A Multiplicity Approach for Equilibrium-Driven Complexity Control

K. O. Chow
City University of Hong Kong
Hong Kong
cspchow@cityu.edu.hk

Abstract - This paper uses a software technology approach to tackle the problem of complexity control. A multiplicity of system models, abstraction levels and development primitives are employed, with special attention paid to the use of relationship in the tackling process. The paper explains the steps needed to locate the balance point between competing factors in the search of the final equilibrium state. The benefits include extracting the part and relationship ideas found in software techniques and applied into controlling complexity.

Keywords: Complexity control, equilibrium, balance, relationship, software method.

1 Introduction

Tackling complexity is an important study with a satisfactory solution remaining elusive. The issue of complexity exists in many subject areas and its study is multi-disciplinary [1, 2]. In the computer science discipline, complex computer systems abound after the proliferation of electronic computers in the past fifty years. Significant progresses have been made, although variation in success exists in different sub-areas such as the rapid hardware advance versus desired software productivity improvement.

Software study is an area that awaits significant progress in productivity, as in Brooks’s paper on ‘No Silver Bullets’ [3]. Two important ingredients in complexity discussion are parts and relationships. Software engineering has studied these two items extensively in an effort to develop quality software products through a sound development process. Examples are entity and relationship [4] and, object and relationship [5-9]. In tackling software complexity, the key concern is to control complexity. A number of key concepts evolved, like abstraction, decomposition and classification. These ideas share similarities with other disciplines, though differed in multi-modeling and relationships.

Due to the complicated intricacy and the subsequent effort to seek a better understanding through a systematic methodology, the study of complexity is often tied to system study, forming complex systems. In this paper, we adapt findings from computer science and take a system methodology approach to analyze, understand and solve the intertwining relationship in the task of complexity control. Important ideas include multi-modeling, abstraction hierarchy and development primitive. System modeling makes it easier to see connections, relationships and patterns of interaction, as multiple perspectives put things in better context. We will show that these ideas have generic attribute and can be applied to other disciplines.

This paper is organized as follows. Section 2 states the key concepts related to complexity control, equilibrium and balance. Section 3 discusses the software engineering ideas used in controlling complexity. Section 4 presents the proposed approach. The last section evaluates the proposal and gives the conclusion of the paper.

2 Complexity and Equilibrium

This section presents the key concepts relevant to the paper proposal, namely complexity control, balance and equilibrium.

2.1 Complex System

A complex system is commonly defined as a system with numerous parts. And many times these parts are arranged intricately. Although parts and intricacy have been the conditions to be investigated, they tend to emphasize on the structural arrangement of parts, with less attention given to interaction, transition and processing. A comprehensive consideration should include both structure and behavior, as behavior is affected by structure and both are closely related. Structure-based behavior includes object interaction, state transition and function processing.

2.2 Equilibrium and Balance

An important consideration in the study of complexity is system equilibrium. Equilibrium is a system condition in which competing forces are balanced. Balance point is the desirable point among opposing forces. For example, the balance point in decomposing a function is seven or less sub-functions, as we want to avoid ending too many sub-parts and become over-decomposition. The ability to maintain balance is important to achieving system equilibrium. System balance is a state condition enabled by interaction behavior and function processing, and is obtained by:

- Locate factors to be included in balance considerations
- Identify balance point
Act on factors to move closer to balance point

2.3 Complexity Control

Complexity control is meant to prevent from aggravating the degree of complexity in a task. A number of techniques have been evolved. The key idea centers in limiting the part number and consequently the scope of consideration. The techniques include:

- Classify into types of units
- Decompose into levels of units
- Compose units into groups

A number of control effects are resulted in the process of controlling complexity. They include:

- A leveled structure formed, with levels of parts
- The number of parts reduced, via composition into larger units
- Relationships between parts become explicit

These results can be viewed from a perspective of relationship and translated into having generalization, aggregation and composition relationships.

In sum, software development methodology is suitable to the complex system study. Because of the similar attributes exhibited, useful system properties include modeling, nesting, system structure, interaction behavior and conditional state. The followings are derived from this section:

- Include behavior into considerations
- Incorporate relationship and state condition
- Adopt the use of multiple techniques

3 Software Development Concepts

This section states the key concepts in the system methodology approach, with the aim to extract relevant ideas for an equilibrium-driven complexity control.

