correlation.

Figure 5 presents three transition steps for the analysis of software systems using communication-channel representation. We start with mapping of software systems to set-theoretical representations. Software systems are represented with an arbitrary number of variables. Each variable is observed once per subjective time increment. The representative values are shown as a table in Fig. 5. In the second transition, which is the mapping of set-theoretical representation to a channel formalism, we show the information transfer between variables and demonstrate the correlations among variables as communication channels. The third step, hierarchical decomposition, takes the channels as input and applies decomposition techniques to find subsystems.

5.1. *Set-theoretical representation of software design*

We start with mapping of software systems to set-theoretical representations. Software systems are represented with an arbitrary (but finite) number of variables. We define a set of K variables for a given software system. The variables represent elements, such as identifiers defined within programs, data values from data segment, function return values, and code segment addresses. Each variable is denoted by X_j where $1 \leqslant j \leqslant K$. Software system is a set of X_j, denoted by the set $S = \{X_1, \ldots, X_K\}$. $X_j's$ values are taken from the set $P_j = \{X_j^1, X_j^2, \ldots, X_j^{n_j}\}$. P_j is a finite set, and its elements depends on the software elements


a
Software System={V1, V2, V3, V4, V5, V6, V7, V8}
b
Software System={V1, V2, V3, V4, V5, V6, V7, V8}
V3
V4
4
12
23
1234
3455
2
5
45
67
543
3455
1
1/3
1
1
1/3
1/3
c
V3
V4
4
12
23
1234
3455
2
5
45
67
543
3455
1
1/3
1
1
1/3
1/3
7
8
2
4
5
1
3
6

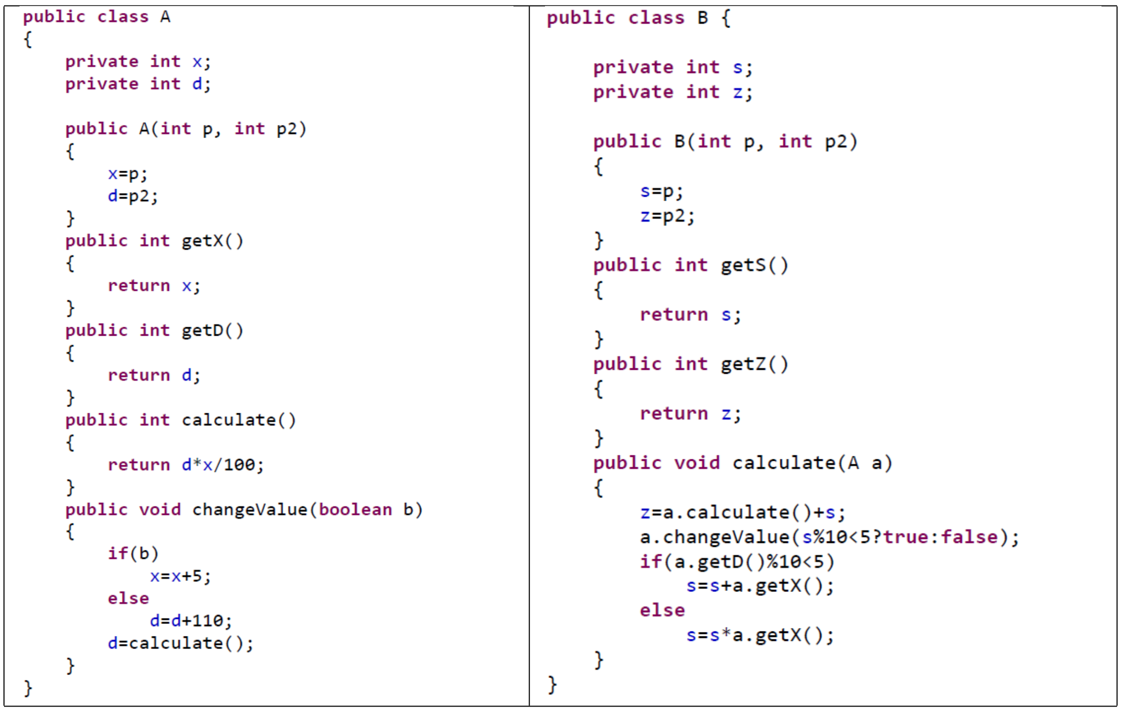
```
Program
begin
A;
if P1 then
B:
else
C;
endif
D;
while P2 do
E;
F;
endWhile
G;
end
```

