Entropy **2015**, *17*, 2094-2116; doi:10.3390/e17042094

OPEN ACCESS

entropy

ISSN 1099-4300

www.mdpi.com/journal/entropy

Article

Research and Measurement of Software Complexity Based on Wuli, Shili, Renli (WSR) and Information Entropy

Rong Jiang [1,2]

[1] School of Information, Yunnan University of Finance and Economics, Kunming 650221, China; E-Mail: jiangrong@ynu.edu.cn; Tel.: +86-0871-6587-6862

[2] School of Software, Yunnan University, Kunming, 650091, China

Academic Editor: J. A. Tenreiro Machado

Received: 30 December 2014 / Accepted: 1 April 2015 / Published: 8 April 2015

Abstract: Complexity is an important factor throughout the software life cycle. It is increasingly difficult to guarantee software quality, cost and development progress with the increase in complexity. Excessive complexity is one of the main reasons for the failure of software projects, so effective recognition, measurement and control of complexity becomes the key of project management. At first, this paper analyzes the current research situation of software complexity systematically and points out existing problems in current research. Then, it proposes a WSR framework of software complexity, which divides the complexity of software into three levels of Wuli (WL), Shili (SL) and Renli (RL), so that the staff in different roles may have a better understanding of complexity. Man is the main source of complexity, but the current research focuses on WL complexity, and the research of RL complexity is extremely scarce, so this paper emphasizes the research of RL complexity of software projects. This paper not only analyzes the composing factors of RL complexity, but also provides the definition of RL complexity. Moreover, it puts forward a quantitative measurement method of the complexity of personnel organization hierarchy and the complexity of personnel communication information based on information entropy first and analyzes and validates the scientificity and rationality of this measurement method through a large number of cases.

Keywords: software complexity; software project; WSR; information entropy

1. Introduction

American computer scientist Frederick Phillips Brooks, the winner of the Turing Award, known as Nobel Award in the field of computers, pointed out that complexity is one of the four essential issues of software and software project management in his paper *No Silver Bullet: Essence and Accidents of Software Engineering* [1]. Famous computer expert, Grady Booch, one of the founders of Unified Modeling Language (UML), believed that excessive complexity was one of the main reasons for the failure of software projects. Narciso Cerpa [2] had made a thorough investigation and research of 70 failed software projects in America, Australia, Chile and other countries and found that the complexity of 81.4% of the projects was underestimated.

Ludovic-Alexandre Vidal's [3,4] results show that project complexity results in damages or failures for the projects and project complexity is ever growing and needs to be understood, analyzed and measured better to assist modern project management. Williams [5] considered that one of the reasons for project failure would be the increasing complexity of projects. Marian Bosch-Rekveldt [6] claimed that one of the reasons for project failure would be an underestimation of the project complexity. Mendel Giezen [7] indicated that the reduction of complexity means that there are fewer unknowns and fewer variables to predict, and thus, the project and planning of the project arguably becomes more manageable. Marian Bosch-Rekveldt [6] and others stressed the importance of complexity in the current research of project management. Gina C. Green's [8] results show that the research of complexity is very important for successful project leadership.

It is evident that it is very necessary to research the complexity of software from the viewpoint of project management. Complexity is an important factor throughout the software life cycle, and it is directly related to software quality, cost and the production schedule and is inversely proportional to controllability. At the same time, it also brings a lot of hidden safety dangers: projects' ever growing complexity is an ever growing source of project risks [3,4]. To the extent that the complexity is beyond the control of the software project manager, the project's failure is inevitable. Deep research of complexity is the prerequisite for control, and the recognition, understanding and measurement of complexity is the basic work that must be done for research.

This paper studies software project management from the viewpoint of complexity, which is a challenging job and also a direction very worthy of further study. It will make people understand complexity more clearly in order to make complexity easier to control and to upgrade the software project management level. This paper proposes a WSR framework of software complexity, which divides the complexity of software into three levels of Wuli (WL), Shili (SL) and Renli (RL). Because the current research focuses on WL complexity, but project management is most in need of the results of RL complexity, this paper presents a measurement method of RL complexity based on the information entropy theory.

The remaining parts of this paper are organized as follows. Section 2 describes the complexity definition and the software complexity research status. Section 3 presents a WSR framework of software complexity. Section 4 explains software project RL complexity and proposes the complexity metrics. In Section 5, case analyses are presented, to show the applicability of the approach. Finally, Section 6 outlines the summary and contributions of this study.

2. Software Complexity: A Literature Review

2.1. The Definition

Complexity is the reciprocal of simplicity. It is increasingly difficult to understand, manage and control with the increase in complexity, which is people's common understanding of complexity. Complexity was proposed by Ludwig Von Bertalanffy, a system science pioneer and American scientist in Austria, in the 1940s. He foresaw that the nature of system science lies in the research of complexity. In the 1990s, the famous Chinese scholar Qian Xuesen [9] said: "those that can't or shouldn't be solved by the reductionism method but new scientific methods are complexity problems". Ludovic-Alexandre Vidal *et al.* [3,4] stated: "Project complexity is the property of a project which makes it difficult to understand, foresee and keep under control its overall behavior, even when given reasonably complete information about the project system."

Turing Award winner Brooks said [10]: "the meaning of complexity is confusing and complicated. Computer software is the most complex entity of all artificial products". Software complexity is a sub-problem of complexity research. Zuse [11] defines it as: "the true meaning of software complexity is to analyze, implement, test, maintain, change and understand the difficulty of software". Lin [12] and others believe that the complexity of software development mainly comes from the complexity of the development system and the complexity of the implementation of the development process within the framework of the development system, and man is the major source of complexity.

2.2. Existing Software Complexity

The current research [13] of software complexity is mainly involved in the fields as shown in Figure 1.

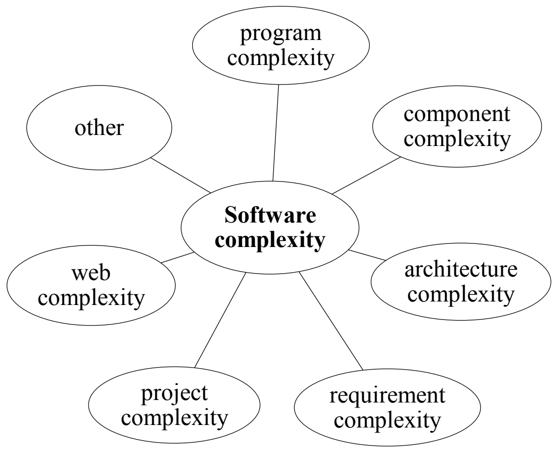

Figure 1. A framework of software complexity.

- Program Complexity

Program complexity normally refers to an abstraction of the software scale, software development difficulty and the quantity of possible hidden errors in software. In Figure 1, the measurement of program complexity has the earliest research history and is fruitful. It can be divided into process-oriented structured measurement of program complexity and object-oriented program complexity. Classical measurement methods are shown in Table 1.

Table 1. Several classical measurement methods of program complexity.