3.1 System Perspectives

Abstraction In computer science, an abstraction is a simplified view of a system which contains selected system details important for a particular purpose. In other disciplines, such as humanity and business, the concept of abstraction has also existed for a long time. Like stepwise refinement and information hiding, abstraction is an important concept in computer science. In software engineering, it is the issue of system complexity, equivalent to Brooks’s essential complexity, which motivates the search for more effective solution. Model-driven software development is the key idea in both the classical and modern structured and object-oriented methodologies, and is a central tool in complexity control. This importance is due to the raising of the abstraction level in software development. A weakness in abstraction is that it is not clear when should abstraction be stopped. This is so because the nature of abstraction implies that the process should be complete when essential features are obtained. However, the task of separating the essentials from non-essentials is a lengthy iterative process and necessitates continuous refinement. Other weakness includes imprecise abstraction.

Multiple Perspectives Abstraction is closely related to modeling. The essence of abstraction is to identify essential properties and ignore unnecessary details. In other words, abstraction involves selective attention and selective ignorance. This focus of concern is based on the modeling perspective. The historical progression of the software technology shows a transition from a serial execution emphasis to one of function processing and later towards oriented towards objects. These emphases reflect perspectives taken in understanding and modeling the system. The current trend in software development is to migrate from a single perspective to multiple perspectives, that is, model a system with different perspectives. Multi-modeling is the prevailing trend in current software development, as is shown in UML [10]. There are different proposals in terms of 2-model, 3-model and others. The more common one is a dual model, comprising structural model and behavioral model. Multiple models create a shortcoming of inconsistencies among system models. The inclusion of multiple models complicates the abstraction process. Other weaknesses are domain specificity and domain dependency.

3.2 Hierarchical Levels

When a system is decomposed into levels, a hierarchy of system is formed. These abstracted hierarchical levels can be grouped into high and low levels, and interpreted as shown in Figure 1 below:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>What, General, Abstract, More information hidden</td>
</tr>
<tr>
<td>Low</td>
<td>How, Detailed, Concrete, Less information hidden</td>
</tr>
</tbody>
</table>

Figure 1. Abstraction Level

Using the concept of levels, software method devises the techniques of decomposition and composition and uses them to organize parts. However, no relationships exist between the hierarchical levels. This applies to the structured approach as well as the object-oriented approach. The standardized UML incorporates meta-model, using a more generic abstraction. This has the benefit of domain-independent characteristic. But the weakness is that it still has to involve domain-specific data. On the other hand, traversing between levels may involve changing of views. This implies that we can unify views through traversal of abstraction levels.

In addition to the lack of relationships, there are three other
shortcomings. It is not clear what constitute high and low levels. Abstraction level is a relative term, with no absolute value. Inconsistencies exist between abstraction levels. In addition, inconsistencies may also exist between models and abstraction levels.

3.3 Relationships

Relationship is also an under-investigated area in complexity study. The inclusion of the relationship concept in software development is a relatively new idea in computer science and happens only with paradigm shifts. It is only with the coming of the data modeling with entity relationship [4] and the object modeling [5-9] that the concept of relationship is explicitly included. The preceding paradigm of structured approach only embodies the hierarchy concept and does not have any relationships.

Similar to abstraction, the concept of classification exists in many disciplines such as biology, economics. Classification uses the idea of commonality to group similar things into types, forming the a-kind-of relationship. At the same time, the hierarchical level concept is employed to produce an organized depiction of parts. In the design of a complex system, classification is used to control complexity by reducing the amount of details that a designer needs to consider in similar groups. To manage a system more abstractly, the idea of subsystem is used. A subsystem may recursively include other subsystems.

The object-oriented methods in the 90s usually prescribe the basic relationships of association, aggregation and inheritance. Currently, the UML 2 [10] includes the following relationships, namely aggregation, association, composition, dependency and generalization. Using the dual model concept, a more complete list of relationships is shown in Figure 2 below.

A weakness is the lack of relationships between the structural and behavioral models. This paper proposes to use relations to maintain balance in complex systems.

<table>
<thead>
<tr>
<th>Structural relationship</th>
<th>Aggregation, association, composition, generalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioral relationship</td>
<td>Sequence, dependency, aggregation, realization</td>
</tr>
</tbody>
</table>

Figure 2. List of Relationships

3.4 Process Primitives

It will be more effective in complexity control if activities are organized in terms of path and direction, together constituting basic activity primitives. A pattern of activities gives rise to a progression path. There are different patterns. They can be linear, fixed, dynamic or iterative. There are different directions of movement, such as top-down, bottom-up, middle-out. It is the combination of the development process stages and the process primitives that gives rise to a variety of process models. A full list of the primitives is given in Figure 3 below.