V1	V2	V3	V4	V5	V6	V7	V8
2	7	4	67	20	10	1212	93
40	123	1234	45	12	5	4	34
6	56	3455	5	34	400	4	3
1	34	23	3455	6787	2	4	456
80	1	23	543	88	123	4	6
15	123	23	2	8	2	4	567
200	999	12	2	1	1222	4	2

V1	V2	V3	V4	V5	V6	V7	V8
2	7	4	67	20	10	1212	93
40	123	1234	45	12	5	4	34
6	56	3455	5	34	400	4	3
1	34	23	3455	6787	2	4	456
80	1	23	543	88	123	4	6
15	123	23	2	8	2	4	567
200	999	12	2	1	1222	4	2

Fig. 5. Analysis of software systems: a) mapping of software systems to set-theoretical representations, b) mapping of set-theoretical representation to channel formalism, c) hierarchical decomposition

Table 1
Example software system



```
public class A
{
    private int x;
    private int d;

    public A(int p, int p2)
    {
        x=p;
        d=p2;
    }
    public int getX()
    {
        return x;
    }
    public int getD()
    {
        return d;
    }
    public int calculate()
    {
        return d*x/100;
    }
    public void changeValue(boolean b)
    {
        if(b)
            x=x+5;
        else
            d=d+110;
        d=calculate();
    }
}
```

```
public class B {

    private int s;
    private int z;

    public B(int p, int p2)
    {
        s=p;
        z=p2;
    }
    public int getS()
    {
        return s;
    }
    public int getZ()
    {
        return z;
    }
    public void calculate(A a)
    {
        z=a.calculate()+s;
        a.changeValue(s%10<5?true:false);
        if(a.getD()%10<5)
            s=s+a.getX();
        else
            s=s*a.getX();
    }
}
```

which X_j represents. The set P_j forms a partition associated with variable X_j.

For example, when an *integer* type identifier defined within a given program is associated with X_1, P_1 takes a set of valid values associated with this integer type. Considering another level of abstraction, we can see that for an integer type with n bits, *unsigned type* represents the non-negative values 0 through $2^n - 1$, so that $P_1 = \{0, \ldots, 2^n - 1\}$, on the other hand, *signed integer type* represents numbers from $-2^{(n-1)}$ through $2^{(n-1)} - 1$, therefore the set P becomes $P_1 = \{-2^{(n-1)}, \ldots, 0, \ldots, 2^{(n-1)} - 1\}$.

The next step is the observation of values associated with variables of the system. Therefore, values associated with each variable are obtained and observed once per cycle. The cycle represents the stable states within a software system. The cycle should allow the variables a chance of changing values so the stable states of software system can be observed. In terms of the communication channel, this maps input variables of a channel to output variables. We observe the K variables for N cycles, and obtain a total of $K \cdot N$ different values. Therefore, observed values for each variable is denoted by $O_j = \{O_j^1, \ldots, O_j^N\}$. Observed number of occurrences of the event in consideration $X_j = X_j^i$ is denoted by $n_{X_j^i}$, such that $\sum_{i=1}^{n_j} n_{X_j^i} = N$. Number of occurrences associated with partition P_j is be denoted by $F_j = [n_{X_j^1}, \ldots, n_{X_j^{n_j}}]$. The variables X_j is grouped into sets to demonstrate decomposition steps during software design. The set $S_i = \{S_i^1, \ldots, S_i^{n_i}\}$ represents a subsystem of given software system, where $\cup_{i=1}^{r} S_i = S$ and $S_i \cap S_j = \emptyset$ for all $i \neq j$.

5.2. *Mapping of software design to communication channel formalism*

As presented in the above section, set-theoretical representation of a software system and the corresponding observed values reveal that there are varieties in observed values. The set theoretical decomposition of the software system leads us to expose the relationships between variables and capture the relationships in the formalism of communication channel.

To represent the interaction between two system variables for example, X_i and X_j:

- the value set, P_i, which is associated with X_i, is taken as a channel input set S,
- the value set, P_j, which is associated with X_j, is taken as a channel output set R, and
- then channel probability is, $P(S_k, R_l) = \frac{n_{S_k R_l}}{n_{R_l}}$, where observed number of occurrences of the

event in consideration$\{R_l = P^l_j\}$ is denoted by n_{R_l}, the number of occurrences of the event $\{S_kR_l = P^k_iP^l_j\}$ is denoted by $n_{S_kR_l}$.