Category	Name	Introduction
Complexity Measurement of Structured Program	Line of Code (LOC) Measurement	With the total number of LOC of a program as a measure of program complexity, it is generally used to estimate the workload for program development. Generally speaking, the program is increasingly complex with the increase in the number of LOC.
	Program Control Structure Complexity Measurement (namely the McCabe Measurement Method [14])	This is a complexity measurement model based on the topological structure of a program, which measures the program complexity by calculating the number of linearly-independent directed cycles in the strongly-connected program chart.
	Program Text Complexity Measurement (namely the Halstead Measurement Method [15])	This measures the program complexity by calculating the number of operators in the program.
	Harrison Complexity Measurement [16]	This is a measurement method of program complexity based on decomposing of the flow graph into ranks. It can determine the nesting levels of nodes in the flow chart.
Complexity Measurement of Object-oriented Program	Chidamber & Kemerer (C&K) Measurement [17,18]	This is a measurement method based on the inheritance tree and composed of six measurement standards: complexity of all methods of each class (WMC), depth of inheritance (DIT), number of children (NOC), number of responses of each class (RFC), coupling degree between objects (CBO) and lack of cohesion in method (LCOM).
	Metric for Object-Oriented Design (MOOD) Measurement [18]	Six measurement indicators are given from four aspects of encapsulation, inheritance, coupling and polymorphism, namely method hiding factor (MHF), attribute hiding factor (AHF), method-inherited factor (MIF), attribute-inherited factor (AIF), coupling factor (CF) and polymorphism factor (PF).
	Chen and Liu Measurement [18,19]	This measures from eight aspects of operation complexity, operation parameter complexity, attribute complexity, operation coupling factor, class inheritance, cohesion, class coupling and reusability.

In addition, other research on program complexity are fruitful. For example, reference [20] introduces a computation method for structured program complexity based on entropy evaluation of random single-value response functions and characteristic software functions. Based on class and object, reference [21] introduces two kinds of object-oriented measurement methods of software space complexity.

- Component Complexity

In traditional research on the complexity of structured software, more attention is paid to the internal details of the module, but component software pays more attention to the interaction relationship between components. The work in [22] proposes a measurement model of component software complexity based on the dependency matrix.

- Architecture Complexity

Too complex of an architecture may cause low comprehensibility and maintainability, but too simple of an architecture may cause reduced reliability and safety. The work in [23] proposes an approach to evaluate the complexity of software architecture. The work in [24] measures the visual complexity of the software architecture by the use of an algebraic expression. The work in [25] studies software complexity metrics based on complex networks.

- Requirement Complexity

Requirement analysis is a very important link in the process of software development. If requirement analysis fails, all may fail. Based on requirement statement templates, [26] puts forward a quantitative measurement method of complexity during the phase of software requirements. Based on quantitative cases, [27] studies the complexity of cases.

- Software Project Complexity

The complexity of software projects is a very important indicator in software project management. The work in [28] proposes some evaluation models and methods for the complexity of software projects based on evidence reasoning.

- Web Application Complexity

The work in [29] proposes entropy-based complexity measures, WCOXIN and WCOXOUT, for web applications. The work in [30] proposes a complexity measure for web application, WCOX, which is defined by entropy theory, and a web model extracted by static analysis.

- Other

The work in [12] analyzes the causes of the complexity of the software development system and its development processes from the perspective of complexity science. It believes that the software development complexity mainly comes from the development system complexity and the development process implementation complexity within the framework of development system, and man is the main source of complexity. The work in [31] designs a new model of the development prototype and related measurement methods of software complexity to study the synthetic complexity in software maintenance environment. The work in [8] shows that cognitive complexity is very important for the performance of project leadership.

2.3. Limits of Existing Software Complexity

- Misunderstood Definition of Software Complexity

In the early stage, software development almost just involves programming, so many scholars think that software complexity is identical to program complexity, the idea of which is carried by numerous scholars and literature works.

- Ambiguous Research Contents

The research on software complexity has a history of more than 40 years, but what aspects does software complexity cover? There is little literature to answer this question, but the author finds that only [12] is related to this question, and it summaries software development complexity as structural complexity, boundary complexity, evolution complexity, concept diversity, limited knowledge and multiple contingency.

- Plentiful Abundant "Micro" Research and Insufficient "Macro" Research

The research on program complexity is most mature and fruitful. It concerns microcosmic program complexity, namely it remains in the interior microscopic properties of the program and lacks overall research on the complexity. Compared with the program, software components, software architecture and requirement analysis and the software project are more macroscopic. Research on these aspects is not fruitful and has not formed a relatively complete theory system.

- Much Research on the Complexity of Technical Factors and Little Research on the Complexity of Non-technical Factors (Especially Personnel Complexity)

In the industry, there is a saying that non-technical factors often more easily lead to the failure of software projects. The software is composed of programs and documents, and it involves many non-technical factors, because of document complexity, so there is not much research. Besides, bad organization structure and management are also important non-technical factors for the failure of software projects, but few scholars study the complexity from this aspect.

3. A WSR Framework of Software Complexity

Wuli-Shili-Renli systems approach [32,33] (WSR methodology), put forward by Jifa Gu, Zhichang Zhu and other scholars at the University of Hull in 1994, is the outcome of the collective efforts between many Chinese and British researchers since the mid-1990s; is a kind of oriental system methodology with Chinese traditional philosophical speculations; and its basic contents are shown in Table 2.

Table 2. Contents of the Wuli-Shili-Renli system.

	Wuli	**Shili**	**Renli**
Object and Content	Objective material world rules, rules.	Organization, system management and work principles.	People, groups, relationships, truth of doing things.
Focuses	What is the focus? Functional analysis.	How to do? Logic analysis.	What is the best thing to do? Maybe? Humanistic analysis.
Principles	Honesty; pursuit of truth.	Coordination; pursuit of efficiency.	Humanity, harmony; pursuit of effectiveness.
Required Knowledge	Natural science.	Management science, systems science.	Humanistic knowledge, behavioral science and psychology.

"Wuli" refers to the mechanism involving the motion of matter, and it usually uses the knowledge of natural science to mainly answer what the "matter" is and to solve the problem of "what it is". "Shili" refers to work principles to mainly solve the problem of how to do. It usually uses the knowledge of engineering, operations research and other aspects to answer "how to do" and to solve the problem of "how to do" mainly. "Renli" refers to the principles of conducting one's self, including organizational and interpersonal relationships. It usually uses the knowledge of humanities and social science to answer "how to do" and to solve the problem of "what is the best thing to do".

System methodology is a set of working procedures, methods, tools and techniques to solve complex system problems under the guidance of a certain system philosophy. As a kind of methodology and thinking, WSR's core lies in dealing with complex problems in consideration of both WL and how to apply WL in SL better; besides, because the understanding and solving of problems and the implementation of management decisions cannot be separated from man, as a system, WSR achieves the knowledge of WL, familiarity of SL and mastery of RL to carry on a systematical, complete and hierarchical research on complex problems.

Software is a kind of invisible logical entity and a pure product of human wisdom, and the invisibility and uncertainty of its production process is higher than that of traditional industries (such as construction and manufacturing), which causes its high complexity. In this paper, software complexity is divided into WL complexity, SL complexity and RL complexity.

WL [34] refers to the objective existence of a project or problem that people face in the treatment process. It is the sum of laws of matter and motion, a symbol of the objective existence of ontology and is independent of man's will. The scientific problems of software complexity belong to this category, for example algorithm complexity, program complexity and software science problems, all of which give expression to WL in WSR theory. SL [34] refers to the mechanisms involved in the objective existence and laws of a project or problem that people face in the treatment process and the effective methods that help people to deal with affairs based on the world and objectively existing mechanisms, namely a kind of "man-matter" interface, including all software technical problems. RL [34] refers to all organizations involved in the treatment process of a project or problem and mutual relationships between people and their change processes, that is, a kind of "man-man" interface, including personnel problems. Software engineering includes engineering technology and engineering management, the former belonging to SL and the latter belonging to RL. Software development tools, requirement analysis technology, software design, component complexity and architecture complexity belong to technological problems in the field of software engineering and are included in SL. In software engineering management, personnel organization, human resource management and personnel communication belongs to RL, so research and straightening out of RL relationships can stimulate people to realize the predetermined target of a project or problem in accordance with acceptable SL.