<table>
<thead>
<tr>
<th>Path</th>
<th>Sequence, iteration, linear, singular, parallel, fixed, dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Top-down, bottom-up, middle-out, forward, backward, inside-out, outside-in</td>
</tr>
</tbody>
</table>

Figure 3. Activity Primitive

4 Multiplicity for Equilibrium

This section links the ideas described in Sections 2 and 3 and explains the proposed equilibrium-driven complexity control.

4.1 Multiplicity Approach

As has been described in Section 3, the key idea is in the exhibition of a multiplicity characteristic. They become the building blocks, or parts, in complexity control. The key multiplicity elements are summarized follows:

- Multiple models
- Multiple hierarchical levels
- Multiple relationships
- Multiple paths and directions

4.2 Proposal

As has been described in Section 2, the key ideas to achieve equilibrium are to obtain the followings:

- Condition state
- Balance point
- Competing factor

We propose to map and unify these ideas to form the building blocks for the proposal. We use the software development methods as exemplary elements in the proposal. Figure 4a gives a mapping between the structural and behavioral models. Figure 4b describes the structure-dependent behavior model by organizing the dual models into two hierarchical levels.

<table>
<thead>
<tr>
<th>Structure</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class, relationship</td>
<td>Use case scenario</td>
</tr>
<tr>
<td>Property</td>
<td>Interaction sequence</td>
</tr>
<tr>
<td>Method</td>
<td>Activity processing</td>
</tr>
</tbody>
</table>

Fig 4a. Structural/Behavioral Models

The structural model is of primary importance in software development. Subsequently it takes a primary role, a major
share and a leading order. It will first give the earliest meaning. It also forms the lower level in a hierarchy of model dependency. The behavioral model is based on, and dependent upon, the structural model, serving as a guide in development direction and a dependency relationship. Figure 5 shows the mapping among equilibrium criteria, competing consideration factors and relationships employed.

<table>
<thead>
<tr>
<th>Equilibrium Criteria</th>
<th>Competing Factors</th>
<th>Relationship Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importance</td>
<td>Primary vs. secondary</td>
<td>Generalization, aggregation</td>
</tr>
<tr>
<td>Share/weight</td>
<td>Major vs. minor</td>
<td>Ordering sequence</td>
</tr>
<tr>
<td>Directional guide</td>
<td>Based on, driven by, oriented to</td>
<td>Dependency</td>
</tr>
</tbody>
</table>

Figure 5. Building Blocks

The competing factors are used as opposing forces to locate the balance point in searching for equilibrium state. For example, the balance strategy between primary and secondary factors will be primary first. The relationships of generalization and aggregation are employed as structure with leveled parts classified, decomposed and composed. The behavior includes interaction between parts, state transition and function processing. All these are based on the structural model built previously. Equilibrium is the final state where the primary and secondary factors are balanced. Complexity is controlled in the process.

### 4.3 Complexity Control Steps

From the relationships derived in previous section, the sequence of modeling is to first create a structural model, and then a behavioral model. This sequential order conforms to the recommendations given in most object-oriented methods. Figure 6 lists the procedural steps.

<table>
<thead>
<tr>
<th>Modeling</th>
<th>Extract essentials</th>
<th>Select emphases</th>
<th>Create models</th>
<th>Identify parts</th>
<th>Form hierarchies</th>
<th>Establish relations among parts</th>
<th>Relate dual models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>Sequence activities with primitives</td>
<td>Set condition states</td>
<td>Consider competing factors</td>
<td>Establish balance point for equilibrium</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Steps

### 5 Conclusions

We have presented an approach to tackle the question of complexity control. In our proposal, we advocate a multiplicity approach, encompassing multiple dimensions of system modeling and development activity. The paper explains the steps needed to locate the balance point between competing factors in the search for final equilibrium state. The benefits include extracting the part and relationship ideas found in software techniques and applied into controlling complexity. There are a number of outstanding issues. These include:

- Work towards domain-independent proposal by resolving the problem of domain specific and domain dependence
- Derive a detailed list of relationships
- Establish relationships between structural and behavioral models to remove inconsistencies, and relationships between hierarchical levels to ensure correct mapping

The proposal will also require realistic applications for further verification. Future works include incorporating modeling language and developing a software tool.

### 6 References