5.3. *Hierarchical decomposition leading to higher organization*

Hierarchical decomposition, through a process partitioning interactions in a channel, produces subsets of channel elements. In Fig. 5, this process is represented as a transformation between channels and set-subsets producing hierarchical combinations of software elements.

Following the notation introduced above, software design decomposes the given software system $S = \{X_1, \ldots, X_K\}$ into r elements, such that the variables X_j is grouped into sets, $S_i = \{S^1_i, \ldots, S^{n_i}_i\}$ which represents an element of a given software system, where $\cup^r_{i=1}S_i = S$ and $S_i \cap S_j = \emptyset$ for all $i \neq j$.

The total interaction is decomposed into transmission such that

$$C_{Total}(X_1X_2\ldots X_K) = \sum_{i=1}^{r} C_{Total}(S_i) + C(S_1, S_2, \ldots, S_r) \quad (5)$$

where $C_{Total}(S_i)$ is the transmission within an element, S_i, and $C(S_1, S_2, \ldots, S_r)$, *correlation formula*. As a result, software system is decomposed into r elements with the total amount of transmission $C(S_1S_2\ldots S_r)$.

5.4. *Application to object-oriented software design*

In this section, we introduce our channel representation of software design using three stages elaborated in Sections 5.1, 5.2, and 5.3. This example, through the use of hierarchical organization, demonstrates the utilization of communication channel in software design.

Table 1 shows the specification of the representative example, with which we demonstrate information theoretical analysis of software design. As stated in Section 5.1, we start with mapping of the software system to a set-theoretical representation. We define a set of four variables, $\{V1, V2, V3, V4\}$, for the software system. In this case, variables represent *Attributes* defined within *Class A* and *Class B*. Mapping between Class *Attributes* and set variables is given in Table 2. The example software system is represented as a set, $S = \{V1, V2, V3, V4\}$. *Class* definitions demonstrate the hierarchy between *Classes and Attributes*. *Class A* is mapped into a set $\{V1, V2\}$ and *Class B* is mapped into a set $\{V3, V4\}$.

The next step is the observation of values associated with variables of the system. For this system, values associated with each variable are observed with the execution of the *calculate* method shown in Table 1. Each *calculate* execution changes values of variables so that each transformation of the system is observed. Representative observations of four variables for nine cycles are given in Table 3.

As stated in Section 5.2, the interaction between variables in the example software system is represented using the communication channel formalism. In terms of mapping to the channel representation, the relationship between $V1$ and $V2$ based on observed values within nine cycles is shown in Fig. 6.

Table 2
Software concepts to set concepts

Software	Set
Example system	$S = \{V1, V2, V3, V4\}$
A.x	$V1$
A.d	$V2$
B.s	$V3$
B.z	$V4$

Table 3
Part of the observed values

Cycle #	V1	V2	V3	V4
1	6	0	7	1
2	6	6	42	7
3	11	0	53	42
4	16	0	69	53
5	16	17	1104	69
6	21	3	1125	1106
7	21	23	1146	1125
8	21	27	24066	1150
9	21	28	505386	24071

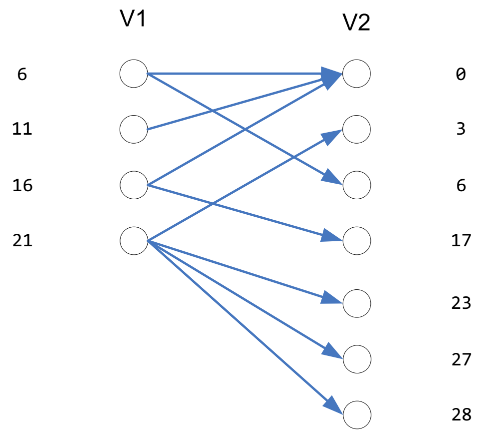


Fig. 6. Actual communication between V1 and V2.

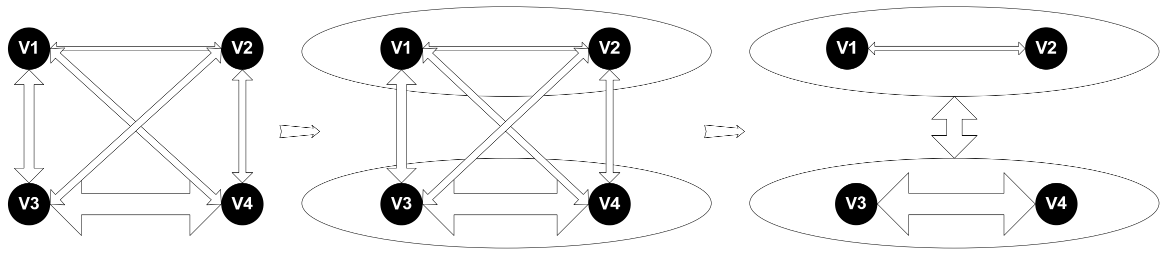


Fig. 7. Decomposition of interaction among variables.