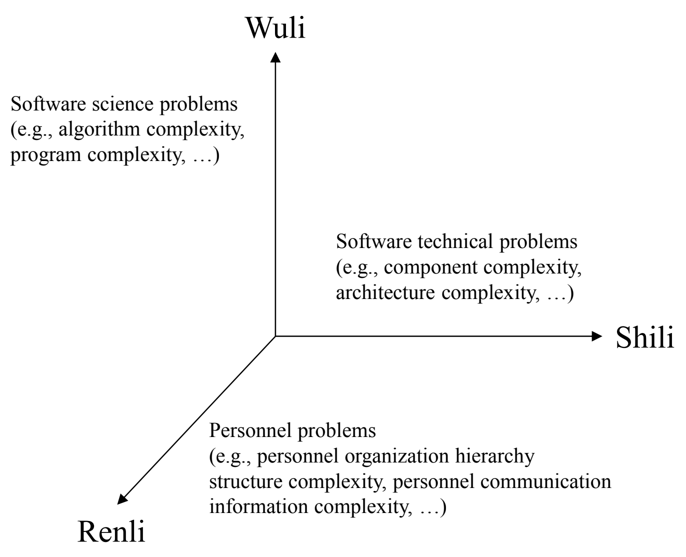
Wuli
Software science problems
(e.g., algorithm complexity,
program complexity, …)
Software technical problems
(e.g., component complexity,
architecture complexity, …)
Shili
Personnel problems
(e.g., personnel organization hierarchy
structure complexity, personnel communication
information complexity, …)
Renli

Figure 2. A WSR framework of software complexity.

Therefore, software complexity can be defined as follows.

Definition 1. *Software_Complexity = <Software_Wuli_Complexity, Software_Shili_Complexity, Software_Renli_Complexity>.*

Because the staff in different roles are concerned with different sides in the process of software development, it is conducive to understand and control the complexity better after dividing software development into the three levels of WL complexity, SL complexity and RL complexity. A good software science worker should have a good command of WL, be familiar with SL and have knowledge of RL, but focus on WL complexity; a good software engineer should be familiar with WL, have a good command of SL and have knowledge of RL, but focus on SL complexity; a software project manager should have knowledge of WL, be familiar with SL and have a good command of RL, but focus on SL complexity.

4. Software Project Renli Complexity and Its Measurement

Software projects and software have both connections and differences, and software is the final result of a software project. It can be seen from the definition of [12] that software complexity should contain the complexity of the software project obviously. The level of complexity of the software project is inversely proportional to the controllable degree of the project. The research on project complexity can provide an important basis and reference for project management and decision-making, but there is little research on the complexity of software projects. Statistics show that the vast majority of software project failures are caused by non-technical factors, namely non-WL or SL complexity. Research shows that the reason for delayed delivery of 45% of software projects is related to personnel organization [35], while man is the main cause of software development complexity [12]. The biggest complexity of software project management lies in the personnel management, but research on RL complexity is the weakest. Therefore, there is important practical significance to understanding, measuring, controlling, managing and reducing the complexity of a software project. Therefore, this section is devoted to discussing the RL complexity of the software project.

4.1. Software Project RL Complexity

In system science, the system is not classified according to the number of subsystems in the system, but by the hierarchical relationship of subsystem components. Simple giant systems do not differ much in the number of subsystems from complex giant systems, and the interaction between subsystems is little. In some complex giant systems, the interaction between subsystems is not complicated, but the nature of the whole system is very complex, rich and colorful, the difference of which is mainly caused by the hierarchy, namely the presence of a hierarchical structure is one of the main symbols of the complexity of this system. The development of a software system is usually completed by project team members. A software project team is a small social system that belongs to one of the most complex systems and differs in its composition of man from complex giant systems. The human society has a hierarchical structure, and all systems composed of people have a hierarchical structure, which is obvious. In short, one of the main reasons resulting in system complexity is the hierarchical structure of the system, namely the levels create complexity. Therefore, the personnel organization hierarchy structure of a software project is one of the main reasons causing the RL complexity of software.

Chester I. Barnard thinks that the effective operation of an organization depends on three basic elements: common goal, cooperation intention and contact information. The core function of a manager is contact information; without information communication, there will be no organization, because the organization cannot command, guide and control individual behaviors without information communication [36]. The relationship among the staff is an important aspect of RL complexity. Any software project organization must have cooperation and leadership relationships among members, which are the basic relations among the staff. These relationships are represented by information communication among the staff, and there will be no cooperation and leadership relationships without information communication. Dr. Kathryn Schwalbe thinks that information communication among the staff is one of the three major causes of the high rates of the failure of software projects. The work in [37] stated: "communication failure is the biggest lion in the way of software project survival." Therefore, the importance of information communication is visible. However, more information communication, a more complex relationship among the staff and the average efficiency of staff development is reduced [37]. Therefore, the amount of communication information becomes an important part of the RL complexity of software.

According to statistics, staff turnover in the software industry increases from traditional 6%–10% to about 20% [38], some companies even as high as 35%; Fortune Magazine reports that staff turnover has become a culture of the IT industry, and the average in-service time of software engineers is only 13 months [39]. It is visible that the mobility and uncertainty of software project staff has become one of the basic characteristics of a software project. It is easy to manage and control certain things, and on the contrary, the bigger the mobility is, the more complex the staff management of a software project will become. Therefore, the mobility and uncertainty of the staff is also one of the important reasons for the RL complexity of software.

Therefore, in this section, software RL complexity can be defined as follows.

Definition 2. *Software Project RL Complexity: Software_Renli_Complexity = <Software_Renli_Complexity (X1), Software_Renli_Complexity (X2), Software_Renli_Complexity (X3)>, where Software_Renli_Complexity (X1) represents the personnel organization hierarchy structure, Software_Renli_Complexity (X2) represents the amount of personnel communication information, Software_Renli_Complexity (X3) represents the mobility and uncertainty of the staff and:*

$$\begin{aligned}&|Software\_Renli\_Complexity|\\&\quad = \alpha|Software\_Renli\_Complexity(X1)| + \beta|Software\_Renli\_Complexity(X2)| +\\&\quad \gamma|Software\_Renli\_Complexity(X3)|\end{aligned}$$

$Software\_Renli\_Complexity|$ denotes the RL complexity degree, $|Software\_Renli\_Complexity(X1)|$ denotes the complexity degree of the personnel organization hierarchy structure of the software project, $|Software\_Renli\_Complexity(X2)|$ denotes the complexity degree of the personnel communication information, $|Software\_Renli\_Complexity(X3)|$ denotes the degree of mobility and uncertainty of the staff, where $\alpha, \beta, \gamma \in (0,1)$ are parameters.

On the basis of existing achievements [40], in the following two sections, information entropy is used to measure the complexity of the software project personnel organization hierarchy structure of and the complexity of personnel communication information. The complexity of systems can be measured by a methodology derived from Shannon's theorem [41–46].

4.2. Measurement of the Complexity of the Software Project Personnel Organization Hierarchy Structure Based on Information Entropy

A software project is completed by the team, so how to form a team that works effectively and is easy to manage becomes the subject in need of research for software project management. In software project management, there are various organizational structure forms with their own advantages and disadvantages and suitable for different occasions. There is no absolutely superior organizational structure. Among them, the project-based organization structure and matrix organization structure are common. From the aspects of the management hierarchy and the span of control, they can also be divided into the traditional towering bureaucratic organization structure and a flat organization structure. A flat organization structure has a small management hierarchy and a short chain of command, so it may obtain a more flexible command and higher management efficiency. Generally [37,47], the project personnel organization structure is relatively simple, so each other's communication in the project team is smoother and faster. The matrix personnel organization structure is more complex, so it is more difficult to manage the project team.

This paragraph measures the complexity of the personnel organization hierarchy structure of a software project based on information entropy. Before the measurement formula of complexity is given, several concepts should be defined first.

Definition 3. *Direct control/subordinate relationship: A software project group* PG *,* $M = \{M_1, M_2, \ldots, M_n\}$*, is the set of members;* M_i *and* M_j *have a direct rank relationship with a direct communication path, where* $i, j \epsilon [1, n], i, j \epsilon N$*, that is* M_i *and* M_j *have a direct control/subordinate relationship, expressed as* $M_i \prec\succ M_j$*; if* M_i *is superior to* M_j*;* M_i *can control* M_j *directly, and* M_j *is directly subordinate to* M_i*, expressed as* $M_i \prec M_j$*,* $M_j \succ M_i$*, respectively.*

Definition 4. *Indirect control/subordinate relationship: A software project group PG, $M = \{M_1, M_2, \dots, M_n\}$, is the set of members; if $M_i <> M_j$ and $M_i \prec M_j$(namely, $M_j \succ M_i$), $M_j <> M_k$ and $M_j \prec M_k$; where $i, j, k \epsilon [1, n], i, j, k \epsilon N$; that is, M_i and M_k have a indirect control/subordinate relationship, expressed as $M_i \prec\circ\succ M_k$, M_i can control M_k indirectly, and M_k is directly subordinate to M_i, expressed as $M_i \prec\circ M_k$, $M_k \circ\succ M_i$, respectively. If $M_k <> M_l$ or $M_k \prec\circ\succ M_l$, where $l \epsilon [1, n], l \epsilon N$, then $M_i \prec\circ\succ M_l$.*

Definition 5. *Control depth: A software project group PG, $M = \{M_1, M_2, \dots, M_n\}$, is the set of members; $cd(M_i, M_j)$ is known as the depth of control of M_i and M_j; if $M_i <> M_j$, then $cd(M_i, M_j) = 1$; if $M_i \prec M_j$, $M_j \prec M_k$, then $cd(M_i, M_k) = 2$; if M_i not $<> M_j$, then $cd(M_i, M_j) = 0$; where $i, j, k \epsilon [1, n], i, j, k \epsilon N$.*

Definition 6. *Hierarchy entropy: A software project group PG, $M = \{M_1, M_2, \dots, M_n\}$, is the set of members, $\forall\, i, j \epsilon [1, n], i, j \epsilon N$; if $M_i <> M_j$ or $M_i \prec\circ\succ M_j$, then:*

$$H(M_i, M_j) = -p(M_i, M_j) ln\, p(M_i, M_j) \tag{1}$$

is known as the hierarchy entropy of M_i and $M_j$where $p(M_i, M_j) = \frac{cd(M_i, M_j)}{\sum_{i=1}^{n}\sum_{j=1}^{n} cd(M_i, M_j)}$ is the contribution ratio, and $cd(M_i, M_j) = 1$ is the depth of control of M_i and M_j.

Definition 7. *Complexity degree of personnel organization hierarchy structure: A software project group PG, $M = \{M_1, M_2, \dots, M_n\}$, is the set of members,*

$$|Software\_Renli\_Complexity(X1)| = \sum_{i=1}^{n}\sum_{j=1}^{n} H(M_i, M_j)$$

namely:

$$|Software\_Renli\_Complexity(X1)| = -\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{cd(M_i, M_j)}{\sum_{i=1}^{n}\sum_{j=1}^{n} cd(M_i, M_j)} ln \frac{cd(M_i, M_j)}{\sum_{i=1}^{n}\sum_{j=1}^{n} cd(M_i, M_j)} \tag{2}$$

is known as the complexity degree of the personnel organization hierarchy structure of the software project, where $i, j \epsilon [1, n], i, j \epsilon N$.

4.3. Measurement of the Complexity of Personnel Communication Information for Software Projects Based on Information Entropy

This section analyzes and measures the complexity of personnel relations from the amount of personnel communication information. A software project group PG, $M = \{M_1, M_2, \dots, M_n\}$, is the set of members, $\forall\, i, j \epsilon [1, n], i, j \epsilon N$. If members M_i and M_j have a direct exchange of information, there is a direct communication path between them, expressed as $M_i \leftrightarrow M_j$; otherwise expressed as $M_i \nleftrightarrow M_j$, so a connected graph can be constructed for PG.

Definition 8. *Amount of information provided by M_i for project team PG: A software project group PG, $M = \{M_1, M_2, \dots, M_n\}$, is the set of members; $I(M_i)$ is known as the amount of information provided by M_i for project team PG, where $i \epsilon [1, n], i \epsilon N$. PGG is the connected graph of project group PG on the staff, $PGG =< M, P >$; $M = \{M_1, M_2, \dots, M_n\}$ is the set of members; $P = \{P_{12}, P_{13}, \dots, P_{1n}, P_{21}, P_{23}, \dots, P_{2n}, \dots, P_{ij}, \dots, P_{(n-1)n}\}$ represents the path set, where $i, j \epsilon [1, n], i, j \epsilon N$, and $i \neq j$, P_{ij} indicates whether there is a direct communication path between two members and*

$$P_{ij} = \begin{cases} 1 \; if \; M_i \leftrightarrow M_j \\ 0 \; if \; M_i \nleftrightarrow M_j \end{cases}, i, j \epsilon [1, n], i, j \epsilon N, i \neq j$$

Then, the amount of information provided by M_i for project team PG,

$$I(M_i) = -ln \frac{\sum_{j=1}^{n} P_{ij}(i \neq j)}{\sum_{i=1}^{n} \sum_{j=1}^{n} P_{ij}(i \neq j)} \tag{3}$$

Definition 9. *Complexity degree of personnel communication information: A software project group PG, $M = \{M_1, M_2, \dots, M_n\}$, is the set of members,*

$$|Software\_Renli\_Complexity(X2)| = \sum_{i=1}^{n} I(M_i)$$

namely,

$$|Software\_Renli\_Complexity(X2)| = -\sum_{i=1}^{n} \sum_{j=1}^{n} \frac{P_{ij}(i \neq j)}{\sum_{i=1}^{n} \sum_{j=1}^{n} P_{ij}(i \neq j)} ln \frac{P_{ij}(i \neq j)}{\sum_{i=1}^{n} \sum_{j=1}^{n} P_{ij}(i \neq j)} \tag{4}$$

is known as the complexity degree of personnel communication information for software projects, where $i, j \epsilon [1, n], i, j \epsilon N$.