Table 4
Transmission between variables

Variables	Transmission
$V1 \Leftrightarrow V2$	2.81
$V1 \Leftrightarrow V3$	4.28
$V1 \Leftrightarrow V4$	3.91
$V2 \Leftrightarrow V3$	3.86
$V2 \Leftrightarrow V4$	3.68
$V3 \Leftrightarrow V4$	7.56

As shown in Table 3, while $V1 = 6$ in the first and second cycles, $V2 = 0$ for the first cycle and $V2 = 6$ for the second cycle. This relationship is shown in the communication channel of Fig. 6 as two communication links starting from the first node of $V1$ and ending in the first and third nodes of $V2$.

Transmission between variables is calculated using Formula 2. As an example, calculation of transmission between $V1$ and $V2$ with the observed values in Table 3 is shown below.

$$O_{V1} = \{6, 6, 11, 16, 16, 21, 21, 21, 21\}$$

$$F_{V1} = \{2, 1, 2, 4\}$$

$$H(V1) = -\frac{2}{9}\log\frac{2}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{2}{9}\log\frac{2}{9} - \frac{4}{9}\log\frac{4}{9} = 1.83$$

$$O_{V2} = \{0, 6, 0, 0, 17, 3, 23, 27, 28\}$$

$$F_{V2} = \{3, 1, 1, 1, 1, 1, 1\}$$

$$H(V2) = -\frac{3}{9}\log\frac{3}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} = 2.64$$

$$O_{V1\cdot V2} = \{6\cdot 0, 6\cdot 6, 11\cdot 0, 16\cdot 0, 16\cdot 17, 21\cdot 3, 21\cdot 23, 21\cdot 27, 21\cdot 28\}$$

$$F_{V1} = \{1, 1, 1, 1, 1, 1, 1, 1, 1\}$$

Table 5
Decomposition of interaction among variables and subsystems

Decomposition	Transmission among elements	Transmission within elements
$\{V1, V2, V3, V4\}$	16.71	16.71
$\{\{V1\}, \{V2\}, \{V3, V4\}\}$	9.15	0+0+7.56
$\{\{V2\}, \{V1, V3, V4\}\}$	4.27	0+12.44
$\{\{V1, V2\}, \{V3, V4\}\}$	6.34	2.81+7.56

$$H(V1\cdot V2) = -\frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} - \frac{1}{9}\log\frac{1}{9} = 3.16$$

$$T(V_1 : V_2) = 1.83 + 2.64 - 3.16 = 1.31$$

A complete description of this software system is given elsewhere [10]. Pairwise relationships and corresponding transmission values for the example system with the complete description are shown in Table 4.

Diagrammatic representation of six pairwise relations are shown in Fig. 7. In Fig. 7, pairwise relations are indicated by arrows whose thickness is directly proportional to transmission values. For example, transmission value between $V3$ and $V4$ is 7.56 in Table 4, and it is shown as the strongest pairwise relationships in Fig. 7 with the thickest arrow.

As stated in Section 5.3, partitioning of these interactions creates the hierarchy within software systems. Groups of highly interacted elements constitute the subsystems of the system, as such producing the desired hierarchy. Following these principles, hierarchical decomposition of the example system is shown diagrammatically in Fig. 7. $V1$ and $V2$ are grouped into one subsystem and $V3$ and $V4$ are grouped into another subsystem.

Communication between variables within the example software system can be decomposed in many different ways. Table 5 shows various decomposition possibilities for the example system. For example, the decomposition $S = \{\{V2\}, \{V1, V3, V4\}\}$ partitions the example system into two subsystems. First subsystem, $\{V2\}$, consists of one software element with no internal communication. Second subsystem $\{V1, V2, V3\}$ is composed of three elements with a total of 12.44 transmission value among the variables, $V1, V3$, and $V4$. Transmission value between the subsystem $\{V2\}$ and the subsystem $\{V1, V3, V4\}$ is 4.27.

These values represent various decomposition possibilities with a requirement for decomposing the system into loosely coupled subsystems or elements.