5. Case Analysis

5.1. Measurement of the Complexity of Software Project Personnel Organization Hierarchy Structure

5.1.1. Cases

(1) Case 1: Any software project organization structure is hierarchical. According to the Capability Maturity Model (CMM) organization model, a software project organization sets up a software engineering process group (SEPG), a software quality assurance group (SQAG) and a software engineering group (SEG) to form a system of checks and balances of legislation, supervision and enforcement. The personnel hierarchy structure is shown in Figure 3, and then, the complexity of the personnel organization hierarchy is calculated.

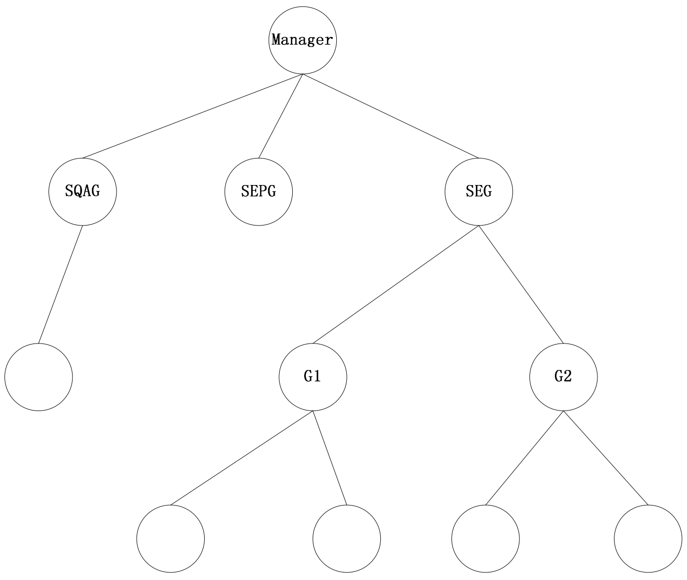

Figure 3. Personnel hierarchy structure (Case 1).

According to Definition 5, the depth of control of M_i and M_j is shown in Table 3.

Table 3. Control depth.

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}
M_1	0	1	1	1	2	2	2	3	3	3	3
M_2	1	0	0	0	1	0	0	0	0	0	0
M_3	1	0	0	0	0	0	0	0	0	0	0
M_4	1	0	0	0	0	1	1	2	2	2	2
M_5	2	1	0	0	0	0	0	0	0	0	0
M_5	2	0	0	1	0	0	0	1	1	0	0
M_7	2	0	0	1	0	0	0	0	0	1	1
M_8	3	0	0	2	0	1	0	0	0	0	0
M_9	3	0	0	2	0	1	0	0	0	0	0
M_{10}	3	0	0	2	0	0	1	0	0	0	0
M_{11}	3	0	0	2	0	0	1	0	0	0	0

According to Formula (1), $H(M_i, M_j) = -p(M_i, M_j)\ln p(M_i, M_j)$, the hierarchy entropy of M_i and M_j is shown in Tables 4 and 5.

Table 4. Contribution ratio.

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}
M_1	0	0.013889	0.013889	0.013889	0.027778	0.027778	0.027778	0.041667	0.041667	0.041667	0.041667
M_2	0.013889	0	0	0	0.013889	0	0	0	0	0	0
M_3	0.013889	0	0	0	0	0	0	0	0	0	0
M_4	0.013889	0	0	0	0	0.013889	0.013889	0.027778	0.027778	0.027778	0.027778
M_5	0.027778	0.013889	0	0	0	0	0	0	0	0	0
M_5	0.027778	0	0	0.013889	0	0	0	0.013889	0.013889	0	0
M_7	0.027778	0	0	0.013889	0	0	0	0	0	0.013889	0.013889
M_8	0.041667	0	0	0.027778	0	0.013889	0	0	0	0	0
M_9	0.041667	0	0	0.027778	0	0.013889	0	0	0	0	0
M_{10}	0.041667	0	0	0.027778	0	0	0.013889	0	0	0	0
M_{11}	0.041667	0	0	0.027778	0	0	0.013889	0	0	0	0

Table 5. Hierarchy entropy.

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}
M_1	0	0.059398	0.059398	0.059398	0.099542	0.099542	0.099542	0.132419	0.132419	0.132419	0.132419
M_2	0.059398	0	0	0	0.059398	0	0	0	0	0	0
M_3	0.059398	0	0	0	0	0	0	0	0	0	0
M_4	0.059398	0	0	0	0	0.059398	0.059398	0.099542	0.099542	0.099542	0.099542
M_5	0.099542	0.059398	0	0	0	0	0	0	0	0	0
M_5	0.099542	0	0	0.059398	0	0	0	0.059398	0.059398	0	0
M_7	0.099542	0	0	0.059398	0	0	0	0	0	0.059398	0.059398
M_8	0.132419	0	0	0.099542	0	0.059398	0	0	0	0	0
M_9	0.132419	0	0	0.099542	0	0.059398	0	0	0	0	0
M_{10}	0.132419	0	0	0.099542	0	0	0.059398	0	0	0	0
M_{11}	0.132419	0	0	0.099542	0	0	0.059398	0	0	0	0

According to Formula (2), calculate the complexity of the personnel organization hierarchy,

$$|Software\_Renli\_Complexity(X1)|_1 = 3.640904786.$$

(2) Case 2: Based on Case 1, this case combines Group G1 and Group G2 of SEG into one group with the remaining unchanged, and the number of members of the whole project group is identical to that in Case 1. Figure 4 shows the project personnel organization hierarchy. Compared with the bureaucratic personnel organization hierarchy in Case 1, the software project organization structure tends to be more flat. Then, calculate the complexity of the personnel organization hierarchy.

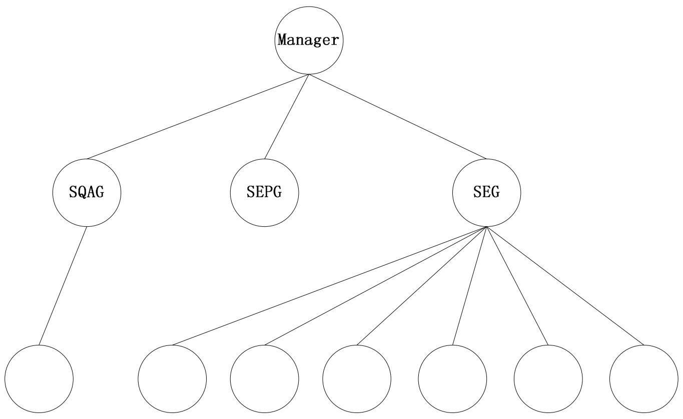

Figure 4. Personnel hierarchy structure (Case 2).

According to Formula (2), calculate the complexity of the personnel organization hierarchy,

$$|Software\_Renli\_Complexity(X1)|_2 = 3.275571498.$$

(3) Case 3: Software enterprise C mainly specializes in software development and adopts the matrix organization structure. Figure 5 shows the project personnel organization hierarchy in which the shaded background refers to software project personnel P. Then, the complexity of the project personnel organization hierarchy is calculated.

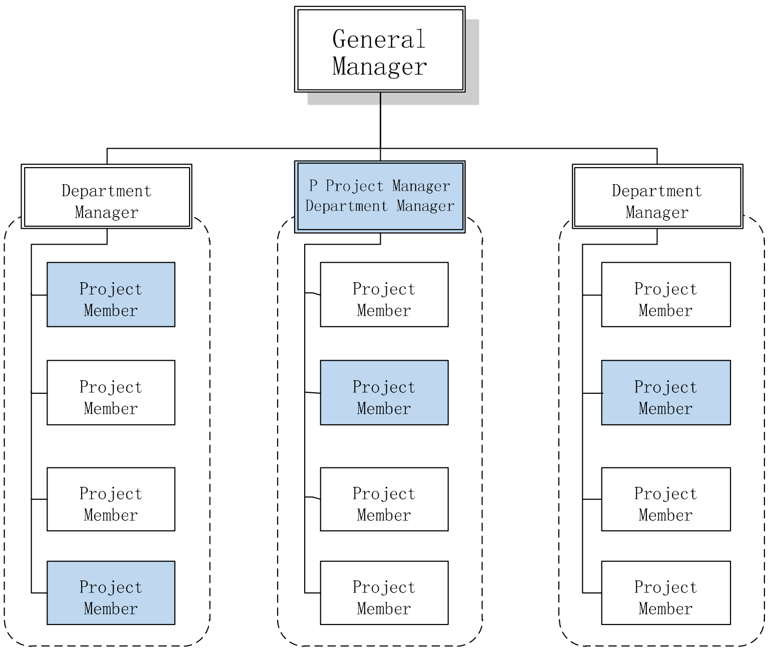

Figure 5. Personnel hierarchy structure (Case 3).

According to Formula (2), calculate the complexity of the personnel organization hierarchy,

$$|Software\_Renli\_Complexity(X1)|_3 = 3.275453525.$$

(4) Case 4: Because project P brings large risks, the quality, schedule and cost requirements are higher. In order to adapt to the environment, enterprise C undergoes restructuration and adjusts the original matrix organization structure to the project-based organizational structure. The

project organization personnel hierarchy is shown in Figure 6, and project P still maintains the original number and personnel. Then, the complexity of the project personnel organization hierarchy is calculated.

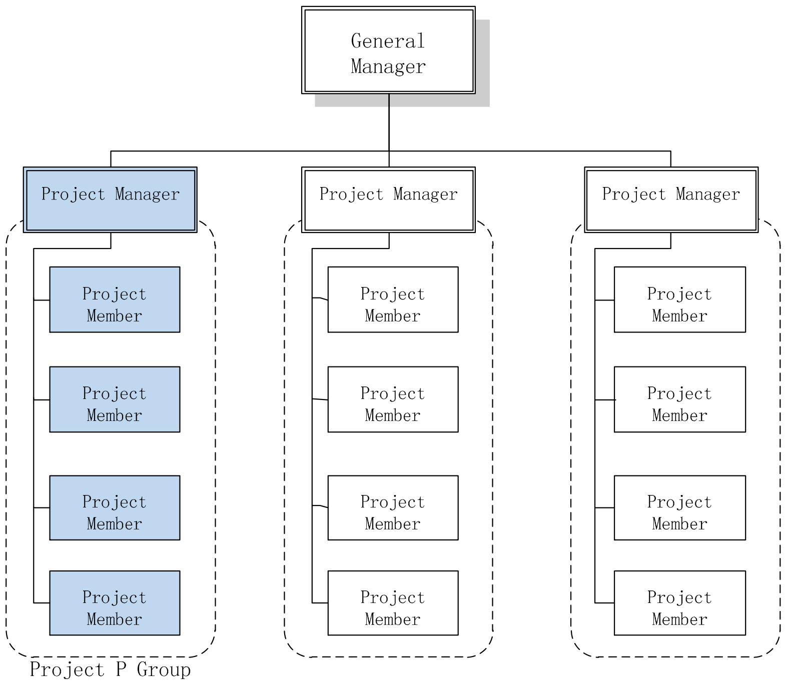

Figure 6. Personnel hierarchy structure (Case 4).

According to Formula (2), calculate the complexity of personnel organization hierarchy,

$$|Software\_Renli\_Complexity(X1)|_4 = 2.831544427.$$

5.1.2. Contrast Analysis

(1) Analysis of the Contrast between Case 1 and Case 2.

Two cases have exactly an identical number of project personnel. Case 1 is a traditional bureaucratic project organization structure, including four hierarchies, while Case 2 is a flat project organizational structure, including three hierarchies only. By measurement according to Formula (2), the results show that,

$$|Software\_Renli\_Complexity(X1)|_1 > |Software\_Renli\_Complexity(X1)|_2$$

The complexity of the personnel organization hierarchy is reduced from 3.640904786 to 3.275571498 with a difference of 0.365333288.

Conclusion: Compared with the bureaucratic organization structure, flat management reduces the complexity of personnel organization hierarchy.

(2) Analysis of Contrast between Case 3 and Case 4.

These two cases are identical in the enterprise, project, number of project personnel, project manager and developers, but only differ in the personnel organization structure, which has been changed from the matrix organization structure to the project-based organization structure. By measurement according to Formula (2), the results show that,

$$|Software\_Renli\_Complexity(X1)|_3 > |Software\_Renli\_Complexity(X1)|_4$$

The complexity of the personnel organization hierarchy is reduced from 3.275453525 to 2.831544427 with a difference of 0.4439099098.

Conclusion: Because there are many leaders of software project personnel in the matrix organization structure (department manager, project manager), more collaboration between departments is needed than in the project-based organization structure; and the complexity of the personnel organization hierarchy is higher, but all members of the project-based organization belong to the same department, leading to a simple structure and a lower complexity of the personnel organization hierarchy.

5.2. Measurement of the Complexity of the Personnel Communication Information for Software Projects

5.2.1. Cases

(1) Case 5: For a software project group, its personnel organization and information interaction is shown in Figure 7, then the complexity of the personnel communication information is calculated.

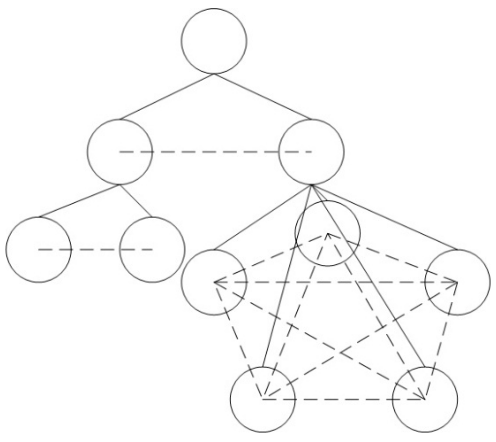

Figure 7. Personnel information interaction (Case 5).

According to Definition 8, $M = \{M_1, M_2, \ldots, M_{10}\}$, $P = \{P_{0102}, P_{0103}, \ldots, P_{0110}, P_{0201}, P_{0203}, \ldots, P_{0210}, \ldots, P_{1001}, P_{1002}, \ldots, P_{1010}\}$, and

$$P_{ij} = \begin{pmatrix} & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & \end{pmatrix}$$

According to Formula (4):

$$|Software\_Renli\_Complexity(X2)|_5 = 2.224303443$$

(2) Case 6: Figure 8 shows the personnel organization and information interaction. It has the same organization structure as Case 5; however, it has one more person.

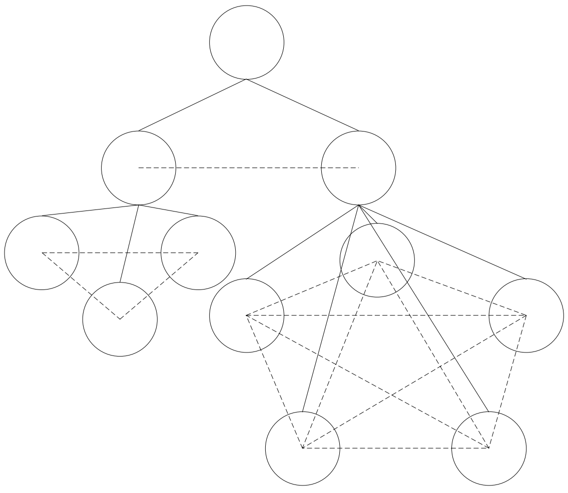

Figure 8. Personnel information interaction (Case 6).