6. Summary

Our key goal was to model software design and understanding of the design process in software engineering based on first principle foundations of science and the practices of "hard" engineering disciplines. We achieved this goal with formal demonstration that:

– Software design is a hierarchical decomposition and addresses all steps from requirements to the final product, and
– Software design imposes an organization and reduces entropy through successive transformations.

We used the mathematical model of communication system to model communication or information exchange among software elements. For multivariate interactions, we applied multivariate information transmission. We utilized Ashby's technique to map a software system into a complex system and then we used the organization measurement technique defined by Rothstein and Watanabe. Finally, we used hierarchical system definition from Simon and Conant to demonstrate that software design is a hierarchical decomposition of complex system.

The communication-channel representation of software systems opens up further useful possibilities for applying engineering analysis to software development. This indicates that current understanding and informal representations of software design, using our results, can further evolve into a type of inquiry involving classical engineering mathematics and concepts. It should be noted that we are initiating a completely new formal modeling approach for software design. With this approach eventually, deductive reasoning about large software would be possible as early as design phase. The mathematical machinery used for this purpose is Communication Channel Formalism of Shannon, which has not been used before in deductive reasoning about software. All other types of mathematical machinery for software modeling have been reviewed by one of the authors elsewhere [35].

We observe that, among the software metrics, our approach has some affinity with run time quality metrics such as run-time object-oriented cohesion metrics [22,24]. Run-time metric related aspects of our approach have been detailed in [28]. Although our intention here is not to develop a new metric, our work can be exploited in the direction of developing simple information theory based cohesion metrics.

Acknowledgments

We thank to anonymous referees for their suggestions and recommendations which greatly improved the presentation of the paper. More specifically, we are thankful to one of the reviewers to direct us to papers that we have missed in the overview. Another reviewer pointed the possibility of using our approach as a basis for a simple run-time cohesion metric and pointed us to two critical papers in this area. We are also thankful to Professor Tanju for discussions, reviews, and critical comments.

References

[1] H. Adeli and W. Kao, Object-oriented blackboard models for integrated design of steel structures, *Computers and Structures* **61**(3) (1996), 545–561.
[2] H. Adeli and G. Yu, An integrated computing environment for solution of complex engineering problems using the object-oriented programming paradigm and a blackboard architecture, *Computers and Structures* **54**(2) (1995), 255–265.
[3] M. Aksit and B. Tekinerdogan, *Software Architectures and Component Technology*, volume 648 of *The Kluwer International Series in Engineering and Computer Science*, chapter Deriving Design Alternatives Based on Quality Factors Software Architectures and Component Technology, pp. 225–257. Springer US, 2002.
[4] W.R. Ashby, Measuring the internal informational exchange in a system, *Cybernetica* **1**(1) (1965), 5–22.
[5] C.G. Bell and A. Newell, The pms and isp descriptive systems for computer structures. In *In Proceedings of the Spring Joint Computer Conference*, AFIPS Press, 1970, pp. 351–374.
[6] G. Booch, R.A. Maksimchuk, M.W. Engle, B.J. Young, J. Conallen and K.A. Houston, *Object-Oriented Analysis and Design with Applications*, The Addison-Wesley Object Technology Series. Addison-Wesley, 3rd edition, 2007.
[7] D. Braha and O.Z. Maimon, *A Mathematical Theory of Design: Foundations, Algorithms, and Applications*, Kluwer, Boston, 1998.
[8] R.C. Conant, Detecting subsystems of a complex system, *IEEE Transactions on Systems, Man and Cybernetics* **2**(4) (1972), 550–553.