According to Formula (4):

$$|Software\_Renli\_Complexity(X2)|_6 = 2.346652816$$

(3) Case 7: After readjustment of the structure of the project group in Case 5, the personnel organization and information interaction is shown in Figure 9. Compared with Case 5, the personnel organization structure management tends to be more flat, and then, the complexity of the personnel communication information is calculated.

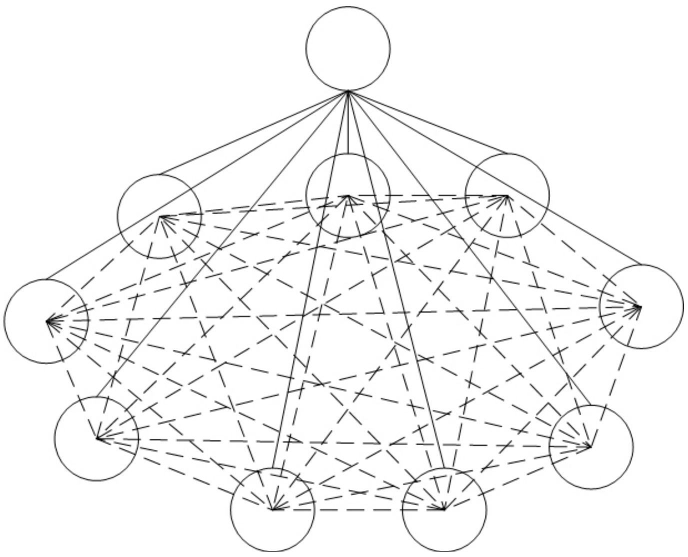

Figure 9. Personnel information interaction (Case 7).

According to Formula (4):

$$|Software\_Renli\_Complexity(X2)|_7 = 2.302585093$$

(4) Case 8*:* Calculate the complexity of the personnel communication information based on the matrix organization structure in Case 3.

According to Formula (4):

$$|Software\_Renli\_Complexity(X2)|_8 = 2.602876917$$

(5) Case 9: Calculate the complexity of the personnel communication information based on the project-based organization structure in Case 4.

According to Formula (4):

$$|Software\_Renli\_Complexity(X2)|_9 = 1.717047029$$

5.2.2. Contrast Analysis

(1) Analysis of Contrast between Case 5 and Case 6

Case 5 and Case 6 have the same organization structure, but Case 6 has one more person. By measurement according to Formula (4), the results show that,

$$|Software\_Renli\_Complexity(X2)|_5 < |Software\_Renli\_Complexity(X2)|_6$$

the complexity of the personnel communication information increases from 2.224303443 to 2.346652816 with a difference of 0.122349373.

Conclusion: In the same organization structure cases, the more people, the higher the complexity of the personnel communication information.

(2) Analysis of Contrast between Case 5 and Case 7

The above two cases are identical in the number of members of the project group, but differ in the personnel organization structure. Case 5 has three hierarchies, and Case 7 has two hierarchies. The former is taller than the latter in structure, and the latter is more flat than the former in structure. By measurement according to Formula (4), the results show that,

$$|Software\_Renli\_Complexity(X2)|_5 > |Software\_Renli\_Complexity(X2)|_7$$

When the personnel organization structure is changed from a bureaucratic organization structure to a flat organization structure, the complexity of the personnel communication information increases from 2.224303433 to 2.302585093 with a difference of 0.07828165.

Conclusion: The large span of control of the flat management organization increases the horizontal information exchanges and the complexity of the personnel communication information.

(3) Analysis of Contrast between Case 8 and Case 9

Case 8 shows a matrix project personnel organization structure, and the latter is the project-based personnel one; by measurement according to Formula (4), the results show that,

$$|Software\_Renli\_Complexity(X2)|_8 > |Software\_Renli\_Complexity(X2)|_9$$

the complexity of the personnel communication information is reduced from 2.602876917 to 1.717047029 with a difference of 0.885829888.

Conclusion: Compared with the project-based organization structure, the matrix software project organization structure has a higher complexity of personnel communicate information under the same scale of personnel due to the existence of multiple leadership roles.

6. Summary

Excessive complexity is one of the main reasons for the failure of software projects. To the extent that the complexity is beyond the control of software project managers, the project's failure is inevitable. Therefore, how to recognize, measure, manage, control and reduce software complexity becomes one of the key problems of software project management. However, this is also a very challenging research topic. At first, this paper analyzes the current research situation of software complexity systematically and points out existing problems in current research. Then, based on system science, it proposes a WSR framework of software complexity, which divides the complexity of software into the three levels of WL, SL and RL, so that the staff in different roles may have a better understanding of the complexity. Because man is the main source of complexity and personnel is the core problem of software project management, the current research focuses on WL complexity; additionally, the research of RL complexity is extremely scarce, so this paper emphasizes the research on the RL complexity of software projects. This paper not only analyzes the composing factors of RL complexity, but also provides the definition of RL complexity. Moreover, it puts forward a quantitative measurement method of the complexity of personnel organization hierarchy and the complexity of communication information based on information entropy, first, and analyzes and validates the scientificity and rationality of this measurement method through a large number of cases. Case analysis indicates that, in the same organization structure cases, the more people, the higher the complexity of personnel communication information. Case analysis also shows that flat management reduces the complexity of the personnel organization hierarchy, but increases the complexity of personnel communication information by comparison with a towering bureaucratic project organization structure. However, project-based personnel organization structure is lower than the matrix software project organization structure in both the complexity of the personnel organization hierarchy and the complexity of the personnel communication information; so it is simpler, but the management and communication of the matrix project organization is very complex, because of high complexity.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant Nos. 61263022 and 61303234), the National Social Science Foundation of China (Grant No. 12XTQ012), the Humanities and Social Sciences Youthful Foundation of the Ministry of Education of China (Grant No. 11YJCZH073), the Natural Science Foundation of Yunnan Province of China (Grant No. 2010ZC100), and the Introduction of Talents Project of Science Research Foundation of Yunnan University of Finance and Economics (Grant No. YC2012D07). The author would like to thank the anonymous reviewers and the editors for their suggestions.

Conflicts of Interest

The author declares no conflict of interest.