[9] S. Dasgupta, *Design Theory and Computer Science: Processes and Methodology of Computer Systems Design*, Cambridge University Press, 1991.
[10] Z. Demirezen, *An information theory based representation of software systems and design*, PhD thesis, University of Alabama at Birmingham, 2012.
[11] Z. Demirezen, B.R. Bryant, A. Skjellum and M.M. Tanik, Design space analysis in model-driven engineering, *Journal of Integrated Design and Process Science* **14**(1) (2010), 1–15.
[12] E. Fermi, *Thermodynamics*, Dover Publications, New York, 1956.
[13] A.J. Fougeres and E. Ostrosi, Fuzzy agent-based approach for consensual design synthesis in product configuration, *Integrated Computer-Aided Engineering* **20**(3) (2013), 259–274.
[14] C. Ghezzi, M. Jazayeri and D. Mandrioli, *Fundamentals of Software Engineering*, Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 2003.
[15] J. Gleick, *The Information: A History, A Theory, A Flood*, Pantheon Books, New York, 1st edition, 2011.
[16] S. Hung and H. Adeli, Object-oriented back propagation and its application to structural design, *Neurocomputing* **6**(1) (1994), 45–55.
[17] W. Kao and H. Adeli, Multitasking object-oriented blackboard model for design of large space structures, *Engineering Intelligent Systems* **10**(1) (2002), 3–8.
[18] A. Karim and H. Adeli, Object-oriented information model for construction project management, *Journal of Construction Engineering and Management* **125**(5) (1999), 361–367.
[19] G. Kiczales, Aspect oriented programming, *ACM SIGPLAN notices: A monthly publication of the Special Interest Group on Programming Languages* **32**(10) (1997), 162.
[20] B. Liskov and J. Guttag, *Abstraction and Specification in Program Development*, MIT Press, 1986.
[21] F. Marcelloni and M. Aksit, Improving object-oriented methods by using fuzzy logic, *SIGAPP Applied Computing Review* **8**(2) (2000), 14–23.
[22] R. Mathur, K.J. Keen and L.H. Etzkorn, Towards a measure of object oriented runtime cohesion based on number of instance variable accesses, In *Proceedings of the 49th Annual Southeast Regional Conference*, ACM-SE '11, 2011, pp. 255–257.
[23] W.J. McGill, Multivariate information transmission, *Psychometrika Psychometrika* **19**(2) (1954), 97–116.
[24] A. Mitchell and J.F. Power, Using object-level run-time metrics to study coupling between objects, In *Proceedings of the 2005 ACM Symposium on Applied Computing*, SAC '05, 2005, pp. 1456–1462.
[25] G. Pahl and W. Beitz, *Engineering Design: A Systematic Approach*, Springer, 1996.
[26] J. Rothstein, Organization and entropy, *Journal of Applied Physics* **23**(11) (1952), 1281–1282.
[27] F.A. Salustri and R.D. Venter, An axiomatic theory of engineering design information, *Engineering with Computers* **8**(4) (1992), 197–211.
[28] R. Seker and M.M. Tanik, An information-theoretical framework for modeling component-based systems, *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* **34**(4) (2004), 475–484.
[29] C.E. Shannon, A mathematical theory of communication, *Bell System Technical Journal* **27**(3) (1948), 379–423.
[30] H.A. Simon, *The Sciences of the Artificial*, MIT Press, Cambridge, Massachusetts, 3rd edition, 1996.
[31] G.F. Smith and G.J. Browne, Conceptual foundations of design problem solving, *IEEE Transactions on Systems, Man, and Cybernetics* **23**(5) (1993), 1209–1219.
[32] W.P. Stevens, G.J. Myers and L.L. Constantine, Structured design, *IBM Systems* **13**(2) (1974), 115–139.
[33] N.P. Suh, *Axiomatic Design: Advances and Applications*, The MIT-Pappalardo Series in Mechanical Engineering. Oxford University Press, New York, 2001.
[34] N.P. Suh, *Complexity: theory and applications*, MIT-Pappalardo series in mechanical engineering. Oxford University Press, 2005.
[35] M.M. Tanik and E.S. Chan, *Fundamentals of Computing for Software Engineers*. Van Nostrand Reinhold, 1991.
[36] M.M. Tanik, A. Ertas and A.H. Dogru, Techniques in abstract design methods in engineering design development, *Control and Dynamic Systems* **61** (1994), 285–328.
[37] T. Tomiyama and H. Yoshikawa, *Design Theory for CAD*, chapter Extended General Design Theory, North-Holland, 1987, pp. 95–130.
[38] S. Watanabe, *Knowing and Guessing; A Quantitative Study of Inference and Information*. Wiley, New York, 1969.
[39] L. Yan and Z. Ma, Comparison of entity with fuzzy data types in fuzzy object-oriented databases, *Integrated Computer-Aided Engineering* **19**(2) (2012), 199–212.
[40] L. Yan and Z. Ma, Conceptual design of object-oriented databases for fuzzy engineering information modeling, *Integrated Computer-Aided Engineering* **20**(2) (2013), 183–197.
[41] Y. Zeng, Axiomatic theory of design modeling, *Transactions of the SDPS: Journal of Integrated Design and Process Science* **6**(3) (2002), 1–28.
[42] Y. Zeng and P. Gu, A science-based approach to product design theory part 1: Formulation and formalization of design process, *Robotics and Computer-Integrated Manufacturing* **15**(4) (1999), 331–339.