References

1. Brooks, F.P. No silver bullet: Essence and accidents of software engineering. *IEEE Comput.* **1987**, *20*, 10–19.
2. Cerpa, N.; Verner, J.M. Why did your project fail? *Commun. ACM* **2009**, *52*, 130–134.
3. Vidal, L.A.; Marle, F.; Bocquet, J.C. Using a Delphi process and the Analytic Hierarchy Process (AHP) to evaluate the complexity of projects. *Expert Syst. Appl.* **2011**, *38*, 5388–5405.
4. Vidal,L.A.; Marle, F.; Bocquet, J.C. Measuring project complexity using the Analytic Hierarchy Process. *Int. J. Proj. Manag.* **2011**, *29*, 718–727.
5. Williams, T.M. Assessing and moving on from the dominant project management discourse in the light of project overruns. *IEEE Trans. Eng. Manag.* **2005**, *52*, 497–508.
6. Bosch-Rekveldt, M.; Jongkind, Y.; Mooi, H.; Bakker, H.; Verbraeck, A. Grasping project complexity in large engineering projects: The TOE (Technical, Organizational and Environmental) framework. *Int. J. Proj. Manag.* **2011**, *29*, 728–739.
7. Giezen, M. Keeping it simple? A case study into the advantages and disadvantages of reducing complexity in mega project planning. *Int. J. Proj. Manag.* **2012**, *30*, 781–790.
8. Green, G.C. The impact of cognitive complexity on project leadership performance. *Inf. Softw. Technol.* **2004**, *46*, 165–172.
9. Xu, G.Z.; Gu, J.F.; Che, H.A. *Systems Science*; Shanghai Education Press of Science and Technology: Shanghai, China, 2000. (in Chinese)
10. Brooks, P.F., Jr. No silver bullet: Essence and accidents of software engineering. *IEEE Comput.* **1987**, *20*, 10–19.
11. Zuse, H. *Software Complexity Measures and Models*; De Gruyter: New York, NY, USA, 1990.
12. Lin, Z.K.; Yang, D.L.; Yang, H. Six elements structure model and complexity analysis for software development system. *J. Syst. Eng.* **2006**, *21*, 368–374.
13. Jiang, R.; Yang, M. Survey on Software Complexity Research. *Comput. Syst. Appl.* 2014, 23. Available online: http://www.c-s-a.org.cn/ch/reader/view_abstract.aspx?file_no=20140901&flag=1 (accessed on 2 April 2015). (in Chinese)
14. McCabe, T.J. A Complexity Measurement. *IEEE Trans. Softw. Eng.* **1976**, *2*, 302–308.
15. Halstead, M.H. *Elements of Software Science*; Elsevier: New York, NY, USA, 1977.
16. Harrison, W.M.; Magel, K.I. A Complexity Measure Based on Nesting Level. *ACM Sigplan Not.* **1981**, *6*, 63–74.
17. Chidamber, S.R.; Kemerer, C.F. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.* **1994**, *20*, 476–493.
18. Lun, L.J.; Ding, X.M.; Li, Y.M.; Zhang, Y. Research on Object-Oriented Software Complexity Metrics Based on Inheritance Graph. *Comput. Eng. Appl.* **2006**, *27*, 93–95.
19. Chen, J.-Y.; Lu, J.-F. A New Metric for Objec-Oriented Design. *Inf. Softw. Technol.* **1993**, *35*, 232–240.

20. Roca, J.L. An entropy-based method for computing software structural complexity. *Microelectron. Reliab*. **1996**, *36*, 609–620.
21. Chhabra, J.K.; Aggarwal, K.K.; Singh, Y. Measurement of object-oriented software spatial complexity. *Inf. Softw. Technol.* **2004**, *46*, 689–699.
22. Jiao, F.; Wang, L.; Hou, J. dependency-matrix based metric model for complexity of component-based software. *Comput. Appl. Softw.* **2009**, *26*, 55–56.
23. Jiao, F.; Wang, L.; Hou, J. Approach to eValuating for complexity of software architecture. *Appl. Res. Comput.* **2008**, *25*, 2377–2379.
24. Huang, W.; Chen, S. Software architecture and its complexity metrics based on component operations. *Comput. Eng. Appl.* **2007**, *43*, 66–70.
25. Li, B.; Wang, H.; Li, Z.; He, K.; Yu, D. Software complexity metrics based on complex networks. *Chin. J. Electron.* **2006**, *34*, 2371–2375.
26. Zhang, S. Evaluation of software requirement complexity and testing method of its effectiveness. *China Railw. Sci.* **2002**, *23*, 30–35.
27. Wang, Y.; Zhang, X. Estimating complexity of use cases driven software and its application. *Comput. Eng. Design* **2007**, *28*, 2543–2546.
28. Jiang, G.; Chen, Y. Software projeet complexity evaluation based on evidence reasoning. *Comput. Eng. Appl.* **2005**, *2*, 4–7.
29. Jung, W.S.; Lee, E.J.; Kim, K.S.; Wu, C.S. an entropy-based complexity measure for web applications using structural information. *J. Inf. Sci. Eng.* **2011**, *27*, 595–619.
30. Jung, W.S.; Lee, E.J.; Kim, K.S.; Wu, C.S. A complexity metric for web applications based on the entropy theory. In Proceedings of the 15th Asia-Pacific Software Engineering Conference (APSEC), Beijing, China, 3–5 December 2008.
31. Hops, J.M.; Sherif, J.S. Development and application of composite complexity models and a relative complexity metric in a software maintenance environment. *J. Syst. Softw.* **1995**, *31*, 157–169.
32. Gu, J.F. *Wuli-Shili-Renli System Approach: Theory and Applications*; Shanghai Education Press of Science and technology: Shanghai, China, 2006. (in Chinese)
33. Gu, J.F.; Zhu, Z. The Wu-li Shi-li Ren-li approach(WSR): An oriental systems methodology. In *Midgley GL and Wiley Jeds. Systems Methodology I: Possibilities for Cross-Cultural Learning and Integration*. University of Hull: Kingston upon Hull, UK, 1995; pp. 29–38.
34. Zhang, C.; Sun, D. Some Concepts and Understandings about WSR. *Syst. Eng.* **2001**, *19*, 1–8. (in Chinese)
35. Mao, M.; Ge, X. Research on risk management model of software project. *Sci. Technol. Manag. Res.* **2005**, *6*, 148–151.
36. Markus, M.L.; Robey, D. Information technology and organizational change:casual structrure in theory and research. *Manag. Sci.* **1988**, *28*, 583–593.
37. Guo, N. *IT Project Management*; Tsinghua University Press: Beijing, China, 2009. (in Chinese)
38. Maddem, S. High-tech Brain Drain. *Am. Netw.* **2000**, *104*, 70–72.
39. Daniels, C.; Vinzant, C. The Joy of Quitting. *Fortune*, **2000**, *141*, 199–202.
40. Qiu, W. *Management Decisions Entropy and the Application*; China Electric Power Press: Beijing, China, 2011. (in Chinese)

41. Menhorn, B.; Slomka, F. Design entropy concept: a measurement for complexity. In Proceedings of the 7th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. Taipei, Taiwan, 9–14 October 2011; pp. 285–294.
42. Menhorn, B.; Slomka, F. Quantitative Analysis of Software Code by States. In Proceedings of the 8th IASTED International Conference on Advances in Computer Science, Phuket, Thailand, 10–12 April 2013.
43. Menhorn, B.; Brix, L.; Slomka, F. Digital hardware projects: A new tool for automated complexity analysis. In Proceedings of the 8th IEEE International Symposium on Industrial Embedded Systems, Porto, Portugal, 19–21 June 2013.
44. Menhorn, B.; Slomka, F. Confirming the design gap. In *Advances in Computational Science, Engineering and Information Technology*; Nagamalai, D., Kumar, A., Annamalai, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2013.
45. Menhorn, B.; Slomka, F. States and complexity. In Proceedings of the First International Conference on Coping with Complexity, Cluj-Napoca, Romania, 19–20 October 2011.
46. Menhorn, B.; Slomka, F. Project Management through States. In Proceedings of International Conference on Management and Service Science, Wuhan, China, 20–22 September 2009.
47. Han, W.; Jiang, L. *Software Project Management Case Course*, 2nd ed.; China Machine Press: Beijing, China, 2009. (in Chinese)